HITACHI
Inspire the Next

uCosminexus Application Server

Expansion Guide

3021-3-J08-10(E)

Notices

■ Relevant program products

See the *Release Notes*.

■ 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

HITACHI, Cosminexus, DABroker, HA Monitor, HiRDB, JP1, OpenTP1, TPBroker, XDM are either trademarks or registered trademarks of Hitachi, Ltd. in Japan and other countries.

AIX is a trademark of International Business Machines Corporation, registered in many jurisdictions worldwide.

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

Microsoft, Active Directory are trademarks of the Microsoft group of companies.

Microsoft, SQL Server are trademarks of the Microsoft group of companies.

Microsoft, Windows are trademarks of the Microsoft group of companies.

Microsoft, Windows Server are trademarks of the Microsoft group of companies.

Microsoft is a trademark of the Microsoft group of companies.

Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

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

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

SPARC is a registered trademark of SPARC International, Inc. Products bearing SPARC trademarks are based on an architecture developed by Sun Microsystems, Inc

UNIX is a trademark of The Open Group.

Other company and product names mentioned in this document may be the trademarks of their respective owners.

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

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

■ Microsoft product screen shots

Microsoft product screen shots reprinted with permission from Microsoft Corporation.

■ Issued

Aug. 2022: 3021-3-J08-10(E)

■ Copyright

Preface

For details on the prerequisites before reading this manual, see the *Release Notes*.

■ Non-supported functionality

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

- Audit log functionality
- Compatibility functionality
- Cosminexus Component Transaction Monitor
- Cosminexus Reliable Messaging
- Cosminexus TPBroker and VisiBroker
- Cosminexus Web Service - Security
- Cosminexus XML Security - Core functionality
- JP1 linkage functionality
- Management Server management portal
- Remote installation functionality for the UNIX edition
- SOAP applications complying with specifications other than JAX-WS 2.1
- uCosminexus OpenTP1 linkage functionality
- Virtualized system functionality
- XML Processor high-speed parse support functionality

■ Non-supported compatibility functionality

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

- Basic mode
- Check of JSP source compliance (cjjsp2java) with JSP1.1 and JSP1.2 specifications
- Database connection using Cosminexus DABroker Library
- EJB client application log subdirectory exclusive mode
- J2EE application test functionality
- Memory session failover functionality
- Servlet engine mode
- Simple Web server functionality
- Switching multiple existing execution environments
- Using EJB 2.1 and Servlet 2.4 annotation

Contents

1 Application Server Functionality

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

1.1 Classification of functionality

Application Server is a product for building an application execution environment based on a J2EE server that supports Java EE 7 and for developing applications that can operate in the execution environment. You can use a variety of functionality such as the functionality compliant with the 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 an excellent processing performance.

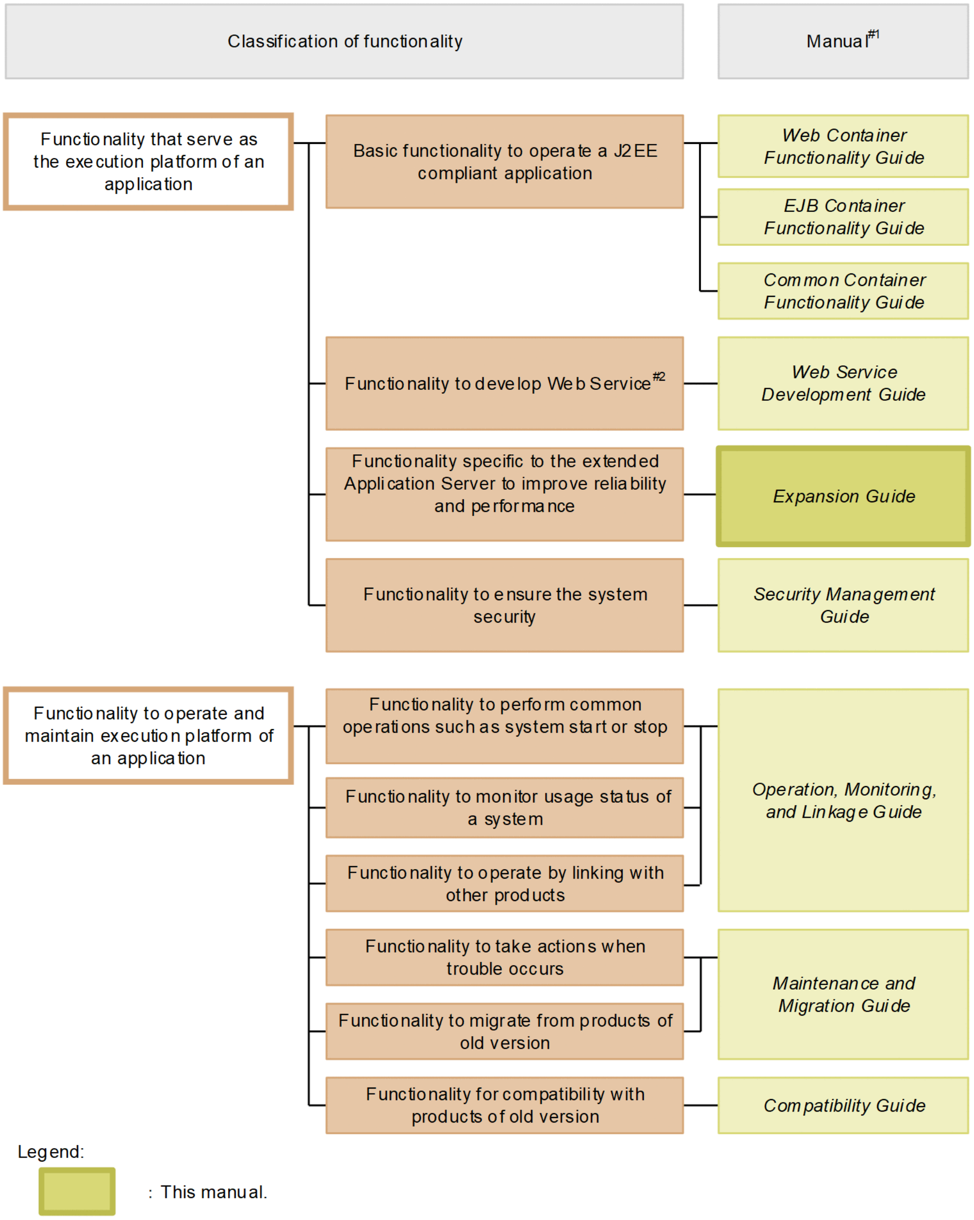
The following are the broad classifications of the Application Server functionality:

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

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



#1

uCosminexus Application Server has been omitted from the manual names.

#2

With Application Server, you can execute SOAP Web Services and RESTful Web Services. Depending on the purpose, see the following manuals except the *uCosminexus Application Server Web Service Development Guide*.

When developing and executing SOAP applications

- *uCosminexus Application Server SOAP Application Development Guide*

When ensuring security for SOAP Web Services and SOAP applications

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

For details on the XML processing

- *uCosminexus Application Server XML Processor User Guide*

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

1.1.1 Functionality as an application execution platform

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

You must examine whether you want to use the functionality that serves as an execution platform for applications, even before you perform the system building or application development.

This subsection describes the functionality that serves as a platform for executing applications, according to their classification.

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

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

Application Server provides a J2EE server that supports Java EE 7. The J2EE server provides a functionality that is compliant with the standard specifications and provides a functionality unique to Application Server.

The basic development functionality can be further classified into three types according to the types of J2EE applications for which you use the functionality. The manuals for the Application Server functionality have been divided 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 (Common container functionality)**
  This classification includes the functionality that can be used in the Web applications and the Enterprise Beans running on the Web and EJB containers respectively.

(2) Functionality for developing Web Services

This classification includes the functionality that serves as an execution environment and a development environment of Web Services.

Application Server provides the following engines:

- A JAX-WS engine that binds the SOAP messages in accordance with the JAX-WS specifications

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

(3) Functionality unique to Application Server, which are extended for improving reliability and performance (Extended functionality)

This classification includes the functionality that is independently extended on Application Server. This classification also includes the functionality implemented by using the non-J2EE server processes such as a batch server, CTM, and a database.

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

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

This classification includes the functionality for ensuring the security of an Application Server-based system. This classification includes the functionality such as the authentication functionality used for preventing unauthorized access and the encryption functionality used for preventing information leakage from communication channels.

1.1.2 Functionality for operating and maintaining an application execution platform

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

This subsection describes the functionality for operating and maintaining an application execution platform, according to their classification.

(1) Functionality used for daily operations, such as starting and stopping a system (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 that is used to output the 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 that is implemented by linking with other products, such as JP1 and cluster software.

(4) Functionality for troubleshooting (Maintenance functionality)

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

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

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

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

This classification includes the functionality used for maintaining the compatibility with older versions of Application Server. For the compatibility functionality, we recommend the migration with the corresponding recommended functionality.

1.1.3 Functionality and corresponding manuals

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

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

Table 1–1: Classification of functionality and corresponding manuals

Classification	Functionality	Reference manual[#1]
Basic development functionality	Web container	*Web Container Functionality Guide*
	Using JSF and JSTL	
	Using JAX-RS 2.0	
	WebSocket	
	NIO HTTP Server	
	Servlet and JSP implementation	
	EJB container	*EJB Container Functionality Guide*
	EJB client	
	Points to be considered when implementing Enterprise Beans	
	Naming management	*Common Container Functionality Guide*
	Managing resource connection and transactions	
	Invoking Application Server from OpenTP1 (TP1 inbound integrated function)	
	Using JPA 2.1	
	CJMS provider	
	Using JavaMail	
	Using CDI with Application Server	

Classification	Functionality	Reference manual[#1]
	Using Bean Validation with Application Server	
	Java Batch	
	JSON-P	
	Concurrency Utilities	
	Managing application attributes	
	Using annotations	
	Format and deployment of J2EE applications	
	Container extension library	
Extended functionality	Executing applications using batch servers	*Expansion Guide*[#2]
	Scheduling and load balancing requests using CTM	
	Scheduling batch applications	
	Inheriting the session information between J2EE servers (Session failover functionality)	
	The database session failover functionality	
	Suppression of Full GC by using the Explicit Memory Management functionality	
	Output of application user logs	
Security management functionality	Authentication using integrated user management	*Security Management Guide*
	Authentication using application settings	
	Using TLSv1.2 in the SSL/TLS communication	
	Controlling with the management functionality of load balancers that use API-based direct connections	
Operation functionality	Starting and stopping systems	*Operation, Monitoring, and Linkage Guide*
	Operating J2EE applications	
Watch functionality	Monitoring statistics (Statistics collection functionality)	
	Monitoring resource exhaustion	
	Audit log output functionality	
	Database audit trail linkage functionality	
	Output of statistical information using management commands	
	Reporting of Management event and auto execution of process by management action	
	Collecting statistics of CTM	
	Output of console logs	
Linkage functionality	Operation of systems linked with JP1	
	Centralized monitoring of the system (integrating with JP1/IM)	
	Automatic operation of the system by using jobs (integrating with JP1/AJS)	
	Collecting and consolidating audit logs (integrating with JP1/Audit Management - Manager)	

Classification	Functionality	Reference manual[#1]
	Linking with cluster software	
	BASIC authentication (integrating with cluster software)	
	Mutual node switching system (integrating with cluster software)	
	N-to-1 recovery system (integrating 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	
	Product Java VM (Hereafter, it might be abbreviated as Java VM) functionality	
Migration functionality	Migrating from an older version of Application Server	
	Migrating to a recommended functionality	
Compatibility functionality	Compatibility functionality for the basic development functionality	*Compatibility Guide*
	Compatibility functionality for the extended functionality	

#1

uCosminexus Application Server has been omitted from the manual names.

#2

This manual.

1.2 Functionality corresponding to the purpose of the system

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

This subsection describes which functionality, from among the functionality extended on Application Server, is best used in which system. The functionality-wise mapping is described on the basis of the following points:

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

The functionality expanded on Application Server include the Java EE standard functionality and the functionality independently expanded on Application Server. When you choose the functionality, you confirm the compliance with the Java EE standards, as and when required.

1.2.1 Functionality used when executing batch applications

The following table describes the functionality used when executing batch applications. Select the functionality according to the purpose of the system. For details on the functionality, see *Reference location* in the table.

Table 1–2: Corresponding functionality used when executing batch applications and the purpose of systems

Functionality		Purpose of system					Compliance with Java EE Standard		Reference location
		Reliability	Performance	Operation and maintenance	Scalability	Others	Standard	Extended	
Functionality for executing batch applications	Executing batch applications	--	--	--	--	--	--	Y	2.3.1, 2.3.2
	Forcefully stopping batch applications	--	--	Y	--	--	--	Y	2.3.3

Functionality		Purpose of system					Compliance with Java EE Standard		Reference location
		Reliability	Performance	Operation and maintenance	Scalability	Others	Standard	Extended	
	Displaying a list of the batch application information	--	--	Y	--	--	--	Y	*2.3.4*
	Output of batch application logs	--	--	Y	--	--	--	Y	*2.3.5*
EJB access functionality	Invoking Enterprise Beans	--	--	--	Y	--	Y	Y	*2.4*
	Lookup of EJB home objects and references of business interfaces by using JND	--	--	--	Y	--	Y	Y	
	Implementing transactions	--	--	--	Y	--	Y	Y	
	Timeout for the RMI-IIOP communication	--	--	--	Y	--	Y	Y	
	Acquiring RMI-IIOP stubs and interfaces	--	--	--	Y	--	Y	Y	
Functionality provided by naming management	Lookup and binding of the objects to the JNDI name space	--	--	--	Y	--	Y	Y	*2.5*#
	Assigning optional names to J2EE resources (user specification name space functionality)	--	--	--	Y	--	--	Y	
	Searching CORBA Naming Service by using the round-robin policy	--	--	--	Y	--	--	Y	
	Caching with the naming management functionality	--	Y	--	--	--	--	Y	
	Switching CORBA Naming Service	--	--	--	Y	--	--	Y	
Functionality provided by resource connections and transaction management	Connection pooling	--	Y	--	--	--	Y	Y	*2.7*, *2.8*
	Warming up of connection pool	--	Y	--	--	--	--	Y	
	Functionality for adjusting the number of connection pools	--	Y	--	--	--	--	Y	
	Connection sharing and association	--	Y	--	--	--	Y	--	
	Statement pooling	--	Y	--	--	--	--	Y	
	Light transactions	--	Y	--	--	--	--	Y	
	Caching the DataSource object	--	Y	--	--	--	--	Y	
	Optimizing sign-on for the container management of DB Connector	--	Y	--	--	--	--	Y	
	Detecting connection failure	Y	--	--	--	--	Y	Y	

Functionality		Purpose of system					Compliance with Java EE Standard		Reference location
		Reliability	Performance	Operation and maintenance	Scalability	Others	Standard	Extended	
	Waiting for acquiring a connection when connections exhaust	Y	--	--	--	--	--	Y	
	Retrying for acquiring a connection	Y	--	--	--	--	--	Y	
	Displaying information of a connection pool	Y	--	--	--	--	--	Y	
	Clearing connection pool	Y	--	--	--	--	--	Y	
	Cancelling the transaction timeout and statements	Y	--	--	--	--	Y	--	
	Output of SQL for the failure investigation	--	--	Y	--	--	--	Y	
	Automatic closing of object	Y	--	--	--	--	Y	--	
	Testing the connection with resources	--	--	Y	--	--	--	Y	
GC control functionality		--	Y	--	--	--	--	Y	*2.9*
Container extension library		--	--	--	Y	--	--	Y	*2.10*
JavaVM functionality		--	--	Y	--	--	--	Y	*2.11*

Legend:

Y: Supported

--: Not supported

Note

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

#

In the case of batch applications, you can assign optional names only to J2EE resources. The description of Enterprise Beans is not applicable.

1.2.2 Functionality for scheduling Enterprise Beans using CTM

The following table describes the functionality for scheduling Enterprise Beans using CMT. Select the functionality according to the purpose of the system. For details on the functionality, see *Reference location* in the table.

Table 1–3: Correspondence between the Enterprise Bean scheduling functionality using CTM and the purpose of systems

Functionality	Purpose of system					Compliance with Java EE Standard		Reference location
	Reliability	Performance	Operation and Maintenance	Scalability	Others	Standard	Extended	
Flow-volume control of requests	Y	Y	--	--	--	--	Y	*3.4*
Controlling priority of requests	Y	Y	--	--	--	--	Y	*3.5*
Dynamically changing the number of concurrent executions of requests	Y	Y	Y	--	--	--	Y	*3.6*
Locking and controlling requests	Y	--	Y	--	--	--	Y	*3.7*
Load balancing of requests	Y	Y	--	Y	--	--	Y	*3.8*
Monitoring and retaining request queues	Y	--	Y	--	--	--	Y	*3.9*
Connection with the TPBroker/OTM client by using the gateway functionality in CTM	--	--	--	Y	--	--	Y	*3.10*

Legend:

Y: Supported

--: Not supported

#

The functionality, for which *Y* is specified only in the *Extended* column of the *Compliance with Java EE Standard* column, indicates the functionality unique to Application Server.

1.2.3 Other extended functionality

The following table describes the other extended functionality. Select the functionality according to the purpose of the system. For details on the functionality, see *Reference location* in the table.

Table 1–4: Correspondence between other extended functionality and the purpose of systems

Functionality	Purpose of the system					Compliance with Java EE Standard		Reference location
	Reliability	Performance	Operation and Maintenance	Scalability	Others	Performance	Operation and Maintenance	
Scheduling of batch applications	Y	Y	--	--	--	--	Y	*Chapter 4*
Session failover functionality	Y	--	Y	--	--	--	Y	*Chapter 5*, *Chapter 6*
Suppression of Full GC by using the Explicit Memory Management functionality	Y	--	--	--	--	--	Y	*Chapter 7*
Output of application user logs	--	--	Y	--	--	--	Y	*Chapter 8*

Legend:

Y: Supported

--: Not supported

#

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

1.3 Description of the functionality described in this manual

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

1.3.1 Meaning of classification

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

- Description
  This part describes the functionality. This describes the purpose, characteristics, and mechanism of the functionality. Read the description, if you want to get 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 properties required for building systems. Read this when you build systems.
- 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 points to be considered when using the functionality. Make sure to read the description of the points to be considered.

1.3.2 Example of tables 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 of a table describing the classification of the functionality.

Example of a table describing the classification of the functionality

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

Category	Title	Reference location
Description	What is the *XX* functionality?	*X.1*
Implementation	Implementing an application	*X.2*
	Definitions in a DD and `cosminexus.xml`#	*X.3*
Setup	Settings in the execution environment	*X.4*
Operation	Performing operations by using the *XX* functionality	*X.5*
Notes	Points to be considered when using the *XX* functionality	*X.6*

#

For details on `cosminexus.xml`, see *13. Managing Application Properties* in the *uCosminexus Application Server Common Container Functionality Guide*.

Tip

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

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

You specify the application settings 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 the properties of a J2EE application* in the *uCosminexus Application Server Application Setup Guide*.

The tags specified in the property files correspond to either a DD or `cosminexus.xml`. For details on the mapping between a DD or `cosminexus.xml` and the tags in the property files, see *2.1 Specifications of the Cosminexus application property file (cosminexus.xml)* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

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

1.4 Main updates in the functionality of Application Server 11-00

This section describes the main updates in the functionality of Application Server 11-00 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 11-00. For details on the functionality, see *Reference location*. *Reference manual* and *Reference location* describe the main features of a particular functionality.
- *uCosminexus Application Server* is omitted from the manual names mentioned in the *Reference manual* column.

1.4.1 Simplifying implementation and setup

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

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

Item	Overview of changes	Reference manual	Reference location
Windows Server support in the development environment	uCosminexus Developer now supports the Windows Server OS so that an application development environment can be built in a cloud environment.	--	--

1.4.2 Supporting standard and existing functionality

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

Table 1–6: Changes made for supporting standard and existing functionality

Item	Overview of changes	Reference manual	Reference location
Servlet 3.0 and 3.1 support	Asynchronous servlets in Servlet 3.0 and the asynchronous I/O API in Servlet 3.1 are now supported.	*Web Container Functionality Guide*	*7.1*
EL 3.0 support	EL 3.0 is now supported.	*Web Container Functionality Guide*	*2.3.3*
JSF 2.2 support	JSF 2.2 is now supported.	*Web Container Functionality Guide*	*Chapter 3*
JAX-RS 2.0 support	JAX-RS 2.0 is now supported.	*Web Container Functionality Guide*	*Chapter 4*
WebSocket 1.0 support	WebSocket 1.0 is now supported.	*Web Container Functionality Guide*	*Chapter 5*
Addition of the NIO HTTP server functionality	The NIO HTTP server functionality was added as an in-process HTTP server that supports asynchronous servlets and non-blocking I/O processing such as WebSocket, instead of the conventional redirector and in-process HTTP server functionality.	*Web Container Functionality Guide*	*Chapter 6*
JPA 2.1 support	JPA 2.1 is now supported so that a JPA provider supporting JPA 2.1 can be used.	*Common Container Functionality Guide*	*Chapter 5*

Item	Overview of changes	Reference manual	Reference location
CDI 1.2 support	CDI 1.2 is now supported.	*Common Container Functionality Guide*	*Chapter 8*
BV 1.1 support	Bean Validation 1.1 is now supported.	*Common Container Functionality Guide*	*Chapter 9*
Java Batch 1.0 support	Batch Applications for the Java Platform (Java Batch) 1.0 is now supported.	*Common Container Functionality Guide*	*Chapter 10*
JSON-P 1.0 support	Java API for JSON Processing (JSON-P) 1.0 is now supported.	*Common Container Functionality Guide*	*Chapter 11*
Concurrency Utilities 1.0 support	Concurrency Utilities for Java EE 1.0 is now supported.	*Common Container Functionality Guide*	*Chapter 12*
WebSocket communication support	A function that relays WebSocket communication from an HTTP server to a J2EE server was added.	*HTTP Server User Guide*	*4.15*

1.4.3 Maintaining and improving reliability

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

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

Item	Overview of changes	Reference manual	Reference location
Change of the encrypted-communication module	The `mod_ssl` module was adopted as an encrypted-communication module for an HTTP server.	*HTTP Server User Guide*	*Chapter 5, Appendix H*

1.4.4 Other purposes

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

Table 1–8: Changes made for other purposes

Item	Overview of changes	Reference manual	Reference location
Addition of V9 compatibility mode	V9 compatibility mode was added to maintain the compatibility with the version 9 of Application Server for users of Application Server in which the J2EE server has been upgraded from version 9 or earlier.	*Maintenance and Migration Guide*	*10.3.3*

2 Executing Applications by Using Batch Servers

A *batch server* is a server used for executing batch applications. This chapter describes the functionality provided with a batch server and details on how to create batch applications.

For details on executing the batch applications that use the scheduling functionality of batch applications, see *4. Scheduling of Batch Applications*.

2.1 Organization of this chapter

An application used for executing a batch job, which is developed in Java, is called a *batch application*. You execute batch applications on batch servers that are the JavaVM processes of the resident type.

For an overview of executing the applications by using batch servers, see *2.2 Overview of the execution environment of batch applications*. For an overview of the resource connections of batch applications, see *2.6 Overview of resource connections and transaction management*.

The following table describes the batch server functionality that is provided by Application Server and the *reference location* for each functionality.

Table 2–1: Batch server functionality provided by Application Server

Functionality	Reference location
Batch application execution functionality	*2.3*
EJB access functionality	*2.4*
Naming management functionality	*2.5*
Resource connection functionality	*2.7*
Transaction management	*2.8*
GC control functionality	*2.9*
Container extension library	*2.10*
JavaVM functionality	*2.11*
Migrating from Java applications	*2.12*
Integrating with JP1/AJS	*2.13*

In addition to the functionality mentioned in *Table 2-1*, the scheduling functionality of batch applications are provided with batch servers. Hereafter, this functionality is called the *scheduling functionality*. For details on the scheduling functionality, see *4. Scheduling of Batch Applications*.

2.2 Overview of the execution environment of batch applications

This section gives an overview of the execution environment of batch applications.

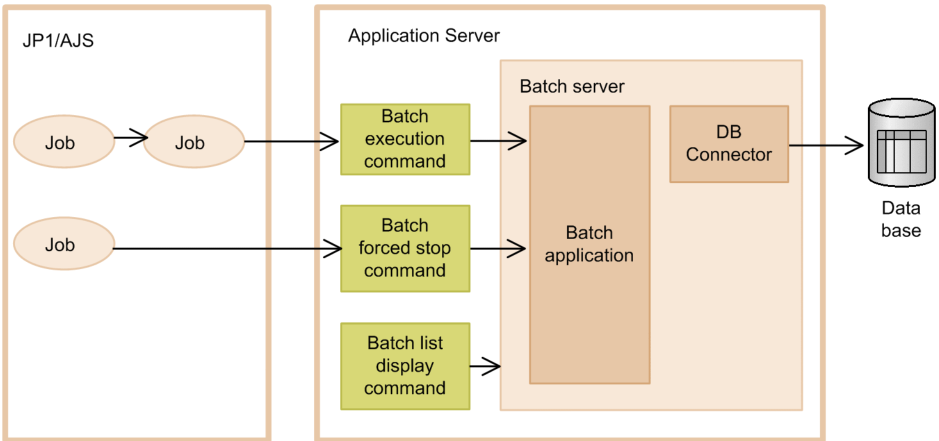
A *batch application* is a Java application with the implemented batch processing. The *execution environment* of batch applications is an environment used for executing batch applications. The execution environment is configured from a batch server that is a JavaVM process of the resident type. With Application Server, you execute the batch applications on the batch server by using commands. You can concurrently execute only one batch application on a batch server.

With batch servers, batch services are provided as the functionality to execute batch applications. If you execute a batch execution command (`cjexecjob` command), the batch service starts the execution of batch applications based on information of the batch applications. If you execute a batch forced stop command (`cjkilljob` command), the batch service forcefully stops batch applications being executed. If you execute a batch list display command (`cjlistjob`), the batch service outputs information of batch applications.

You can integrate the execution environment of batch applications with JP1/AJS. You can execute a batch application from JP1/AJS by defining the batch execution command as a JP1/AJS job in advance. You can also define the batch forced stop command as a JP1/AJS job.

The following figure shows the flow of executing batch applications.

Figure 2–1: Flow of executing batch applications



2.2.1 Systems executing batch applications

The systems executing batch applications require batch servers. You can also integrate the systems that execute batch applications, with the following products:

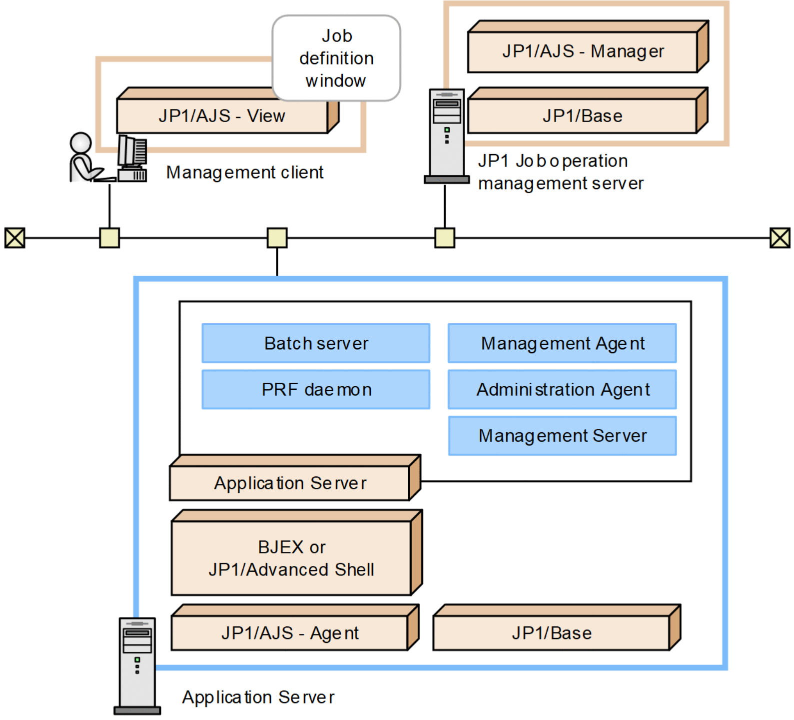
- `JP1/AJS`
- `BJEX` or `JP1/Advanced Shell`

If you integrate the systems with these products, you can define the start and stop of batch servers and the start of batch applications as jobs, for automatic execution of the batch applications. Also, if you integrate the system with BJEX or JP1/Advanced Shell you can use the functionality with the conditional execution of job steps that uses return values of batch application execution commands and automatically stop the batch application when you forcefully stop the job.

This subsection describes the structure of a system that executes batch applications. For details on the system that uses the scheduling functionality, see *4. Scheduling of Batch Applications*.

The following figure shows the configuration example of a system that executes batch applications.

Figure 2–2: Configuration example of a system that executes batch applications



In this figure, the system that executes batch applications is integrated with the following products:

- `JP1/AJS`
- `BJEX` or `JP1/Advanced Shell`

If the system is not integrated with these products, you do not require Administration Client, JP1 job Management Server, BJEX, JP1/Advanced Shell, JP1/AJS - Agent, and the JP1/Base of the batch server shown in the figure.

2.2.2 Procedure for operating batch servers and batch applications

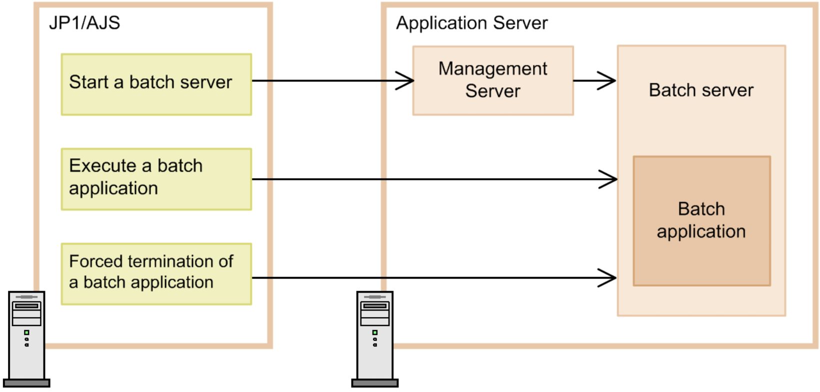
This subsection describes the procedure for operating batch servers and batch applications for each system structure.

(1) Systems integrated with JP1/AJS

With a system that is integrated with JP1/AJS, you can start a batch server, or execute and forcefully stop a batch application with JP1/AJS. With JP1/AJS, you define the operations of batch servers and batch applications as jobs, in advance.

The following figure shows the flow of operations of batch servers and batch applications.

Figure 2–3: Flow of operations of batch servers and batch applications (Integrated with JP1/AJS)



The batch server is started from JP1/AJS via Management Server of Application Server. On the other hand, you can directly execute and forcefully stop batch applications from JP1/AJS. With JP1/AJS, you define these operations as the UNIX jobs or PC jobs in advance.

For details on the job definition in JP1/AJS, see *2.13.1 Settings for integrating with JP1/AJS*.

Reference note

You can also configure without deploying Management Server. However, if you configure without deploying Management Server, and if an attempt to forcefully stop a batch application fails, you must manually restart the batch server. If you monitor the batch server by using Management Server, the batch server automatically restarts when trouble occurs. Hence, we recommend the operations using Management Server.

(2) Systems integrated with JP1/AJS, BJEX and JP1/Advanced Shell

With a system that is integrated with the following products, you can start a batch server, or execute and forcefully stop a batch application from JP1/AJS, BJEX or JP1/Advanced Shell.

- `JP1/AJS`
- `BJEX` or `JP1/Advanced Shell`

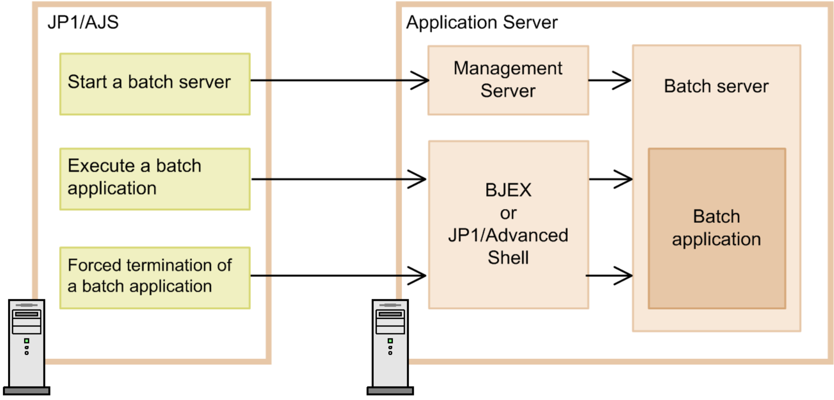
With JP1/AJS, BJEX and JP1/Advanced Shell, you define the operations of batch servers or batch applications as jobs in advance.

Important note

When you want to integrate a system with BJEX, you must also integrate the system. When you use JP1/AJS and want to integrate a system with JP1/Advanced Shell, you do not need to integrate the system with JP1/AJS.

The following figure shows the flow of operations of batch servers and batch applications.

Figure 2–4: Flow of operations of batch servers and batch applications (integrating with JP1/AJS, BJEX, and JP1/Advanced Shell)



The batch server is started from JP1/AJS via Management Server of Application Server. Execute batch applications and forced stop of batch applications from JP1/AJS via BJEX or JP1/Advanced Shell. Therefore, in JP1/AJS, BJEX, and JP1/Advanced Shell, you define the following operations as jobs in advance:

- Starting a batch server
  You define as a UNIX job or a PC job of JP1/AJS.
- Executing a batch application
  Specify a job definition XML file of BJEX or job definition script file of JP1/Advanced Shell as a UNIX job or a PC job of JP1/AJS. Define execution of batch applications in the job definition XML file of BJEX or job definition script file of JP1/Advanced Shell.
- Forced stop of the batch application
  If you forcefully stop the running UNIX job or PC job from JP1/AJS, BJEX or JP1/Advanced Shell that receives the instruction, this automatically stops the batch application.

For details on the job definition of JP1/AJS, BJEX, and JP1/Advanced Shell, see *2.13.2 Settings for integrating with JP1/AJS, BJEX, and JP1/Advanced Shell*.

Reference note

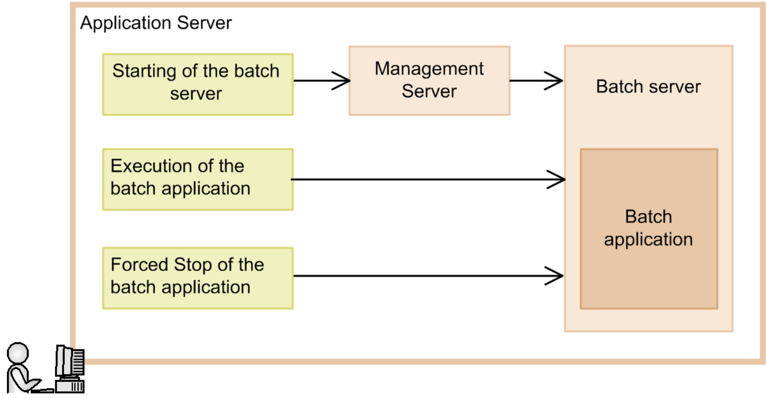
- You can also configure without deploying Management Server. However, if you configure without deploying Management Server, and if an attempt to forcefully stop a batch application fails, you must manually restart the batch server. If you monitor the batch server by using Management Server, the batch server restarts automatically when trouble occurs. Hence, we recommend the operations using Management Server.
- You can use the job log output functionality of BJEX with the batch server. However, the CPU operating time and memory usage of the `cjexecjob` command are output to the log that is output by using the job log output functionality. You cannot output the CPU operating time and memory usage of a job step of Java batch application itself.

(3) Systems not integrated with JP1/AJS, BJEX, and JP1/Advanced Shell

With systems not integrated with JP1/AJS, BJEX, and JP1/Advanced Shell, you directly use commands to start batch servers or forcefully stop batch applications.

The following figure shows the flow of operations of batch servers and batch applications.

Figure 2–5: Flow of operations of batch servers and batch applications



A batch server is started via Management Server of Application Server by using the commands provided by the Smart Composer functionality. On the other hand, you directly use commands (the *batch execution* and *batch forced stop* commands) provided by the batch application execution functionality to forcefully stop the batch applications.

Reference note

You can also configure systems without deploying Management Server. However, if you configure the systems without setting up Management Server and if and attempt to forcefully stop a batch application fails; you must manually restart the batch server. If a batch server is monitoring using Management Server, when trouble occurs, the batch server automatically. Hence, we recommend the operations using Management Server.

2.2.3 Setup and operation of the batch application execution environment

This subsection describes how to set up and operate the batch application execution environment. This subsection also describes the programs that can be integrated with the batch application execution environment.

(1) Setting up the batch application execution environment

You use the Smart Composer functionality and server management commands to set up the batch application execution environment. The procedure for setting up the batch application execution environment is as follows:

1. Set up systems by using the Smart Composer functionality.
   You define the system configuration in the Easy Setup definition file and use the commands provided with the Smart Composer functionality to execute the batch setup of systems.
2. Set up resource adapters by using the server management commands.
   You implement this process only when connecting to a database from a batch application.

For details on the Smart Composer functionality and the server management commands, see *4.6 Setting up a system for executing batch applications* in the *uCosminexus Application Server System Setup and Operation Guide*.

Important note

If you want to set up multiple batch servers, you must perform changes in such a way so that the port number of TCP/IP, used with the server, is not duplicated. For the batch server also, you use the port number of TCP/IP that is being used with a J2EE server. If you want to concurrently start multiple batch servers, and concurrently start the batch server and the J2EE server, set up in such a way so that no duplicate port numbers are used. For details on the port numbers, see *3.15 TCP/UDP port numbers used by Application Server processes* in the *uCosminexus Application Server System Design Guide*.

Reference note

You can also set up the batch application execution environment by using the management portal. For details on setting up the batch application execution environment by using the management portal, see *5. Setting up and deleting a system that executes batch applications* in the *uCosminexus Application Server Management Portal User Guide*.

(2) Operating batch application execution environment

The procedure for operating the batch application execution environment is as follows:

1. Starting a system
   You use the commands provided with the Smart Composer functionality and start the entire system including the batch servers. You also start DB Connector, when connecting to resources from batch applications.
2. Executing a batch application
   Start a batch application by using the `cjexecjob` command.
3. Stopping a batch server
   You use the commands provided with the Smart Composer functionality and stop the entire system including the batch server.

   Reference note

   For details on starting and stopping the batch application execution environment by using the management portal, see *6.1. Starting and stopping a system* in the *uCosminexus Application Server Management Portal User Guide*.

If you integrate a system with JP1/AJS, batch servers and batch applications can start from JP1/AJS. If you integrate the system with JP1/AJS, BJEX, and JP1/Advanced Shell, the batch servers can start from JP1/AJS and the batch applications can start from BJEX or JP1/Advanced Shell.

For details on starting and stopping the systems, see *2.6 Settings for starting and stopping the system* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*. For details on how to start the batch applications, see *2.3.2 Executing batch applications*.

You can use the following operation functionality with the systems executing batch applications:

(a) Functionality that support daily operations of systems

In addition to start and stop of systems, you can monitor the operation status and resource usage status of batch servers. This subsection gives an overview of the functionality used for supporting the daily operations of the systems.

- **Monitoring statistics (statistics collection functionality)**
  This functionality regularly monitors the operation status of batch servers and acquires statistics for server performance and resource information.
- **Output statistics by using management commands**

This functionality monitors the logical servers in management domain and acquires statistics by using management commands.

- **Monitoring resource exhaustion**

  This functionality monitors the resources such as memory and threads with batch servers as the target. The information, related to the resources targeted for monitoring, is output to a file at regular intervals. An alert is generated if the status of resources, targeted for monitoring, exceeds the specified threshold. If an alert is generated, a message is output and the event is reported to Management Server.
- **Reporting Management event and automatically executing a processing from Management action**

  Management events can be issued by considering all the messages output when a batch application is running, as triggers. By defining the operations to be performed, when Management events are reported, on Management Server machine, actions are now automatically executed when Management events occur.
- **Collecting statistics of CTM**

  When using the functionality of scheduling batch applications, you can collect statistics output from CTM. You can analyze processing performance of CTM on the basis of this information.

For details on the functionality that support daily operations of the systems, see *1.2.1 Functionality to support daily system operations* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

(b) Functionality that support maintenance of systems

You can output the information of processes, started by Administration Agent, such as a batch server as console logs. This subsection gives an overview of console log output.

- **Console log output**

  You can output console output information such as standard output and standard error output of processes, started by Administration Agent, to the console log. For details on the console log output, see *11. Output of the Console Log* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

You can output the logs of batch application as user logs. The user log output is one of the extended functionality. This subsection gives an overview of user log output.

- **User log output**

  If an exception occurs in a batch application, you can output message and log in the Hitachi Trace Common Library format. For details on user logs, see *8. User Log Output for Applications*.

(c) Functionality that support monitoring of a system

With this functionality you can output the history and the operations executed by the system architects and operators for programs of Application Server. You can also record the accounts used when a batch application accessed a database. This subsection given an overview of the functionality that supports the monitoring of systems.

- **Output of audit logs**

  You can output the operations performed by system architects and operators for the programs of Application Server and the history of program actions associated with the operations. You can use this information for monitoring the systems.
- **Integrating with the database audit trail functionality**

  By integrating a system with the database audit trail functionality provided by a database, you can record the accounts that are used when a batch application access the database.

For details on the functionality that supports the monitoring of systems, see the following chapters:

- *6. Audit Log Output Functionality* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*
- *7. Database Audit Trail Linkage Functionality* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*

(d) Functionality for maintaining a system

You can acquire troubleshooting data when a batch server detects an error. This subsection gives an overview of the functionality used for maintaining a system.

- **Troubleshooting**

  If you use a command during error detection, you can acquire troubleshooting data when Management Server detects the fault at the logical server. You can also output and collect the `snapshot` log of component software on Application Server.

  For example, if trouble occurs in a system, the `snapshot` log is automatically collected as troubleshooting information.

- **Performance analysis of a system by using performance analysis trace**

  The performance analysis trace is the functionality that collects performance analysis information output by the functionality of Application Server. You can analyze system performance and bottlenecks on the basis of this information.

For details on the functionality used for maintaining the systems, see the *uCosminexus Application Server Maintenance and Migration Guide*.

(3) Integrating with other programs

You can integrate the following programs with the systems executing batch applications:

- Integrating with JP1
- Integrating with cluster software

For details on the integration with JP1, see *12. Operating a JP1 Integrated System* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*. For details on the integration with cluster software, see *15. Linking with Cluster Software* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

(a) Overview of the management functionality by integrating with JP1

This subsection gives an overview of the management functionality by integrating with JP1.

- **Centralized monitoring of systems (integrating with JP1/IM)**

  By performing centralized monitoring of resource status of entire business system, you can understand and examine operation performance, detect trouble occurrences, determine causes, and take countermeasures. You can implement this functionality by integrating with JP1/IM.

- **Automatically operating a system by using jobs (integrating with JP1/AJS)**

  By defining and automating a schedule of starting and stopping servers and applications in advance, you can achieve efficient resource allocation, operation efficiency, and power saving. By integrating with JP1/AJS, you can implement automatic operations of systems by using custom jobs.

- **Collecting and consolidating audit log (integrating with JP1/Audit Management - Manager)**

  You can automatically collect the audit logs used for monitoring systems, and manage the logs in a batch. You can implement this functionality by integrating with JP1/Audit Management - Manager.

(b) Overview of node switching functionality by integrating with cluster software

This subsection gives an overview of the node switching functionality by integrating with cluster software. The cluster software that can be integrated is; Windows Server Failover Cluster# (in Windows) and HA monitor (in AIX and Linux).

#

You can use Windows Server 2016 Standard/Datacenter and Windows Server 2019 Standard/Datacenter as an OS.

- **BASIC authentication**

  This is a system configuration where executing node and standby node are set up in one-to-one ratio. In the case of the batch application execution environment, the BASIC authentication operations are supported on Application Server. When an error is detected on the executing node server or maintaining a system, this functionality is used for continuing the business processing by automatically switching to a server that is already kept in the standby state. As a result, you can decrease system downtime and impact on business processing of client.

  In the case of the batch application execution environment, you cannot use BASIC authentication of Management Server, because Management Server is not deployed.

- **Mutual node switching system**

  With the BASIC authentication configuration, two servers operate as active nodes and serve as spare nodes for each other. The operations of the mutual node switching systems on Application Server are supported.

- **Node switching system that targets host unit management model**

  This is a system configuration where N executing nodes and one spare node of the host unit management model are placed. The operations of node switching systems on Application Server of the host unit management model are supported.

2.2.4 Using multibyte characters

When using multibyte characters with the following items, use the same encoding of multibyte characters for all items:

- When using multibyte characters in the option definition file (`usrconf.cfg`) for batch applications
- When using multibyte characters in the option definition file (`usrconf.cfg`) for batch servers
- When specifying multibyte characters in the arguments of the `cjexecjob` command
- When output multibyte characters to `java.lang.System.out` or `java.lang.System.err` in the source code of batch applications

Enable display of corresponding character encoding in the environment variable `LANG` of console, on which the batch server and the `cjexecjob` command are executed.

2.3 Batch application execution functionality

The *batch application execution functionality* is one of the functionality provided with a batch server. The batch application execution functionality executes batch applications and outputs the data, output by batch applications, to the log output functionality.

This section describes the batch application execution functionality.

The following table describes the organization of this section.

Table 2–2: Organization of this section (Batch application execution functionality)

Category	Title	Reference location
Description	Overview of the batch application execution functionality	*2.3.1*
	Executing batch applications	*2.3.2*
	Forcefully stopping batch applications	*2.3.3*
	Displaying list of batch application information	*2.3.4*
	Log output of batch applications	*2.3.5*
	Executing the commands used with batch applications	*2.3.6*
Implementation	Implementing batch applications (batch application creation rules)	*2.3.7*
	Implementing batch applications (when connecting to resources)	*2.3.8*
	Implementing batch applications (when accessing EJB)	*2.3.9*
Setup	Settings of the execution environment (batch server settings)	*2.3.10*
Notes	Points to be considered when creating batch applications	*2.3.11*

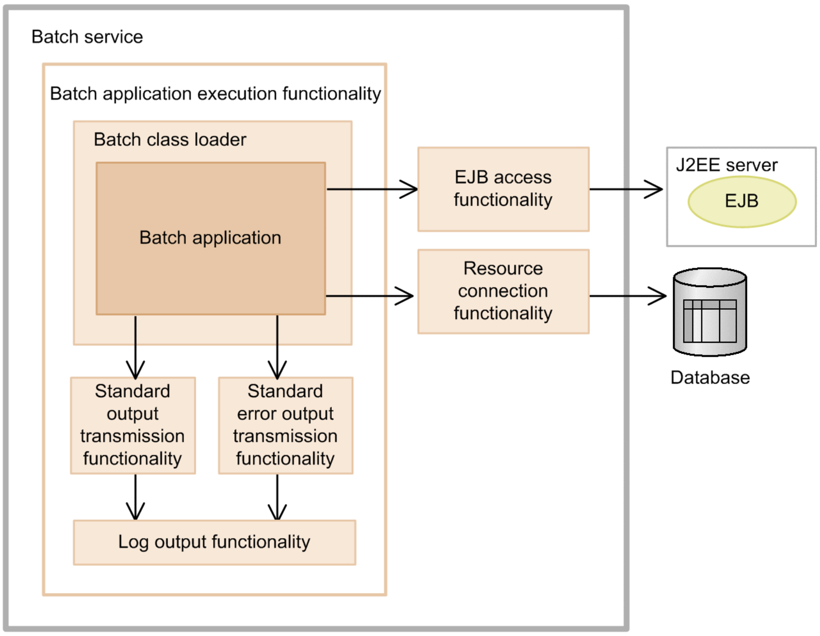
There is no specific explanation of *Operation* for this functionality.

2.3.1 Overview of the batch application execution functionality

The *batch application execution functionality* is used for executing batch applications. A batch application is executed on batch loader provided with the batch application execution functionality. The contents output by the batch application that is being executed are output to the log output functionality.

The following figure shows the batch application execution functionality.

Figure 2–6: Overview of the batch application execution functionality



You can integrate the batch application execution functionality with the EJB access functionality and the resource connection functionality.

- If you integrate the batch application execution functionality with the EJB access functionality, you can access EJB of other J2EE servers from a batch application.
- If you integrate this functionality with the resource connection functionality, you can connect to a database from a batch application.

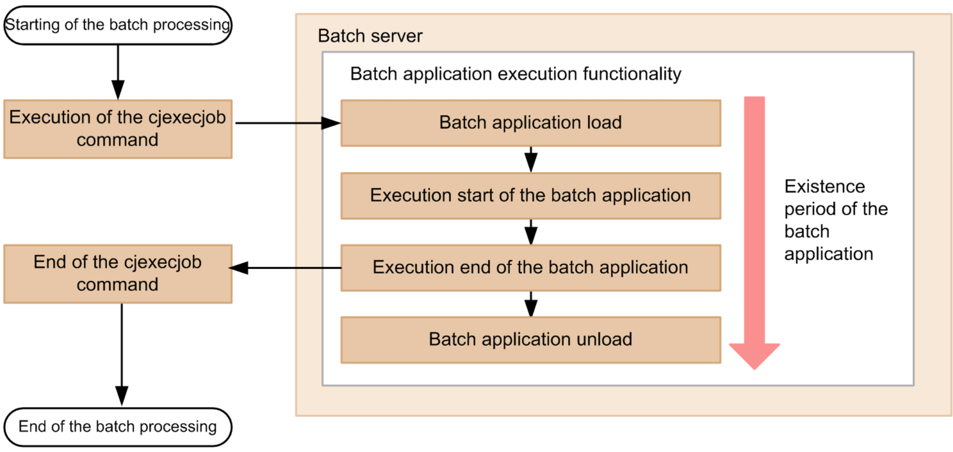
For details on the EJB access functionality, see *2.4.1 Functionality that you can use with EJB access* and for details on the resource connection functionality, see *2.7 Resource connection functionality*.

The following subsections describe life cycle of a batch application and a class loader that executes the batch application:

(1) Life cycle of a batch application

You use the `cjexecjob` command to start a batch application. The following figure shows the life cycle of a batch application.

Figure 2–7: Life cycle of a batch application



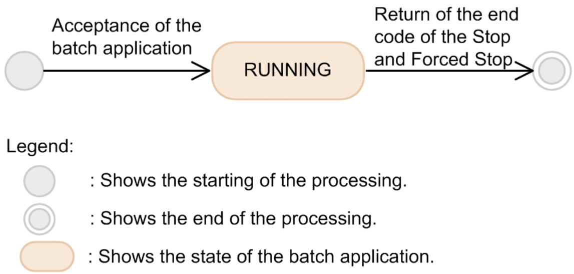
1. If you execute the `cjexecjob` command, batch class loader loads a batch application.
2. The batch application is executed on a batch server.
3. The batch application processing ends.
   After the batch application processing ends, GC occurs for the batch class loader that loaded the batch application.
4. Classes of the batch application are unloaded.

> **Important note**
>
> A batch application is loaded to a batch class whenever you execute the `cjexecjob` command and the class is unloaded when processing is complete. We do not recommend that you operate a resident batch application on a batch server.

(2) State transition of a batch application

The following figure shows the state transition of a batch application.

Figure 2–8: State transition of a batch application (When the scheduling functionality is not used)



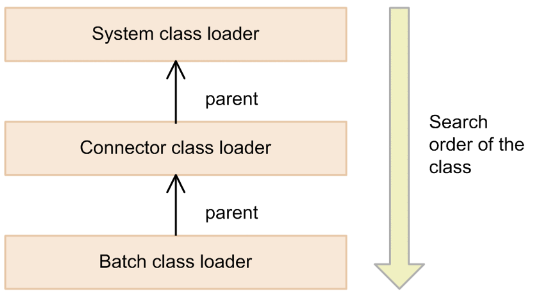
`RUNNING` is a state that shows that the batch application is executing on a batch server.

You can confirm the state of a batch application from the batch application information. For details on how to display the batch application information, see *2.3.4 Displaying list of batch application information*.

(3) Class loader executing a batch application

When executing a batch application, a class loader for the batch application is generated on the batch server. The batch application is executed on the class loader. The following figure shows the configuration of a class loader for batch applications.

Figure 2–9: Configuration of a class loader executing batch applications



The following subsections describe the above figure:

- System class loader

A system class loader loads classes provided by the component software of Application Server and classes of the container extension library.

- Timing of generation: When the J2EE server starts up
- Timing of destruction: When the J2EE server stops

- Connector class loader

  A connector class loader loads the classes included in resource adapters. Only one connector class loader exists on a batch server.

  - Timing of generation: When the J2EE server starts up
  - Timing of destruction: When the J2EE server stops

- Batch class loader

  A batch class loader loads the classes included in a batch application. A context class loader of the thread, used for executing a batch application, is a batch class loader.

  - Timing of generation: When a batch application is executed
  - Timing of destruction: When the batch application ends

When a batch class loader is generated, a message stating that the batch class loader is generated is output (KDJE55013-I message). A message stating that end processing of the batch class loader is executed is also output (KDJE55014-I message).

For details on precautions to be taken for destroying class loader, see *Appendix B.1 Configuring default class loader* in the *uCosminexus Application Server Common Container Functionality Guide*. Properly read the class loader destruction timing and the message that is output when destroying the class loader.

2.3.2 Executing batch applications

You use the `cjexecjob` command to start batch applications. When execution of the main method of a batch application ends, the batch server performs Full GC. This subsection describes how to start batch applications, and processing for starting and ending the batch applications.

If you want to stop a running batch application, you forcefully stop the batch application. For details on how to forcefully stop the batch applications, see *2.3.3 Forcefully stopping a batch application*.

(1) How to start batch applications

This subsection describes how to start batch applications.

You use the `cjexecjob` command to start a batch application. You use the following three methods execute the `cjexecjob` command:

1. Method for directly executing the `cjexecjob` command

   You start a batch application by using this method, if you do not want to use JP1/AJS, BJEX, and JP1/Advanced Shell.
2. Method for defining the `cjexecjob` command as a JP1/AJS job and executing from JP1/AJS

   You start a batch application by using this method, if you use only JP1/AJS.
3. Method for defining the `cjexecjob` command as a BJEX job step and executing a BJEX job from JP1/AJS

   You start a batch application by using this method, if you use JP1/AJS and BJEX.

4. Method for using and executing the `adshjava` command provided by JP1/Advanced Shell from the job definition script file of JP1/Advanced Shell
   Start a batch application by using this method if you want to use JP1/Advanced Shell. With this method, you can directly execute JP1/Advanced Shell or you can also execute JP1/Advanced Shell via JP1/AJS.

For details on the definitions of JP1/AJS, BJEX and JP1/Advanced Shell jobs that are used when starting a batch application with method 2, 3, and 4, see *2.13 Integrating with JP1/AJS*.

You start a batch server in advance for executing a batch application from JP1/AJS, BJEX, and JP1/Advanced Shell.

(2) Processing of starting a batch application

If you specify a class name and class path of a batch application in the `cjexecjob` command, the batch application specified in the `cjexecjob` command starts. The following processing is executed when you start a batch application:

1. Output a message (KDJE55000-I) stating that processing of a batch application starts.
2. Output a message (KDJE55001-I) stating that the `main` method of the batch application starts.
3. Execute the `public static void main(String[])` method or the `public static int main(String[])` method.

When starting a batch application, the `public static void main(String[])` method or the `public static int main(String[])` method of an execution class that is specified in the `cjexecjob` command is invoked. For method arguments, you set up the arguments specified after class name of the `cjexecjob` command.

If an attempt to start a batch application fails

If the `main` method is not defined in a batch application, an attempt to start the batch application fails. If an attempt to start the batch application fails, the batch server and the `cjexecjob` command operate as follows:

- Batch server operation
  The batch server outputs a message, and returns the information stating failure in starting the batch application with message string to the `cjexecjob` command.
- The `cjexecjob` command operation
  The command outputs the message string received from the batch server and forcefully stops the application. `1` is returned as a return value of the command.

The following table describes the conditions when an attempt to start a batch application fails, and the messages output by a batch server.

Table 2–3: Conditions where an attempt to start a batch application fails

Condition where an attempt to starting batch application fails	Message output by batch server
An attempt to read `usrconf.properties` (user property file for batch application) fails.	KDJE55035-E
The class specified in the `cjexecjob` command does not exist.	KDJE55006-E
The `public static void main(String[])` method or the `public static int main(String[])` method is not defined.	
Signatures of both the `public static void main(String[])` method and the `public static int main(String[])` method are different.	
`java.lang.NoClassDefFoundError` occurs when loading a class specified in the `cjexecjob` command.	KDJE55007-E

Condition where an attempt to starting batch application fails	Message output by batch server
A class required for invoking the `public static void main(String[])` method or the `public static int main(String[])` method is not found.	
An error occurs in the `static{}` block.	
You cannot execute the `main` method due to a problem other than those mentioned above.	KDJE55008-E

(3) Processing of ending a batch application

The batch application processing ends after execution of the `main` method is complete. The processing executed at the end a batch application is as follows:

1. Output a message (KDJE55002-I) stating that the end processing of a batch application starts.
2. Output a message (KDJE55003-I) stating that the end processing of a batch application is complete.
3. Perform Full GC.
4. Send the end code of the batch application to the `cjexecjob` command.

The following table describes the end conditions of batch applications and operations of batch servers and the `cjexecjob` command during that time.

Table 2–4: End conditions of batch applications

End condition of batch application	Batch server operation	cjexecjob command operation
The `main` method is executed until end.	Outputs the KDJE55002-I message and ends the execution of the batch application. Outputs the KDJE55003-I message after ending the application.	Ends normally. Return value: `0`
The `return` statement is executed using the `public static void main(String[])` method.		
`return` *end-code* is executed using the `public static int main(String[])` method.		Ends normally. Return value: end code specified in `return`
A class that inherits `java.lang.Throwable` or `java.lang.Throwable` is thrown outside the `main` method.	Outputs the KDJE55009-E message. Outputs exception stack trace to exception log. Ends execution of the batch application.	Outputs exception stack trace to standard error output. Abnormally ends execution of the batch application. Return value: `1`
Batch server terminated (forced termination of batch server or unexpected down of JavaVM).	None.	Outputs the KDJE55021-E message and abnormally ends execution of the batch application. Return value: `1`

An execution of a batch application does not end even if you end the `cjexecjob` command by using **Ctrl+C** or signal during the execution of the batch application. Execute `cjkilljob` if you want to forcefully stop execution of a batch application. However, if you forcefully stop using the `cjkilljob` command, the end code of the `cjexecjob` command is indefinite. For details on the batch forced stop command, see *2.3.3 Forcefully stopping a batch application*.

(4) Points to be considered when executing batch applications

If you invoke and use EJB or DB Connector from batch applications, existence of EJB and DB Connector to be used is not checked when you start the batch applications. If EJB or DB Connector referenced from the batch applications do not exist, a runtime error occurs during the execution of the batch applications. Before starting a batch application, check whether EJB at the reference location exists or not. When connecting to a database from a batch application by using DB Connector, keep the DB Connector started on the batch server.

2.3.3 Forcefully stopping a batch application

You can stop a running batch application as and when required. This is called *Forced Stop* of a batch application. This subsection describes Forced Stop of a batch application.

(1) How to forcefully stop a batch application

You use the `cjkilljob` command to forcefully stop a batch application. You use the following three methods to execute the `cjkilljob` command:

1. Method to directly execute the `cjkilljob` command

   You execute the command by using this method if you do not want to use JP1/AJS.

2. Method to define the `cjkilljob` command as a recovery job of JP1/AJS and to execute with extension of forced stop of a job or a jobnet

   You execute the command by using this method when using JP1/AJS irrespective of the usage status of BJEX or JP1/Advanced Shell.

   For details on the definition of JP1/AJS job when forcefully stopping a batch application as a recovery job of JP1/AJS with method 2, see *2.13 Integrating with JP1/AJS*.

3. Method for forcefully stopping a batch application by extending forced stop of BJEX or JP1/Advanced Shell

   If you execute a batch application by using BJEX or JP1/Advanced Shell, the batch application executed with these products is also automatically stopped when BJEX or JP1/Advanced Shell is forcefully stopped. You do not need to define recovery jobs of JP1/AJS in this method.

Note that if an attempt to forcefully stop a batch application fails, the batch server is forcefully stopped. As a result, when executing multiple applications in continuation, you must restart the batch server. We recommend that you tperform settings in advance, in such a way so that the batch server automatically restarts, when an attempt to forcefully stop the batch application fails. You can implement automatic restart of a batch sever by using operation monitoring of Management Server. For details, see *2.4. Automatically Restart When a Failure Occurs* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

(2) Processing of forced stop of a batch application

You use the `cjkilljob` command to forcefully stop a running batch application. When using the `cjkilljob` command, you execute the method cancellation for the thread that executes the batch application, and forcefully stop the batch application.

The method cancellation is a functionality that cancels the running methods. However, you may or may not cancel a method, depending on the area where you are executing the method. The area where you can cancel a method is called a *non-protected area* and the area where you cannot cancel a method is called a *protected area*. A method is canceled if the method under execution is in a non-protected area. The same method cancellation is performed during the forced stop of a batch application that is performed by the functionality of monitoring the J2EE application execution time. For details on the method cancellation processing, see *5.3.4 Method Cancellation* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

The following processes are executed when you forcefully stop a batch application.

1. Output a message (KDJE55004-I) stating that the processing to forcefully stop a batch application starts.

2. Execute the method cancellation for the `public static void main(String[])` method or the `public static int main(String[])` method.

   If an attempt to cancel a method fails, an attempt to execute forced stop fails, KDJE55017-E is output, and the batch server forcefully stops. If an attempt to execute forced stop fails, restart the batch server.
3. Output a message (KDJE55005-I) stating that the processing of forcefully stopping the batch application is complete.
4. Perform Full GC.
5. Send the end code of batch application to the `cjexecjob` command.

The following table describes the conditions where a batch application is forcefully stopped:

Table 2–5: Conditions for forced stop of a batch application

Condition for forced stop of batch application	Batch server operation	cjexecjob command operation
Batch forced stop command is executed during the execution of a batch application.	Outputs KDJE55004-I message and starts forced stop of the batch application. Outputs KDJE55005-I message when forced stop is complete. Outputs KDJE55017-E message, if an attempt to execute forced stop fails.	Normal path when performing batch forced termination Return value: `1`

If you execute a batch application by using the `adshjava` command of JP1/Advanced Shell, JP1/Advanced Shell automatically executes the `cjkilljob` command and forcefully stops the batch application when you forcefully stop a job of JP1/Advanced Shell.

(3) Points to be considered when forcefully stopping a batch application

If an attempt to forcefully stop a batch application fails, the batch server is forcefully stopped. When executing multiple batch applications in continuation, you must restart the batch server before starting a batch application that is to be executed after forcefully stopping the batch server. Hence, set up in such a way so that the batch server automatically restarts by using Management Server. For details, see *5.3.4 Method Cancellation* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

2.3.4 Displaying list of batch application information

You can display a list of information such as state of the running batch applications and start time of batch execution commands as batch application information. This subsection describes the list display of batch application information.

(1) How to display a list of batch application information

To display a list of batch application information, you directly execute the `cjlistjob` command irrespective of whether JP1/AJS is used.

You can acquire the batch application information in the unit of a batch server. In arguments of the `cjlistjob` command, you specify the batch server name for which you want to acquire the batch application information.

(2) Processing of displaying a list of batch application information

If you execute the `cjlistjob` command, you can acquire information of batch applications running on the batch server that is specified in the argument. The batch application information is output to the standard output format.

The following table describes the batch application information that you can acquire.

Table 2–6: Batch application information that you can acquire

Item of batch application information that you can acquire	Contents
State of batch application	`running` is output. `running` shows the `RUNNING` state of a batch application. For details, see *2.3.1(2) State transition of a batch application*.
Batch application name	Class name of batch application specified in the `cjexecjob` command is output.
Root application information of performance analysis trace	Communication number of root application of performance analysis trace is output. You can check the state of batch application by comparing with the root application information that is output to the performance analysis trace file.
Execution time of batch execution command	The time, at which `cjexecjob` is executed, is output in the following format. Δ shows a single byte space. `yyyy/mm/ddΔhh:mm:ss.ssssss` `yyyy`: Western calendar year, `mm`: Month, `dd`: Day, `hh`: hour, `mm`: Minute, `ss`: Second, `ssssss`: Microsecond

If there is no batch application, nothing is output even if you execute the `cjlistjob` command. In such case, the `cjlistjob` command ends normally.

The following example shows the output format and the output example of the `cjlistjob` command. Δ shows a single byte space.

Output format of cjlistjob command

```
State-of-batch-applicationΔBatch-application-nameΔRoot-application-informati
on-of-performance-analysis-traceΔExecution-time-of-batch-execution-command
```

Output example of cjlistjob command

```
running com.hitachi.mypackage.batchApp1 0x0000000000123456 2008/04/14 17:27:
35.689012
```

This example shows that a batch application that started on 2008/4/14 17:27:35.689012, on the batch server specified in argument of the `cjlistjob` command is being executed.

2.3.5 Log output of a batch application

On a batch server, execution logs of batch applications are output. In the execution logs, the output contents that are output by the running batch applications are output as the standard output or standard error output, for each batch execution command. You can use this information for investigation when a fault occurs.

A batch server outputs the data, written by batch applications in `java.lang.System.out` and `java.lang.System.err` to the respective following locations:

- Data written to `java.lang.System.out` by batch applications

  This data is output to the following locations by standard output transfer functionality of batch server:

  - User output log of batch server
  - Standard output of batch server

- Standard output of `cjexecjob` command

- Data written to `java.lang.System.err` by batch application

  This data is output to the following locations by the standard error output transfer functionality of batch server:

  - User error log of batch server
  - Standard error output of batch server
  - Standard error output of `cjexecjob` command

The messages output by the `cjexecjob`, `cjkilljob`, and `cjlistjob` commands are output to the respective following locations according to the message level:

I (Information)

Output to standard output. However, the messages (KDJE55029-I, KDJE55030-I, and KDJE55052-I) that show usage method of commands are output to the standard error output.

E (Error) and W (Warning)

Output to the standard error output.

For details on the commands used for executing the batch applications, see *3.3 Commands used with batch applications* in the *uCosminexus Application Server Command Reference Guide*. For details on the message levels, see *7.1 Format for describing the messages* in the manual *uCosminexus Application Server Messages*.

2.3.6 Executing commands used in a batch application

This subsection describes the execution of commands used in a batch application.

You can use the following three types of commands in a batch application:

- `cjexecjob` command (batch execution command)

  This command is used for executing a batch application.

- `cjkilljob` command (batch forced stop command)

  This command is used to forcefully stop a running batch application.

- `cjlistjob` command (batch list display command)

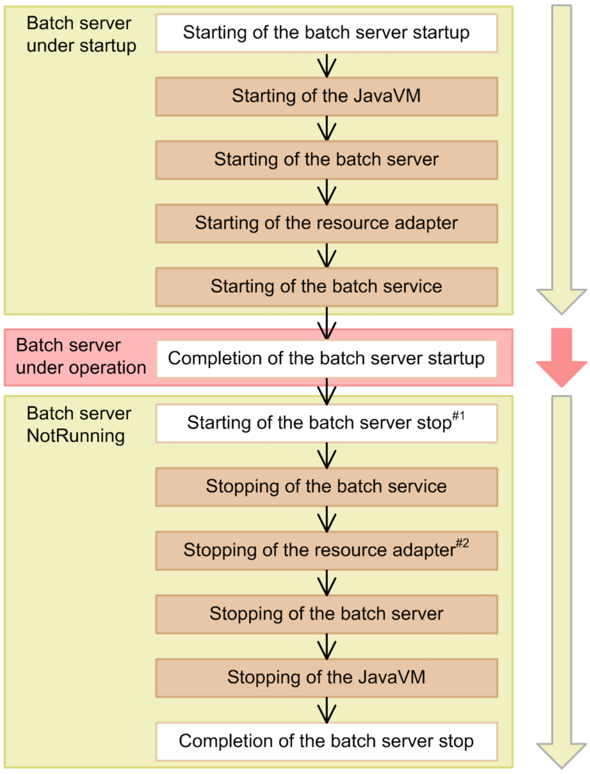
  This command is used for displaying a list of batch application information.

You might not be able to execute these commands depending on the state of the batch server. The following subsections describe batch server states and execution of commands. For details on the commands, see *3.3 Commands used with batch applications* in the *uCosminexus Application Server Command Reference Guide*.

(1) States of batch server and execution of commands

You might not be able to execute the `cjexecjob`, `cjkilljob`, and `cjlistjob` commands depending on the state of a batch server. The following figure shows the state of a batch server and availability of the commands for execution.

Figure 2–10: States of a batch server and availability of commands for execution



Legend:

: Can execute the command

: Can not execute the command (the "KDJE55012-E" message is output)

#1: In the case of an executing batch application, stand by till the batch application ends.
#2: In the case of a start-state resource adapter, stop the resource adapter.

You cannot execute the `cjexecjob`, `cjkilljob`, and `cjlistjob` commands after stopping a batch server. The KDJE55010-E message is output.

If another command is processing on the batch server, you might not be able to execute the commands, depending on the type of the command. The following table describes availability of the commands for execution, when a command is processing on a batch server.

Table 2–7: Availability of the commands for execution when a command is processing on a batch server

Command to be executed		Command under processing			
		`cjexecjob`	`cjkilljob`	`cjlistjob`	Server management command
`cjexecjob`		N	N	Y	Y
`cjkilljob`		Y	N	Y	Y
`cjlistjob`		Y	Y	Y	Y
Server management command	`cjstoprar`	N	N	Y	Δ#1
	Command other than `cjstoprar`	Y	Y	Y	Δ#1

Command to be executed	Command under processing			
	cjexecjob	cjkilljob	cjlistjob	Server management command
cjstopsv or cmx_stop_target	Y#2	Y#2	Y#2	Δ#1
cjdumpsv	Y	Y	Y	Y

Legend:

Y: Can be executed

N: Cannot be executed

Δ: Varies as per command type

#1: Operations vary as per the type of the server management command. For details, see *3.2.2 Exclusive access control of server management commands* in the *uCosminexus Application Server Application Setup Guide*.

#2: In the case of a running batch application, outputs the KDJE55033-I message and waits for the end of the batch application.

(2) If a batch server terminates abnormally during a command processing

When processing of the `cjexecjob`, `cjkilljob`, or `cjlistjob` command is executing on a batch server and if the batch server terminates abnormally, the KDJE55021-E message is output. Confirm the state of the batch server and execute the command again.

(3) Points to be considered when executing commands

Consider the following when executing commands:

- If there is no batch server when executing the `cjexecjob`, `cjkilljob`, or `cjlistjob` command, the command outputs the KDJE55010-E message and ends abnormally.
- If the `ejbserver.ctm.enabled` parameter in the Easy Setup definition file and the value specified in the `batch.ctm.enabled` key in `usrconf.cfg` (option definition file for batch application) do not match, an error might occur when executing the following commands:
    - When executing the `cjexecjob` command, the command might output the KDJE55067-E message and end abnormally.
    - When executing the `cjlistjob` command, the batch application information might not be output.

2.3.7 Implementing a batch application (Batch application creation rules)

A batch application is a Java application with the implemented contents of batch processing. This subsection describes the rules for creating batch applications.

(1) File format of a batch application

You set up a batch application in the class file format specified in JavaVM. When using multiple classes, you can also perform the following:

- Include a directory with the deployed class file, in class path.
- Include a JAR file with the archived class file is archived, in class path.

(2) Processing that can be implemented in a batch application

In a batch application, you can implement the processing that can be coded in Java. However, there are some points to be considered when using the threads that are used in file operations or batch applications. For details on the points to be considered when creating an application, see *2.3.11 Points to be considered when creating a batch application*.

(3) Starting batch processing

Define one of the following methods in the batch application, as a method to start batch processing:

- `public static void main(String[])`
- `public static int main(String[])`

You cannot execute a batch application, if the return value type of the `main` method and modifier are different. You can specify *throws* in the `main` method. The arguments specified in the `cjexecjob` command are passed to the arguments of the `main` method by a string array.

If you have enabled the use of the JavaVM end method, the batch server creates a batch application execution start thread and registers the thread in a thread group (`batchThreadGroup`). You can use the JavaVM end method if `true` is specified in the `ejbserver.batch.application.exit.enabled` parameter in the Easy Setup definition file. For details on how to set up the `ejbserver.batch.application.exit.enabled` parameter, see *2.3.10 Settings in the execution environment (batch server settings)*.

(4) Ending batch processing

Processing ends when the batch application state changes to one of the following states:

- Execution of the `main` method of the class specified in arguments of the `cjexecjob` command ends.
- An exception or error is thrown outside the `main` method.

A thread of the batch application (thread belonging to `batchThreadGroup`) ends when the state changes to one of the following states.

- If you invoke the JavaVM end method.
- If the `main` method returns.
- If an exception occurred in the `main` thread is not caught.

The following table describes end processing that you can execute when ending a batch application.

Table 2–8: End processing, that you can execute when closing a batch application

Method of closing a batch application		End processing that you can use	
		`java.io.deleteOnExit`	Shutdown hook
Ending the application by invoking JavaVM end method	Ending the application by invoking `java.lang.System.exit(int)`	Y	Y
	Ending the application by invoking `java.lang.Runtime.exit(int)`	Y	Y

Method of closing a batch application		End processing that you can use	
		java.io.deleteOnExit	Shutdown hook
	Ending the application by invoking java.lang.Runtime.halt(int)	Y	N
Ending the application with **Ctrl+C**		N	N
Ending the application due to returning of the main method		Y	Y
Ending the application due to exception in the main thread		Y	Y

Legend:
Y: Can be used
N: Cannot be used

You can use the JavaVM end method, if `true` is specified in the `ejbserver.batch.application.exit.enabled` parameter in the Easy Setup definition file. For details on the settings of the `ejbserver.batch.application.exit.enabled` parameter, see *2.3.10 Settings in the execution environment (batch server settings)*.

2.3.8 Implementing a batch application (When connecting to resources)

This subsection describes how to create a batch application to be connected to resources. This subsection also describes how to create a batch application and how to migrate from an existing batch application.

(1) When creating a batch application

If you want to create a batch application, we recommend using DB Connector for connecting to resources. DB Connector is a resource adapter provided on Application Server used for connecting to a database. The following subsection describes how to connect to resources by using DB Connector.

1. Set DB Connector on batch server.

   You assign an optional name to an object of DB Connector by using the user-specified name space functionality and register the object in JNDI name space. Make sure to use the user-specified namespace functionality, when connecting to a database from a batch application.

   You deploy DB Connector on a batch server, and then set up the optional name in the HITACHI Connector Property file. As shown in the following example, you add the `<optional name>` tag in the `<resource-external-property>` tag of the HITACHI Connector Property file and set up an optional name.

   Example of setting

```
<connector-runtime>
     :
   <resource-external-property>
       <optional-name>optional name of DB Connector</optional-name>
   </resource-external-property>
</connector-runtime>
```

   For details on how to assign optional names of DB Connector, see *2.6 Assigning optional name to Enterprise Bean or J2EE server (user-specified name space functionality)* in the *uCosminexus Application Server Common Container Functionality Guide*.

For details on how to set up DB Connector, see *3.3 Resource connections* in the *uCosminexus Application Server Common Container Functionality Guide*.

2. Perform lookup of DB Connector with the optional name specified in step 1 and acquire the connection factory (`javax.sql.DataSource` interface).

   Acquire `java.sql.Connection` from the acquired connection factory. The following example shows coding:

```
String dbName = <Optional name of DB Connector>;
     InitialContext ic = new InitialContext();
     DataSource ds = (DataSource) ic.lookup(dbName);
     Connection con =  ds.getConnection();
```

3. Connect to resources using the acquired `java.sql.Connection`.

   `java.sql.Connection` provided by JDBC driver and API are the same.

> **Important note**
>
> When using DB Connector, start DB Connector on a batch server, and then start the batch application.

(2) When migrating from an existing batch application

When migrating from an existing batch application (Java application), following two methods are used to connect to resources:

- Changing to resource connections that use DB Connector provided with Cosminexus.
- Connecting to resources by using JDBC driver (without changing connection method).

When you do not use DB Connector, you need not modify code of the batch application. Note, however, that in this case, the functions provided by DB Connector and the GC control function cannot be used. This section describes the migration method of changing the resource connection method to DB Connector and the migration method of using JDBC driver (without changing connection method).

(a) Changing to resource connections that use DB Connector

If you want to use DB Connector, change the batch application so that you can acquire `java.sql.Connection` from DB Connector. You use the following method for changing batch applications:

1. Set up DB Connector on batch server.

   You assign an optional name to an object of DB Connector by using the user-specified name space functionality and register the object in JNDI name space. Make sure to use the user-specified namespace functionality, when connecting to a database from a batch application.

   You deploy DB Connector on the batch server and then set up the optional name in the HITACHI Connector Property file. As shown in the following example, you add the `<optional-name>` tag in the `<resource-external-property>` tag of the HITACHI Connector Property file and set up an optional name.

   Example of setting

```
<connector-runtime>
     :
   <resource-external-property>
       <optional-name>optional name of DB Connector</optional-name>
   </resource-external-property>
</connector-runtime>
```

For details on how to assign optional names of DB Connector, see *2.6 Assigning optional name to Enterprise Bean or J2EE server (user-specified name space functionality)* in the *uCosminexus Application Server Common Container Functionality Guide*.

For details on how to set up DB Connector, see *3.3 Resource connections* in the *uCosminexus Application Server Common Container Functionality Guide*.

2. Change the code of resource connection processing in batch application so as to use DB Connector.

   The following example shows a batch application before processing. The underlined parts show the `Connection` acquisition processing. Change this processing to the underlined processing with *Batch application after change*. The underlined part of *Batch application after change* is the `Connection` acquisition processing of DB Connector.

   - Batch application before change

```
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection(uri,"user","pass");
con.setAutoCommit(false);
Statement stmt = con.createStatement();
stmt.executeBatch();
con.commit();
```

   - Batch application after change

```
String dbName = <Optional name of DB Connector>
InitialContext ic = new InitialContext();
DataSource ds = (DataSource)ic.lookup(dbName);
Connection con = ds.getConnection();
con.setAutoCommit(false);
Statement stmt = con.createStatement();
stmt.executeBatch();
con.commit();
```

You can use `java.sql.Connection` acquired from DB Connector in the same way as `java.sql.Connection` of JDBC driver. Hence, if you change only the acquisition method of `java.sql.Connection`, there is no need to change the code of other batch applications.

Important note

When using DB Connector, start DB Connector on a batch server, and then execute the batch application.

(b) Connecting to resources by using JDBC driver

When using a JDBC driver, you need not modify code of a batch application. However, you must add libraries of the JDBC driver to be used to the class path of batch server. For details, follow the settings of the JDBC driver to be used. The following example describes how to add libraries of the JDBC driver to the class path of a batch server. To add libraries to the class path of a batch server, you add the following code to `usrconf.cfg` (option definition file for batch server):

`add.class.path =` *full-path-of-library-of-JDBC-driver*

For details on `usrconf.cfg` (option definition file for a batch server), see *3.2.1 usrconf.cfg (Option definition file for batch servers)* in the *uCosminexus Application Server Definition Reference Guide*.

(3) Notes on batch application to be connected to resources

Note the following, when creating a batch application to be connected to resources:

(a) Points to be considered when executing a batch application

Do not stop or change the settings of DB Connector, when a batch application is running. You stop or change the settings of DB Connector after the batch application ends.

(b) Closing a connection

With a batch server, connection is not automatically closed. Therefore, implement in the application in such a way so that the used connections close without fail.

(c) Using local transactions of JTA

You can use local transactions of JTA in a batch application. You use local transactions of JTA with the following methods:

1. Acquire the `UserTransaction` object with one of the following methods:
    - Acquire by performing lookup of Naming Service.
      Lookup name: `HITACHI_EJB/SERVERS/`*Server-name*`/SERVICES/UserTransaction`
    - Acquire by using the `getUserTransaction` method of the `com.hitachi.software.ejb.ejbclient.UserTransactionFactory` class.
2. Invoke the `begin()` method of the `UserTransaction` object and start the transaction.
3. Connect to resources.
4. Invoke the `commit()` or `rollback` method of the `UserTransaction` object and conclude the transaction.

For details on how to use the `UserTransaction` interface, see *3.4.8 Processing overview and points to be considered when using UserTransaction interface* in the *uCosminexus Application Server Common Container Functionality Guide*.

The points to be considered when using `UserTransaction` are as follows:

- You can use only the `main` thread for UserTransaction. You cannot use with a user thread.
- Start and conclude the transaction with main thread.
- Transaction is not inherited when a thread is generated.
- You cannot pass connections or interface (statement) acquired from connections between threads. If you use this interface, the operation becomes invalid.

(d) Concluding a transaction

If you start a transaction in a batch application, make sure to implement conclude processing in the batch application. If you close the batch application without implementing conclude processing of the transaction, the transaction is rolled back after a timeout time exceeds.

In this case, depending on the specified value of `ejbserver.batch.application.exit.enabled` parameter in Easy Setup definition file, behavior at the time of starting a transaction (`javax.transaction.UserTransaction#begin()`) in the batch application to be executed new, varies.

If "true" is specified in ejbserver.batch.application.exit.enabled parameter

You can start the transaction in the batch application to be executed next (accepts `javax.transaction.UserTransaction#begin()`).

If "false" is specified in ejbserver.batch.application.exit.enabled parameter

You cannot start the transaction in the batch application to be executed next. In this case, `javax.transaction.NotSupportedException` occurs and `KDJE31009-E No nested transaction is supported` is output as detailed information.

Restart the batch server for recovering from the State in which you cannot start a transaction.

For details on the settings of the `ejbserver.batch.application.exit.enabled` parameter, see *2.3.10 Settings in the execution environment (batch server settings)*.

2.3.9 Implementing a batch application (when accessing EJB)

You can access EJBs of a J2EE application from a batch application. When creating a batch application that accesses EJBs, you can perform lookup of EJBs to be accessed, with the following name and then use EJBs.

- Name bound automatically (name starting with Portable Global JNDI name or `HITACHI_EJB`)
- Optional name that uses user-specified namespace functionality

When accessing EJB, you prepare the batch application with the following procedure:

1. Preparing EJB to be accessed from the batch application
   Set up a J2EE application that includes EJB to be accessed from the batch application to the start state.
2. Implementing the batch application
   In the batch application, you implement the code for using EJB.
3. Executing the batch application
   You execute the batch application created in step 2.

The following subsections describe the procedure:

(1) Preparing EJB

You prepare a J2EE application, including EJB to be accessed, from a batch application. Also you prepare a J2EE server for executing the J2EE application. For details on how to set up the J2EE servers, see *4.1 System setup for improving machine performance by allocating Web Server to a different host* in the *uCosminexus Application Server System Setup and Operation Guide*.

Start the J2EE application on the J2EE server that is set up. Use the `cjgetstubsjar` command, and acquire the RMI-IIOP stub and interface of the started J2EE application.

If you perform lookup on an optional name, when accessing EJB from a batch application, you specify an optional name of EJB in advance by using the user-specified namespace functionality. For details on setting the optional names of EJBs, see *2.6 Assigning optional name to Enterprise Bean or J2EE server (user-specified name space functionality)* in the *uCosminexus Application Server Common Container Functionality Guide*.

(2) Implementing batch application

You implement the code for acquiring EJB that is set up in *(1) Preparing EJB*, in a batch application. The following example shows coding:

```
String EjbName = EJB-lookup-name;
InitialContext ic = new InitialContext();
Object objref = ic.lookup(EjbName);
Home-interface-class-name home =
   (Home-interface-class-name) PortableRemoteObject.narrow(objref,
                                Home-interface-class-name.class;
EJB-object-class-name ejbobj = home.create();
```

Prepare home interface and EJB object file in advance. You must include home interface and EJB object in class path when compiling and executing batch applications.

(3) Executing the batch application

When executing a batch application, you specify the stub acquired in *(1) Preparing EJB* and the interface file used in *(2) Implementing batch application* with full path, in class path.

You specify URL of Naming Service used to search EJB, as a value of `java.naming.provider.url` in `usrconf.properties` (user property file for batch application).

However, when concurrently using the resource connection functionality and the EJB access functionality, use the Naming Service switching functionality and specify Naming Service which performs lookup of EJB. In this case, do not specify `java.naming.provider.url` in `usrconf.properties` (user property file for batch application). For details on the Naming Service switching functionality, see *2.10 Switching CORBA Naming Service* in the *uCosminexus Application Server Common Container Functionality Guide*.

2.3.10 Settings in the execution environment (batch server settings)

You must perform batch server settings, if you want to use batch application execution functionality. Implement the batch server settings in the Easy Setup definition file.

Important note

By default, the scheduling functionality is not used (set to `false`). When not using the scheduling functionality, do not change settings of the following parameter and key.

- `ejbserver.ctm.enabled` parameter of a logical J2EE server (j2ee-server) in the Easy Setup definition file.
- `batch.ctm.enabled` key in `usrconf.cfg` (object definition file for batch application)

You specify the definition of batch application execution functionality in the `<configuration>` tag of the logical J2EE server (j2ee-server) of the Easy Setup definition file.

The following table describes the definition of batch application execution functionality in the Easy Setup definition file.

Table 2–9: Definition of batch application execution functionality in the Easy Setup definition file

Field	Parameter to be specified	Specification contents	Required or Optional
Setting for building a server as a batch server	`batch.service.enabled`	To build a server as a batch server, make sure to specify `true`.	Required

Field	Parameter to be specified	Specification contents	Required or Optional
Setting for not using SecurityManager	`use.security`	SecurityManager is not used. Make sure to specify `false` in parameter value.	Required
Setting for enabling light transaction functionality	`ejbserver.distributedtx.XATransaction.enabled`	You cannot use global transactions. You use local transactions[#1]. Make sure to specify `false` in parameter value. Because *false* is set up by default, do not change this parameter	Required
Setting of not using the Explicit Memory Management functionality	`add.jvm.arg`	If the Explicit Memory Management functionality is not implemented in batch application, we recommend disabling the Explicit Memory Management functionality. To disable the Explicit Memory Management functionality, specify `-XX:-HitachiUseExplicitMemory` as a value of parameter. In case of default settings, the Explicit Memory Management functionality is enabled (`-XX:+HitachiUseExplicitMemory`).	Optional
Setting of real server name	`realservername`	Specify real server name of the batch server. Note that the real server name you specify must be unique within the host. If real server name is omitted, logical server name is specified.	Optional
Setting of JavaVM operation when invoking JavaVM end method	`ejbserver.batch.application.exit.enabled`	If you invoke the following JavaVM end methods in a batch application, specify whether JavaVM is to be ended. • `java.lang.System.exit(int)` • `java.lang.Runtime.exit(int)` • `java.lang.Runtime.halt(int)` Default value is `true` (end the thread of batch application without ending JavaVM). If you specify `true` or omit the setting, when you invoke JavaVM end method, thread of the batch application (thread belonging to batchThreadGroup) is ended and JavaVM is not ended. If you specify `false`, when you invoke JavaVM end method, JavaVM is ended for each batch server. As a result, you cannot use JavaVM end method and shut down hook in the batch application. [#2]	Optional

Legend:

Required: Must be specified

Optional: Specify as and when required

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

#1 In the case of a batch server, you use the light transaction functionality that provides the optimized environment in a local transaction. The light transaction functionality is enabled, if `false` is specified in the `ejbserver.distributedtx.XATransaction.enabled` parameter.

#2 If `false` is specified in the `ejbserver.batch.application.exit.enabled` parameter, you cannot use the JavaVM end method and shutdown hook. Take actions as followings:

- Actions for the JavaVM end method
  You code the batch processing contents in the `public static int main(String[])` method. When returning end code, you use `return` *end-code*. However, if you use `return`, the `finally` block is executed.
- Actions for shutdown hook
  If you want to implement processing, when closing a batch application, code the processing in the `finally` block of the `main` method.

2.3.11 Points to be considered when creating a batch application

This subsection describes the processing, for which you must take care when creating a batch application, and the functionality that you cannot use in a batch application. Confirm these contents, and then create a batch application.

(1) Processes that require attention

You must take care of the following processing, when creating a batch application:

(a) File and directory operations

Do not operate the following files and directories in a batch application:

- Files and directories below the Cosminexus installation directory
  For details on the files and directories under the Cosminexus installation directory, see *Appendix B Post-Installation Directory Configuration* in the *uCosminexus Application Server System Setup and Operation Guide*.
- Files and directories below the work directories of batch server
  For details on the work directories of a batch server, see *Appendix C.2 Work directory of the batch server* in the *uCosminexus Application Server System Setup and Operation Guide*.

When handling the files and directories in a batch application, you cannot use a relative path as a path of the files and directories. If you want to acquire a relative path from a directory by executing the `cjexecjob` command, use the value of `ejbserver.batch.currentdir`. For details on `ejbserver.batch.currentdir`, see *ejbserver.batch.currentdir property* in the *uCosminexus Application Server API Reference Guide*.

The following example shows how to modify a batch application.

Before modification

```
File f = new File("DataFile.txt");
```

After modification

```
File f = new File(System.getProperty("ejbserver.batch.currentdir") + System.
getProperty("file.separator") + "DataFile.txt");
```

(b) Using threads

A batch server does not wait for end of thread that is created or started by a batch application. When using threads in a batch application, you implement in such a way so that all the started user threads are completed before ending the batch application. User threads are out of scope of method cancellation.

If `true` is specified in the `ejbserver.batch.application.exit.enabled` parameter in the Easy Setup definition file, note the following points:

- You cannot create a thread group (`ThreadGroup`).
- If a handler inheriting `UncaughtExceptionHandler` that is an interface of `java.lang.Thread class` is registered in a batch application, the processing of the registered handler is executed, when invoking JavaVM end method. In this case, the `jp.co.Hitachi.soft.jvm.SpecialThrowable` exception might be passed in an argument of the `uncaughtException` method.

If a thread created by a batch application remains, the classes of the batch application or the used resources are not released. As a result, when you attempt to start next batch application, the batch application might fail to start. In a user thread, you cannot invoke the following batch server functionality:

- Batch application execution functionality
- EJB access functionality
- Functionality provided by naming management
- Functionality provided by resource connection and transaction management
- GC control functionality
- Functionality provided by container extension library

(c) Automatic closing of resources when ending JavaVM

On a batch server, a batch application is executed on JavaVM of the server. Therefore, if the implementation expects the processing of automatically closing resources, as a part of ending JavaVM, memory or file descriptor leakage might occur. For example, leakage occurs in the following cases:

- If a ZIP file or a JAR file is open and if you do not explicitly close the file, `C` heap area leaks.
- If you specify `ejbserver.batch.application.exit.enabled=false` in `usrconf.properties` on a batch server, the file is not deleted until the batch server stops even if you use `java.io.File.deleteOnExit().C` heap area leaks until the batch server stops.

To avoid these problems, implement batch applications in such a way so that the resources close properly.

If you do not explicitly close files and sockets, the timing of resource release is indefinite. This might impact the execution of subsequent batch applications. Make sure to explicitly close files and sockets.

In the case of a batch server, you cannot use automatic closing of connection. Make sure to close connection inside batch applications.

(d) Using JavaVM end methods

If you specify `true` in the `ejbserver.batch.application.exit.enabled` parameter in the Easy Setup definition file, you can use the following JavaVM end methods:

- `java.lang.System.exit(int)`
- `java.lang.Runtime.exit(int)`
- `java.lang.Runtime.halt(int)`

For details on the settings of the `ejbserver.batch.application.exit.enabled` parameter, see *2.3.10 Settings in the execution environment (batch server settings)*. For details on the points to be considered when using the JavaVM end methods, see *(3) Points to be considered when using JavaVM end method*.

(e) Using shutdown hook

If you specify `true` in the `ejbserver.batch.application.exit.enabled` parameter in the Easy Setup definition file, you can use shutdown hook in the following cases:

- If you invoke JavaVM end method
- If the `main` method returns
- If an exception that occurs in the `main` thread is not caught

For details on the settings of the `ejbserver.batch.application.exit.enabled` parameter, see *2.3.10 Settings in the execution environment (batch server settings)*.

(2) Functionality that you cannot implement in batch applications

You cannot use the following functionality in a batch application. Take actions by using the procedure shown in *Action*.

- **Input from standard input**
  You cannot perform input processing from standard input that uses `java.lang.System.in`.

  Action

  Use a file when an input processing is required.

- **Using JNI**
  You cannot use the execution functionality of native libraries through JNI.

  Action

  When using JNI, use through container extension library. In that case, load native libraries in container extension library.

- **Replacing set of system properties**
  You cannot use the following method:
  - `java.lang.System.setProperties(java.util.Properties)`

  Action

  Use `java.lang.System.setProperty(String, String)`.

- **Reallocating standard output stream and standard error output stream**
  You cannot use the following methods:
  - `java.lang.System.setOut(java.io.PrintStream)`
  - `java.lang.System.setErr(java.io.PrintStream)`

  Action

  Do not use `java.lang.System.out` and `java.lang.System.err`. You directly use the `PrintStream` object that you want to output.

(3) Points to be considered when using JavaVM end method

If you specify `true` in the `ejbserver.batch.application.exit.enabled` parameter in the Easy Setup definition file, JavaVM is not ended even if you use the JavaVM end method in a batch application. In such case, you can end only the thread invoked by the JavaVM end method.

This subsection describes the points to be considered when using the JavaVM end method if you have specified `true` in the `ejbserver.batch.application.exit.enabled` parameter.

(a) Differences with Java language specifications

The specifications of the JavaVM end method used in a batch application are different from the Java language specifications. The following table describes the differences with Java language specifications.

Table 2–10: Differences with Java language specifications

Field	In Java language specification	In batch application
End target	JavaVM	Thread which invoked JavaVM end method

Field	In Java language specification	In batch application
java logic coded after invocation	Processing, coded after invoking JavaVM end method, is not executed.	Processing, coded after invoking JavaVM end method, is executed in the following cases: • If JavaVM end method is coded in try block, corresponding `finally` block is executed. #1 • If `java.lang.Thread.UncaughtExceptionHandler` is registered in thread, `UncaughtExceptionHandler` is executed. #1
Multiple invocation	Cannot be used.	Multiple invocation is performed in the following cases: • If you invoke the same JavaVM end method from `finally` block as in the case of the thread that invokes the JavaVM end method • If you invoke JavaVM end method from multiple user threads#2, started from the batch application

#1: You might not be able to end the thread, if an exception occurs in the `finally` block and in the method invoked in the `finally` block, and execution of the `finally` block is interrupted in middle without catching the exception in the `finally` block.

In the following cases, the time might be required for ending a thread or you might not able to end the thread:

- If Java program processing for which the time is required is coded in the `finally` block and in the method invoked in the `finally` block
- Java program processing for which the time is required is coded in `java.lang.Thread.UncaughtExceptionHandler`

An infinite loop, monitor waiting by the `synchronized` statement, and waiting by `java.lang.wait()` are processes of a Java program that require time.

#2: User thread shows a child thread created by a batch application. Take care of the following, when using user threads:

- If `interruptedException` is caught in the `run()` method, a user thread is not ended, and remains as it is.
- If the `main` thread has ended even if the user thread remains, start of next application is accepted. However, memory leakage occurs.

Reference note

If you want to end JavaVM when executing the JavaVM end method, you specify `false` in the `ejbserver.batch.application.exit.enabled` parameter in the Easy Setup definition file. If you specify `false`, JavaVM ends for each batch server when executing the JavaVM end method.

(b) Processing when you invoke JavaVM end method

This subsection describes a processing, when you invoke JavaVM end method for each batch application.

If you invoke JavaVM end method in a batch application, implemented with a single thread

You end the `main` thread and end the execution of a batch application.

The following table describes the operations, when the JavaVM end method is invoked for multiple times.

Table 2–11: Operations when JavaVM end method is invoked for multiple times (In the case of a batch application implemented with a single thread)

No.	Field	Operation
1	End reporting to batch application execution functionality	Report end only when you invoke first JavaVM end method. Do not report when you invoke second or later JavaVM end method.
2	Returning end code	The end code, specified in argument, is enabled when you invoke first JavaVM end method. The end code, specified in argument, is disabled when you invoke second or later JavaVM end method.

No.	Field	Operation
3	Thread, which invoked JavaVM end method	The thread ends irrespective of number of invocation of JavaVM end method.

If you invoke JavaVM end method in a batch application, implemented with multithread

A thread that invokes the JavaVM end method ends. The processing of other threads varies according to the source thread, from which the JavaVM end method is invoked.

- If you invoke the JavaVM end method with the `main` thread, and if the `main` method returns or if an exception occurred in the `main` thread is not caught.

  Batch application execution functionality executes `interrupt` of `java.lang.Thread` class for all running user threads.

- If you invoke JavaVM end method with user thread

  The batch application execution functionality executes `interrupt` of the `java.lang.Thread` class for all running user threads, except the following threads:

  - User thread, which invoked JavaVM end method

  - `main` thread

  The batch application execution functionality executes method cancellation for the `main` thread that invokes a user thread. If method cancellation is successful, the `main` thread ends, and then the batch application ends. If an attempt to execute method cancellation fails, JavaVM ends for each batch server.

  We do not recommend invoking the JavaVM end method in user threads because an attempt to execute method cancellation.

In both the cases, when the `main` thread ends, start of next batch application is accepted irrespective of whether a user thread remains.

2.4 EJB access functionality

You can access EJB of other J2EE application from a batch application. This functionality is called *EJB access*. This section describes the functionality that you can use with the EJB access functionality.

For details on how to create a batch application that accesses EJBs, see *2.3.9 Implementing a batch application (when accessing EJB)*.

The following table describes the organization of this section.

Table 2–12: Organization of this section (EJB access functionality)

Category	Title	Reference location
Description	Functionality that you can use with EJB access	*2.4.1*
Setup	Settings in the execution environment (batch server settings)	*2.4.2*

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

2.4.1 Functionality that you can use with EJB access

The following table describes the functionality that you can use with EJB access. For details on the functionality, see the description in *Reference location*.

Table 2–13: Functionality that you can use with EJB access

Category		Functionality	Explanation	Reference manual#	Reference location
JNDI	Basic function ality	Binding and lookup of objects to JNDI namespace	You can lookup the reference of EJB home objects and business interface from Naming Service by using automatically bound name (Portable Global JNDI name or name starting with `HITACHI_EJB`) or user-specified namespace.	*Common Container Functionality Guide*	*2.3*
	Extende d function ality	Searching CORBA Naming Service with round-robin policy	For the systems, which are configured with multiple Naming Service and J2EE servers, you can perform lookup from a batch application with round robin. Accordingly, you can achieve load balancing.		*2.7*
		Caching by naming management functionality	You can maintain (cache) objects, looked up from Naming Service, on memory. You can decrease cost on access performance to Naming Service by using cache.		*2.8*
EJB		Executing Enterprise Bean	You can invoke Enterprise Bean, being executed in EJB container, from a batch application. However, you can use only remote invocation method. You cannot perform local invocation.	*EJB Container Functionality Guide*	*2.2*
		Invoking Enterprise Bean			*3.4*
		Acquiring RMI-IIOP stub and interface	You can invoke an application from a batch application by using RMI-IIOP functionality of Cosminexus TPBroker.		*3.7*
		Invoking remote interface of EJB	You can select send operation if communication failure occurs when invoking EJB from a batch application.		*2.13*
Transaction		Transaction management	You can start and conclude transactions in a batch application. However, you cannot use global transactions in a batch application.	*Common Container*	*3.4*

Category	Functionality	Explanation	Reference manual#	Reference location
			Functionality Guide	
	Implementing transactions in EJB client application	You can start and conclude transactions in a batch application by acquiring `UserTransaction`. The following two methods are used to acquire UserTransaction: 1. Method in which `UserTransactionFactory` class is used 2. Method in which lookup is used	*EJB Container Functionality Guide*	*3.5*
Others	Setting timeout in EJB container	You can set up a timeout of the RMI-IIOP communication in communication between a batch server and a Naming Service, and a batch server and a J2EE server. In a batch application, a timeout of Stateful Session Bean, timeout of EJB object of Entity Bean, and timeout of instance acquisition waiting are not applicable.	*EJB Container Functionality Guide*	*2.11*
	Performance analysis of system that uses performance analysis trace	You can output performance analysis trace of batch application.	*Maintenance and Migration Guide*	*Chapter 7*
	Outputting user log of an application	You can output log of a batch application.	This manual	*Chapter 8*

uCosminexus Application Server is omitted in the manual names mentioned in *Reference manual*.

2.4.2 Settings in the execution environment (Batch server settings)

You must perform batch server settings, if you want to use EJB access functionality.

You implement the batch server settings in the Easy Setup definition file. You specify the definition of the EJB access functionality 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 EJB access functionality in the Easy Setup definition file.

Table 2–14: Definition of EJB access functionality in the Easy Setup definition file

Field	Parameter to be specified	Setting contents
Timeout of RMI-IIOP communication	`ejbserver.rmi.request.timeout`	Specify a communication timeout between client and server of the RMI-IIOP communication.
Operation of EJB client when communication failure occurs in remote interface	`ejbserver.container.rebindpolicy`	Specify reconnection operation and request resending operation to be performed on the batch server, if communication failure occurs when invoking the EJB method.
Fixing communication port and IP address of batch server	`vbroker.se.iiop_tp.scm.iiop_tp.listener.port`	Specify communication port of batch server.
	`vbroker.se.iiop_tp.host`	Specify whether the IP address or host name, used by batch server, are to be fixed.

Note: For details on Easy Setup definition file and parameters, see *uCosminexus Application Server Definition Reference Guide*.

2.5 Naming management functionality

Naming management is one of the functionality provided by a J2EE service. The J2EE service is a functionality used as a component functionality of J2EE container. With the naming management, names and storage locations of objects (EJB home objects corresponding to Enterprise Bean, references of business interface, and J2EE resources) are managed. By using the naming management functionality, for a batch application, you can use the required objects from the names even if you do not know the storage location of Enterprise Beans or resources to be invoked. This section describes the naming management functionality that you can use with batch servers, and how to set up the naming management functionality.

The following table describes the organization of this section.

Table 2–15: Organization of this section (Naming management functionality)

Category	Title	Reference location
Description	Naming management functionality that you can use on a batch server	*2.5.1*
Setup	Settings in the execution environment (batch server settings)	*2.5.2*

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

2.5.1 Naming management functionality that you can use on a batch server

The following table describes the naming management functionality that you can use on a batch server. For details on the naming management functionality, see *2. Naming management* in the *uCosminexus Application Server Common Container Functionality Guide*.

Table 2–16: Naming management functionality

Functionality	Explanation
Binding and lookup of objects to JNDI namespace	Bind and manage objects as name of JNDI namespace. You can perform lookup from batch applications by using bound names. In a batch application, you cannot use lookup in `java:comp/env`.
Assigning optional name to Enterprise Bean or J2EE resources (user-specified namespace functionality)	You can assign optional name to J2EE resources. You can perform lookup from a batch application with any name set up as an optional name. When connecting to a database from a batch application, make sure to specify optional name to J2EE resources. For details on J2EE resources, see *2.6 Assigning optional name to Enterprise Bean or J2EE server (user-specified name space functionality)* in the *uCosminexus Application Server Common Container Functionality Guide*. In a batch application, description of Enterprise Bean is not applicable.
Searching the CORBA Naming Service by round-robin policy	You can perform lookup of EJB home object references having the same name (optional name) registered on multiple CORBA Naming Service, in compliance with round robin policy.
Caching by the naming management functionality	You can perform caching of already looked up EJB home object references. You can decrease the time required for processing when you perform lookup of the same object from second time onwards.
Switching CORBA Naming Service	You can switch JNDI namespace to be targeted for lookup by using instance prefix judgment of the `InitialContext` class.

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

- The objects other than the CORBA object reference are registered by converting the targeted objects to the CORBA objects and registering the CORBA object reference to the CORBA Naming Service.
- The objects other than CORBA objects are searched by searching the CORBA object reference and acquiring original objects by reverse conversion from the CORBA objects.

2.5.2 Settings in the execution environment (Batch server settings)

You must perform batch server settings, if you want to use naming management functionality.

You implement the batch server settings in the Easy Setup definition file. You specify the definition of the naming management functionality in the `<configuration>` tag of a logical J2EE server (j2ee-server) of the Easy Setup definition file.

The following table describes the definition of the naming management functionality in the Easy Setup definition file.

Table 2–17: Definition of naming management functionality in the Easy Setup definition file

Field	Parameter to be specified	Setting contents
Basic set up	`ejbserver.naming.host`	Specify host name of the CORBA Naming Service. #1
	`ejbserver.naming.port`	Specify port number of the CORBA Naming Service. #1
Round robin search#2	`ejbserver.jndi.namingservice.group.list`	Specify group of the CORBA Naming Service.
	`ejbserver.jndi.namingservice.group.`*Specify-group-name*`.providerurls`	Specify root location of the CORBA Naming Service belonging to each group.
	`java.naming.factory.initial`	Specify class in which implementation of `InitialContextFactory` is delegated.
Naming caching	`ejbserver.jndi.cache`	Specify whether the caching in naming is to be enabled.
	`ejbserver.jndi.cache.interval`	Specify cache clearing interval.

	`ejbserver.jndi.cache.interval.clear.option`	Specify range of cache clearing.
		Example of setting for regularly clearing cache (when defining physical tier) is as follows:
		(Example)
		`<configuration>`
		`<logical-server-type>j2ee-server</logical-server-type>`
		`<param>`
		`<param-name>ejbserver.jndi.cache</param-name>`
		`<param-value>on</param-value>`
		`</param>`
		`<param>`
		`<param-name>ejbserver.jndi.cache.interval</param-name>`
		`<param-value>60</param-value>`
		`</param>`

Field	Parameter to be specified	Setting contents

		`<param>`
		`<param-name>ejbserver.jndi.cache.interval.clear.option</param-name>`
		`<param-value>check</param-value>`
		`</param>`
		`:`
		`<configuration>`

Communication timeout of Naming Service	`ejbserver.jndi.request.timeout`	Specify timeout time for communicating with Naming Service.

Note: For details on the Easy Setup definition file and parameters, see *4.3 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

#1: By default, batch server automatically starts and uses the CORBA Naming Service having *localhost* host name and `900` port number by inline process.

#2: For round robin search, using the user-specified namespace functionality is a prerequisite. If you want to use user-specified namespace functionality, you must customize operation settings of server management commands. For details on how to specify the settings, see *2.6.7 Settings in the execution environment* in the *uCosminexus Application Server Common Container Functionality Guide*.

2.6 Overview of resource connections and transaction management

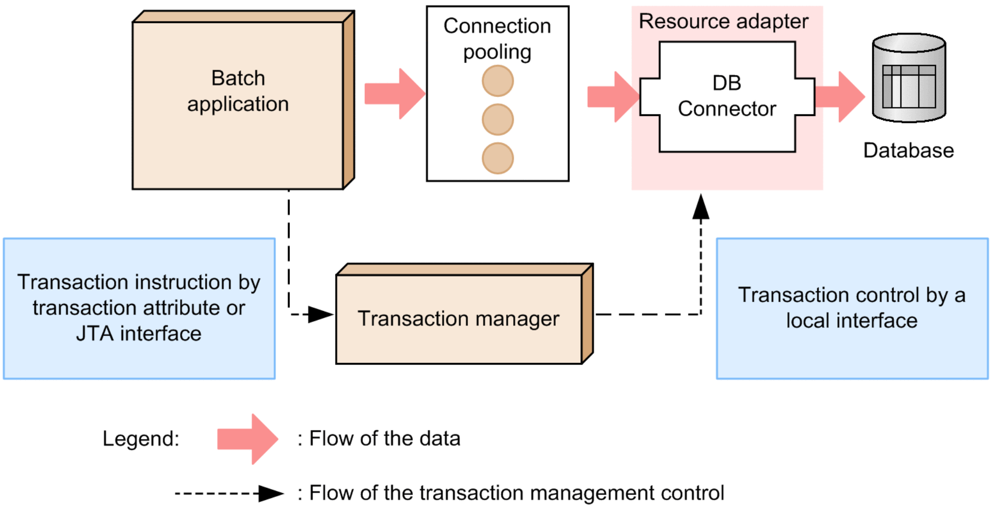
In a batch application, you can connect to a database by extending processing. To connect to a database from a batch application, you deploy and use a resource adapter, corresponding to the resources to be connected. DB Connector that is the resource adapter used for connecting to a database is provided with Application Server.

With Application Server, the connection pooling and the transaction management functionality are provided for accessing these resources efficiently and reliably. When using the connection pooling, you can perform pooling of connections for resources and use the connections efficiently. Properly remove the connections, in which failure occurs, from the connection pool. When using the transaction management functionality, transaction manager properly controls transactions of resource access, on the basis of transaction attributes specified for each method and instructions by JTA interface (`UserTransaction`).

You cannot use global transactions in a batch application.

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

Figure 2–11: Example of connecting to resources by using connection pooling and transaction management functionality



For details on how to create a batch application to be connected to resources, see *2.3.8 Implementing a batch application (When connecting to resources)*.

2.7 Resource connection functionality

In a batch application, you can use a database as a resource. This section describes how to connect to a database from a batch application.

The following table describes the organization of this section.

Table 2–18: Organization of this section (Resource connection functionality)

Category	Title	Reference location
Description	Databases that can be connected	*2.7.1*
	How to connect to resources	*2.7.2*
	Types of DB Connector (RAR file)	*2.7.3*
	How to use a resource adapter	*2.7.4*
	How to set up a resource adapter	*2.7.5*
	Procedure for setting a resource adapter	*2.7.6*
Setup	Settings in the execution environment	*2.7.7*

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

2.7.1 Databases that can be connected

You can connect to the following databases from a batch server. However, you cannot use global transactions on a batch server.

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

#:

You can connect to SQL Server only in Windows.

For using these databases, you use a resource adapter. To use a resource adapter, you must use the server management commands and perform operations such as setting properties of and importing resource adapters. For details on setting the resource adapters, see *2.7.7(2) Resource adapter settings*.

Before setting resources, understand the points to be considered when setting resources. When using the server management commands, customize operation settings of the server management command as and when required. For details on the points to be considered when setting resources and operation settings for using the server management commands, see *3.3 Customizing operation settings of server management commands* in the *uCosminexus Application Server Application Setup Guide*.

For the following details on connecting to databases, see *3.6 Connecting to databases* in the *uCosminexus Application Server Common Container Functionality Guide*.

- Overview of connecting with DB Connector
- Mapping of databases and JDBC drivers

- JDBC specifications supported by DB Connector
- Prerequisites and points to be considered when connecting to a database [#]

#:

See this description depending on the type of database to be connected.

2.7.2 How to connect to resources

To connect to a database form a batch application, directly use JDBC drivers or use resource adapters provided on Application Server. Use DB Connector if you want to use a resource adapter. The following table describes the functionality that you can use when connecting to a database from a batch application, for each connection method. If you use DB Connector, in addition to the functionality mentioned in the table below, you can also use the functionality provided by DB Connector. For details on the functionality provided by DB Connector, see *2.7.4(1) Resource adapter functionality*.

Table 2–19: Functionality that you can use when connecting to a database

Functionality that you can use			How to connect	
			DB Connector	JDBC driver
Executing SQL			Y	Y
Using transactions	Transactions by Connection API		Y	Y
	JTA	Local transactions	Y	N
		Global transactions	N	N
GC control functionality			Y	N

Legend:
Y: Can be used
N: Cannot be used

2.7.3 Types of DB Connector (RAR file)

When connecting to a database by using DB Connector, you use a RAR file appropriate to the JDBC driver to be used. You use the server management commands to operate the RAR file. For details on how to operate the RAR files by using the server management commands, see *4. Setting Resource Adapter* in the *uCosminexus Application Server Application Setup Guide*.

The following table describes the types of JDBC driver and RAR files that you can use on a batch server:

Table 2–20: Mapping of JDBC drivers and RAR files

JDBC driver	RAR file	Explanation
HiRDB Type4 JDBC Driver	`DBConnector_HiRDB_Type4_CP.rar`	This RAR file is used to connect to HiRDB and XDM/RD E2. Use this file when you do not perform transaction management or use local transactions.
Oracle JDBC Thin Driver	`DBConnector_Oracle_CP.rar`	This RAR file is used to connect to Oracle. Use this file when you do not perform transaction management or use local transactions.

JDBC driver	RAR file	Explanation
JDBC driver of SQL Server	`DBConnector_SQLServer_CP.rar`	This RAR file is used to connect to SQL Server (only in Windows). Use this file when you do not perform transaction management or use local transactions.

Note: When using a new RAR file of DB Connector, you can use a template file of the HITACHI Connector Property file provided with Application Server and define properties. The template file of the HITACHI Connector Property file is provided for RAR files of all DB Connectors. For details on the provided template files, see *4.1.13 Template files of the HITACHI Connector Property File* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

2.7.4 How to use a resource adapter

When connecting to resources by using a resource adapter, deploy the resource adapter as a J2EE resource adapter. A J2EE resource adapter is the resource adapter deployed on a J2EE server. For details on how to deploy the resource adapters, see *3.3.7 How to set up a resource adapter* in the *uCosminexus Application Server Common Container Functionality Guide*.

(1) Resource adapter functionality

The following table describes the functionality that you can use for connecting to a database, in the case of a batch server. For details on the functionality, see description at respective *Reference location*.

Table 2–21: Resource adapter functionality

Functionality	Field	Explanation	Reference manual#1	Reference location
Functionality for performance tuning	Connection pooling	You can speedily process connection requests from the application by pooling the connections in memory.	*Common Container Functionality Guide*	*3.14.1*
	Warm-up of connection pool	Create connections of the number that is specified when you start a server or a resource adapter. By pooling the connections, speedily process the connection requests immediately after starting the connection pool.		*3.14.2*
	Functionality for adjusting number of connections	This functionality gradually decreases unnecessary connections in the pool, at regular intervals.		*3.14.2*
	Connection sharing association	By sharing connections, you can decrease the processing time required for acquiring connections. In connection sharing, connect logical connections and physical connections that are the connections of the connection destination resources, as many-to-one. However, you cannot use connection association in a batch application.		*3.14.3*
	Statement pooling	In the processing that uses `PreparedStatement` and `CallableStatement`, you can pool these statements and shorten the processing time required for creating the same statements.		*3.14.4*
	Caching of DataSource objects	When making search request of DataSource type object by using JNDI interface, you can cache DataSource objects.		*3.14.7*
	Optimizing sign-on in container management of DB Connector	If you want to perform sign-on in container management, you can optimize on the sign-on operation.		*3.14.8*

Functionality	Field	Explanation	Reference manual#1	Reference location
Functionality for fault tolerance	Detecting connection failure	You can detect whether trouble has occurred in pooled connections. As a result, you can return only valid connections against connection requests from user program.	*Common Container Functionality Guide*	*3.15.1*
	Waiting for acquiring connections when connections exhaust	You can set up the connection acquisition requests to standby if connections of the specified maximum value are pooled in the connection pool and there are no connections that you can use.		*3.15.2*
	Retrying connection acquisition	If there are no connections that you can use, in the connection pool, or if establishing a physical connection of connection destination resource fails, you can automatically re-execute the processing of acquiring connection.		*3.15.3*
	Displaying connection pool information	You can display connection information in the connection pool using a command.		*3.15.4*
	Clearing connection pool	If trouble occurs on a database server and the connection is disconnected, you can delete the unnecessary connection pools with a command.		*3.15.5*
	Cancelling statements	You can cancel a statement if transaction timeout occurs when the running SQL processing has not returned.		*3.15.8*
	Outputting SQL for failure investigation	If failure such as deadlock and slowdown occurs, you can output the issued SQL to log. You can use the log for analyzing the cause of failure.		*3.15.10*
	Automatic closing of objects	If you could not close the Statement objects opened by user program, DB Connector can automatically close the objects.		*3.15.11*
Testing connection to resources	Testing connection to resources	You can check whether a resource adapter is correctly specified during the environment setup.		*3.17*
Assigning optional name to Enterprise Bean or J2EE resources (user-specified namespace functionality)	Assigning optional name to J2EE resource#2	You can assign optional name to J2EE resources. You can perform lookup from batch application, on any name specified as an optional name.		*2.6*
Performance analysis of system by using performance analysis trace	PRF trace of connection ID	This functionality collects performance analysis information output by the functionality. Based on this information, you can analyze system performance and bottlenecks.	*Maintenance and Migration Guide*	*Chapter 8*

#1 *uCosminexus Application Server* is omitted in the manual name mentioned in the *Reference manual* column.

#2 For a batch server, make sure to use the optional name of the resource adapter.

The following table describes functionality that you can use for each type of resource adapter.

Table 2–22: Functionality that you can use for each type of resource adapter

Functionality	Field	Type of resource adapter
		DB Connector
Functionality for performance tuning	Connection pooling	Y
	Warm-up of connection pool	Y
	Functionality for adjusting number of connections	Y
	Connection sharing association#	Y
	Statement pooling	Y
	Caching of DataSource objects	Y
	Optimizing sign-on in container management of DB Connector	Y
Functionality for fault tolerance	Detecting connection failure	Y
	Waiting for acquiring connections when connections exhaust	Y
	Retrying connection acquisition	Y
	Displaying connection pool information	Y
	Clearing connection pool	Y
	Cancelling statements	Y
	Outputting SQL for failure investigation	Must be enabled
	Automatic closing of objects	Y
Testing connection to resources	Testing connection to resources	Y
Assigning optional name to Enterprise Bean or J2EE resources (user-specified namespace functionality)	Assigning optional name to J2EE resource#	Y
Performance analysis of system by using performance analysis trace	PRF trace of connection ID	Y

Legends:

Y: Can be used

You cannot use connection association with a batch application.

(2) Functionality other than resource adapter

This subsection describes the functionality other than the implemented resource adapters. You can use the functionality described here irrespective of the type of resource adapter.

The following table describes the functionality other than the implemented resource adapters. For details on the functionality, see *Reference location*.

Table 2–23: Functionality other than resource adapter

Functionality	Field	Explanation	Reference manual#	Reference location
Functionality for performance tuning	Light transaction	This functionality provides an environment optimized with local transactions. Make sure to enable the light transaction functionality.	*Common Container Functionality Guide*	*3.14*
Functionality for fault tolerance	Transaction timeout	This functions rolls back the transactions at invoke destination when a fixed time elapses after transaction start time.	*Common Container Functionality Guide*	*3.15*

uCosminexus Application Server is omitted in the manual name mentioned in *Reference location*.

Important note

The transaction management functionality on a J2EE server includes the functionality for automatically concluding transactions. However, you cannot use the functionality of automatically concluding transactions on a batch server.

(3) Notes for optional name of resource adapter

If you have deployed multiple resource adapters with the same optional name, an error message is output and an attempt to start the resource adapters fails.

2.7.5 How to set up resource adapters

To connect to a database from a batch application, use a resource adapter called *DB Connector*. This subsection describes setting of resource adapter, which is used on a batch server. On a batch server, deploy and use a resource adapter as a J2EE resource adapter.

Reference note

If resource adapter is DB Connector, you can use template file of Connector property file provided with Application Server. If you use the template file of Connector property file, you can edit the Connector property file before importing DB Connector. As a result, the operation of acquiring Connector property file to be edited, by using the server management command (`cjgetrarprop` or `cjgetresprop` command) is no more required. Templates in Connector property file are stored in the following locations. Copy and use the template files.

- In Windows,
  Cosminexus-installation-directory `\CC\admin\templates\`
- In UNIX,
  `/opt/Cosminexus/CC/admin/templates/`

For details on template files of Connector property file and points to be considered when using the template files, see *4.1.13 Template files of the HITACHI Connector Property File* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

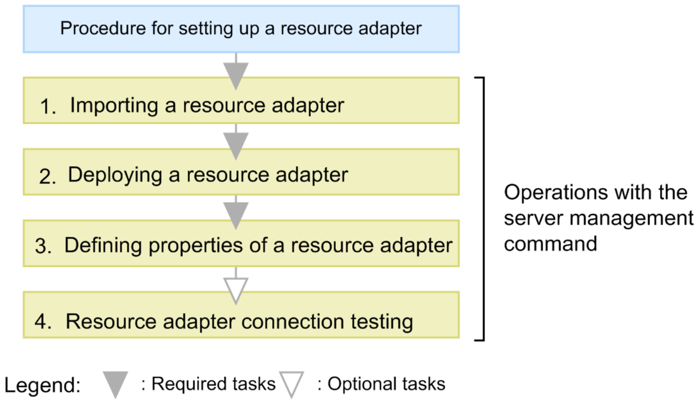
> **Important note**
>
> To use the resource adapter that was used with Application Server of an earlier version, you must perform migration for the resource adapter. For details about how to perform resource migration, see *Chapter 10* in the *uCosminexus Application Server Maintenance and Migration Guide*.

2.7.6 Procedure for setting a resource adapter

You use the server management commands to set up resource adapters. With a batch server, you deploy and use a resource adapter as a J2EE resource adapter.

The following figure shows the flow of new settings of a resource adapter, to be used on a batch server.

Figure 2–12: Flow of new settings of a resource adapter, to be used on a batch server



The following subsections describe steps from 1 through 4 of the above figure:

1. Import a resource adapter by using a server management command.

   You import a resource adapter by using the `cjimportres` command.

   For details on the resource adapters to be imported, see *2.7.3 Types of DB Connector (RAR file)*.

2. Deploy a resource adapter by using a server management command.

   You deploy a resource adapter by using the `cjdeployrar` command.

   If you deploy a resource adapter, you can use the resource adapter as a J2EE resource adapter. *J2EE resource adapter* is a resource adapter that is deployed as a shared stand-alone module on a batch server. If you deploy the resource adapter imported by the server management command, you can use the resource adapter on the batch server.

3. Define resource adapter properties by using server management commands.

   Acquire Connector property file with the `cjgetrarprop` command, edit the file and reflect edited contents with the `cjsetrarprop` command.

   In the case of a batch server, you specify an optional name to the resource adapter by using the user-specified namespace functionality. You define the settings of optional names performed with the user-specified namespace functionality in property of the resource adapters. For details on settings of the user-specified namespace functionality, see *2.6 Assigning optional name to Enterprise Bean or J2EE resource (user-specified namespace functionality)* in the *uCosminexus Application Server Common Container Functionality Guide*.

   For details on the contents that you can specify in the property definition of resource adapters, see *2.7.7(2) Resource adapter settings*.

4. Test connection of the resource adapter by using a server management command.
   Execute the test connection of resource adapters by using the `cjtestres` command. For details on the validation contents of Connection Test performed for the resources, see *3.17 Testing connection to resources* in the *uCosminexus Application Server Common Container Functionality Guide*.

For details on the operations with the server management commands, see *3. Basic Operations of Server Management Commands* in the *uCosminexus Application Server Application Setup Guide*. For details on the `cjimportres` command, see *cjimportres (importing resources)* in the *uCosminexus Application Server API Reference Guide*. For details on the `cjdeployrar` command, see *cjdeployrar (deploying resource adapters)* in the *uCosminexus Application Server API Reference Guide*. For details on the `cjgetrarprop` command, see *cjgetrarprop (acquiring properties of RAR file)* in the *uCosminexus Application Server API Reference Guide*. For details on the `cjtestres` command, see *cjtestres (testing connections of resources)* in the *uCosminexus Application Server API Reference Guide*. For details on the properties, see *4. Property Files Used for Setting Resources* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

For details on the following procedures, see *3.3.8 Procedure for setting resource adapters (when deploying and using as J2EE resource adapter)* in the *uCosminexus Application Server Common Container Functionality Guide*. In that case, read *J2EE server* as *Batch server* and *J2EE application* as *batch application*.

- Procedure for changing settings of a resource adapter
- Procedure for replacing a resource adapter

Reference note

In the following cases, you can efficiently specify a resource adapter by exporting and importing:

- If you export a resource adapter, which is specified in development environment, import it to operating environment, and then use it
- If you export a resource adapter, which is already running in operating environment, import it to add-on batch server, and then use it

You execute export and import with `cjexportrar` and `cjimportres`.

You cannot export and import, and use resource adapters between the hosts having different version and platform of Application Server. Set a new resource adapter when setting a resource adapter on a host, which exports resource adapters, and a host having different version and platform of Application Server.

2.7.7 Settings in the execution environment

You must perform batch server and resource adapter settings if you use resource connection functionality.

This subsection describes the settings for using the resource connection functionality.

(1) Batch server settings

You implement the batch server settings in the Easy Setup definition file. You specify the definition of batch application execution functionality in the `<configuration>` tag of logical J2EE server (j2ee-server) of the Easy Setup definition file.

The following table describes the definition of resource connection functionality in Easy Setup definition file.

Table 2–24: Definition of resource connection functionality in Easy Setup definition file

Field	Parameter to be specified	Setting contents
Enabling connection sharing outside the transactions managed by Application Server	`ejbserver.connectionpool.sharingOutsideTransactionScope.enabled`	Specify operation of connection sharing to be performed when you acquire multiple connections outside the transactions managed by Application Server.
Caching of DataSource objects	`ejbserver.jndi.cache.reference`	Specify whether the caching of DataSource object is to be enabled.
Optimizing sign-on in container management of DB Connector	`ejbserver.connectionpool.applicationAuthentication.disabled`	Specify whether the sign-on optimization functionality of container management is to be enabled.

Note: For details on the Easy Setup definition file and parameters, see *4.3 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

(2) Resource adapter settings

In the case of a batch application to be connected to resources, you specify an optional name to resource adapters by using the user-specified namespace functionality. To lookup resource adapters from a batch application, you use the optional name set up with user-specified namespace functionality. For details on the settings of the user-specified namespace functionality, see *2.3.8 Implementing a batch application (When connecting to resources)*.

Reference note

Before performing resource settings, understand the notes on resource settings. When using server management commands, customize operation settings of server management commands as and when required. For notes on the resource settings and operation settings for using the server management commands, see *3.3 Customizing operation settings of server management commands* in the *uCosminexus Application Server Application Setup Guide*.

You implement resource adapter settings in the HITACHI Connector Property file.

The following table describes the definition of resource connection functionality in the HITACHI Connector Property file.

Table 2–25: Definition of resource connection functionality in the HITACHI Connector Property file

Category	Field	Setting contents
General information	Transaction support level	Set transaction support level in the `<transaction-support>` tag. You specify no transaction management (`NoTransaction`) or local transaction (`LocalTransaction`). You cannot specify global transaction (`XATransaction`) for batch servers.
Configuration properties	Waiting time until database connection is established	Specify a waiting time of a batch application until a database connection is established, in `loginTimeout` of the `<config-property>` tag.
	Cancelling statements	Specify whether the statement cancellation performed when a transaction timeout occurs is to be enabled, in `CancelStatement` of the `<config-property>` tag.
	Pool size of PreparedStatement[#1]	Specify pool size of `PreparedStatement`, in `PreparedStatementPoolSize` of the `<config-property>` tag.

Category	Field	Setting contents
	Pool size of CallableStatement[#1]	Specify pool size of `CallableStatement`, in `CallableStatementPoolSize` of the `<config-property>` tag.
Runtime properties	Minimum value and maximum value of a connection	Specify minimum value and maximum value of connections to be pooled in the connection pool, in `MinPoolSize` and `MaxPoolSize` of the `<property>` tag.
	Detecting failure in a connection	Specify a timing of detecting connection failure, in `Validation`Type of `<property>` tag. Specify failure detection interval in `ValidationInterval` When setting a timeout when connection failure is detected, enable the usage of connection management thread in `NetworkFailureTimeout`. [#2]
	Connection acquisition retry	Specify retry count in case of connection acquisition failure, in `RetryCount` of the `<property>` tag. Specify retry intervals in `RetryInterval`.
	Connection sweeper	Specify an interval for automatically destroying connections (connection sweeper) in `SweeperInterval` of the `<property>` tag. Specify a time from last usage time of connection to judging whether the connection is to be automatically destroyed, in `ConnectionTimeout`.
	Waiting for acquiring connections when connections exhaust	Specify whether to wait for acquiring connections when connections exhaust, in `RequestQueueEnable` of the `<property>` tag. Specify waiting time in `RequestQueueTimeout`.
	Warm-up of connection pool	When using the warm-up functionality of connection pool, specify `Warmup` in the `<property>` tag.
	Connection management threads	When using connection management threads, specify `NetworkFailureTimeout` in the `<property>` tag. When using connection management threads, settings for using a timeout in the connection failure detection functionality and functionality for adjusting number of connections are enabled.
	Functionality for adjusting number of connections	Specify an interval for operating the functionality for adjusting number of connections, in `ConnectionPoolAdjustmentInterval` of the `<property>` tag. When setting a timeout for the functionality for adjusting number of connections, you enable the usage of connection management threads in `NetworkFailureTimeout`. [#2]

Note: For details on the HITACHI Connector Property file, see *4. Property Files Used for Setting Resources* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

#1: In the case of XDM/RD E2 11-01 or earlier versions, specify `0` in these properties, because you cannot use the statement pooling functionality.

#2: Set up with the same key. Therefore, when you use a timeout with the connection failure detection functionality, you use of a timeout is also enabled in the functionality for adjusting number of connections. For timeout time, specify any time (default value is `5` seconds) in the key (`ejbserver.connectionpool.validation.timeout`), specified in a J2EE server of the Easy Setup definition file.

For details on the definition of DB Connector properties specified when connecting to a database by using DB Connector, see *4.1.2 Overview of settings and operations* in the *uCosminexus Application Server Application Setup Guide*.

2.8 Transaction management

This section describes the transaction management when connecting to resources.

The following table describes the organization of this section.

Table 2–26: Organization of this section (Transaction management)

Category	Title	Reference location
Description	Overview of transaction management when connecting to resources	*2.8.1*
Setup	Settings in execution environment (batch server settings)	*2.8.2*

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

2.8.1 Overview of transaction management when connecting to resources

There are two methods of managing transactions when connecting to resources; managing transactions with Application Server and directly managing transactions by a user without managing with Application Server. When connecting to a database, you can manage transactions by using transaction manager with Application Server. For details on the transaction management, see *3.4.1 How to manage transactions in resource connection* in the *uCosminexus Application Server Common Container Functionality Guide*.

A transaction that can be managed with a batch server is a local transaction. You cannot use global transactions. Make sure to enable the light transaction functionality on a batch server. Light transaction functionality is a functionality which provides an environment, which is optimized with local transactions. For details on the local transactions and light transaction functionality, see *3.4.2 Local transactions and global transactions* in the *uCosminexus Application Server Common Container Functionality Guide*.

When invoking EJBs, if a system exception occurs at invocation destination, transactions at invocation source and invocation destination operate as follows.

Transactions at invocation source

Transactions are not marked for rollback.

Transaction at invocation destination

Transactions are rolled back by container. These operations are defined in EJB specifications.

2.8.2 Settings in the execution environment (Batch server settings)

You must perform batch server settings, if you want to use transaction management functionality.

You implement the batch server settings in the Easy Setup definition file. You specify the definition of the transaction management functionality in the `<configuration>` tag of a logical J2EE server (j2ee-server) of the Easy Setup definition file. You specify the following parameters:

- `ejbserver.jta.TransactionManager.defaultTimeOut`

  You specify the default value of a timeout for transactions started on a batch server.

For details on a transaction timeout, see *3.15 Functionality for fault tolerance* in the *uCosminexus Application Server Common Container Functionality Guide*. For details on the Easy Setup definition file and parameters, see *4.3 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

2.9 GC control functionality

The GC control functionality is available on batch servers. This section provides an overview, shows the processing flow, and describes the setup procedure for the GC control functionality.

The following table describes the organization of this section.

Table 2–27: Organization of this section (GC control functionality)

Category	Title	Reference location
Description	Overview of the GC control functionality	*2.9.1*
	Processing flow of GC control	*2.9.2*
Setup	Settings in the execution environment (batch server settings)	*2.9.3*

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

2.9.1 Overview of the GC control functionality

GC (garbage collection) is a technology that automatically reclaims the memory areas that were used by programs and makes those memory areas available again to other programs. GC is performed by a Java VM.

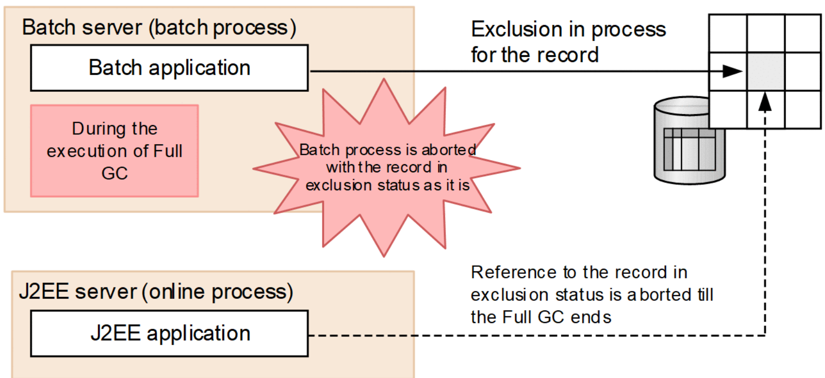
The processing of GC takes time. All programs running on a Java VM stop while the Java VM is performing GC. Therefore, whether GC is performed properly greatly affects the processing performance of the system.

A batch server provides the GC control functionality to prevent a batch application from excluding a resource for a long time. The GC control functionality explicitly performs Full GC while no resources are excluded. This functionality can prevent Full GC from occurring while resources are excluded.

The following describes the GC control functionality by using an example.

If the GC control functionality is not used in an environment in which batch processing and online processing are performed concurrently, the problem shown in the following figure can occur.

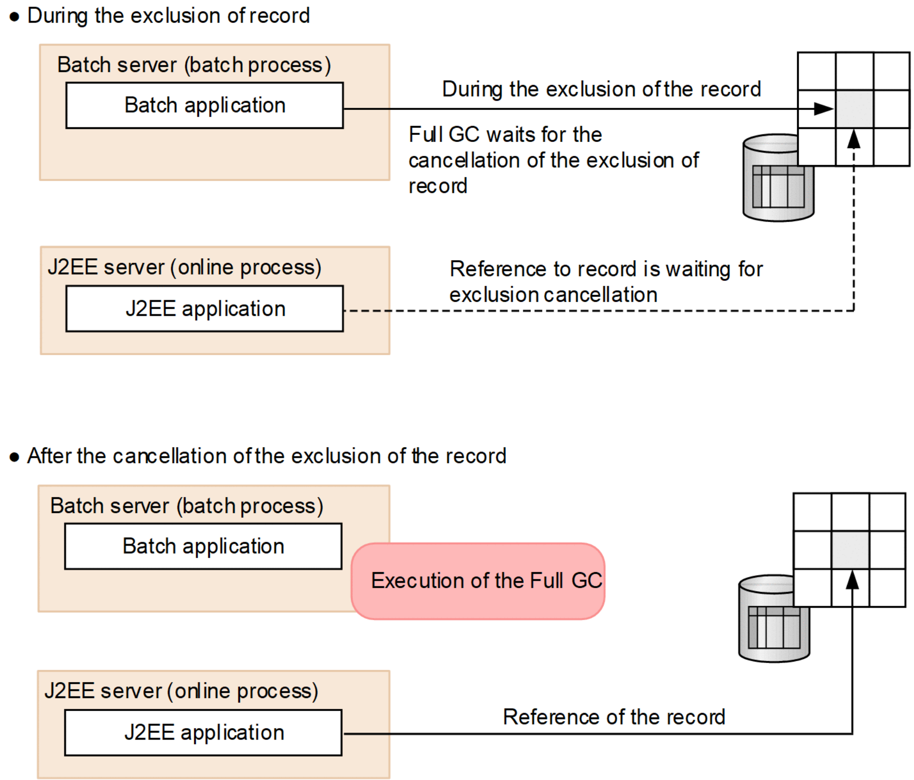
Figure 2–13: Case where the GC control functionality is not used



In the preceding figure, Full GC occurs while a resource is excluded by a batch application. As a result, a batch application processing stops when performing the resource exclusion. In this situation, if online processing accesses the excluded resource (record), the online processing also stops until the Full GC on the batch server finishes.

The following figure shows a case where the GC control functionality is used.

Figure 2–14: Case where the GC control functionality is used



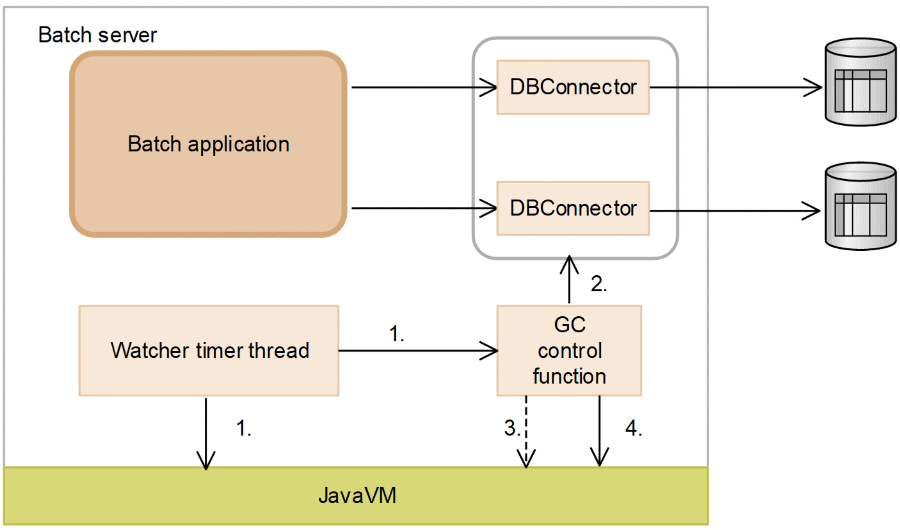
As shown in the preceding figure, if a resource (record) is excluded by a batch application when execution of Full GC is requested, Full GC begins to wait.

When the record is not excluded, Full GC occurs on the batch server. The online processing is also able to access the resources. As a result, you can avoid long time resource exclusion in a batch application.

2.9.2 Processing flow of GC control

GC control is processed as shown in the following figure.

Figure 2–15: Processing flow of GC control



1. Monitoring memory
   Monitoring timer thread monitors memory of JavaVM. The GC control functionality is requested to perform GC when certain conditions are met. For the conditions, see section (1) shown later.
2. Checking resource exclusion
   When requested to perform GC, the GC control functionality checks whether excluded resources exist.
3. Placing Full GC in a wait state
   Full GC is placed in a wait state while resources are excluded.
4. Performing Full GC
   When the resources are not excluded, Full GC starts.

The following subsections describe each process:

(1) Monitoring memory

The monitoring timer thread monitors the memory used by the Java VM and, when any of the following conditions are met, requests the GC control functionality to perform GC:

If serial GC is enabled:

- Size of the used Tenured area / Size of the entire Tenured area × 100 ≥ GC control threshold value
- Size of the entire New area / Maximum size of free Tenured area × 100 ≥ GC control threshold value
- Size of the used Metaspace area / Maximum size of the Metaspace area × 100 ≥ GC control threshold value

If G1 GC is enabled:

- Size of the used Java heap area / Size of the entire Java heap area × 100 ≥ GC control threshold value
- Size of the used Metaspace area / Maximum size of the Metaspace area × 100 ≥ GC control threshold value

(2) Checking resource exclusion

When requested to perform GC, the GC control functionality runs a check on the connections used by the batch application. When checking the connections, the functionality confirms whether the batch application is performing resource exclusion.

The following table describes states considered as the resources are in exclusion.

Table 2–28: States considered as resource is under exclusion

Transaction	State		DB Connector	JDBC
Out of transaction	Executing SQL statement#1	• When executing `java.sql.Statement#execute` • When executing `java.sql.Statement#executeUpdate` • When executing `java.sql.Statement#executeQuery` • When executing `java.sql.Statement#executeBatch`	Y	N
	Performing operations for ResultSet	• When executing `java.sql.ResultSet#deleteRow` • When executing `java.sql.ResultSet#insertRow` • When executing `java.sql.ResultSet#updateRow`	Y	N
	Performing operation such as	• When executing `java.sql.Statement#addBatch`	Y	N

Transaction	State		DB Connector	JDBC
	object acquisition#1	• When executing `java.sql.Connection#prepareCall` • When executing `java.sql.Connection#prepareStatement`		
During transaction	• When executing transaction with Connection API#2 • When executing local transaction (JTA)#2.		Y	N
	When executing global transaction (JTA)		--	--

Legend: Y: Handled as resource under exclusion

N: Handled as resource not under exclusion

--: Not applicable

#1: `java.sql.Statement` in the table -includes sub-interfaces `java.sql.PreparedStatement` and `java.sql.CallableStatement`.

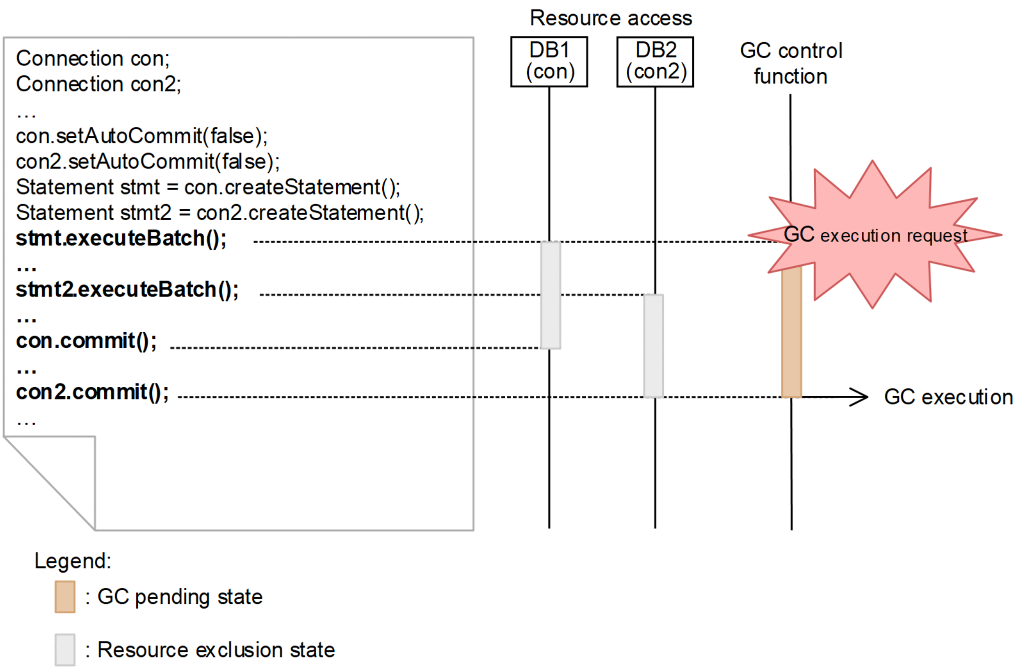
#2: Shows a status after starting a transaction (after executing `setAutoCommit(false)` or `UserTransaction.begin`), in which execution of SQL statement or operation for `ResultSet` is performed one or more times, and transaction conclusion processing is not complete.

Resource operations performed by using JDBC are handled as no resource exclusion. For example, if you execute a program that includes execution of SQL statements for JDBC and transaction processing in DB Connector, only the transaction processing in DB Connector is subject to GC control.

(3) Placing Full GC in a wait state

If the functionality judges that excluded resources exist, it outputs the KDJE55024-I message and places Full GC in a wait state. Full GC continues to wait while there is at least one excluded resource. The following figure shows an example case where Full GC is placed in a wait state.

Figure 2–16: Example case where Full GC is placed in a wait state



In this figure, two resources are accessed in a single job program. If the GC control functionality is requested to perform Full GC while resources are excluded, the functionality places Full GC in a wait state. When `con2.commit()` that ends the access of two resources is executed, the exclusion is removed.

(4) Performing GC

When no resources are excluded, the functionality performs Full GC.

(5) Notes

- You can concurrently execute only one batch application.
- You can execute processing to multiple resources from one batch application. However, you cannot use global transactions.
- Even when Full GC is placed in a wait state, the Java VM might perform Full GC if free memory becomes insufficient. This is likely to occur when a large threshold value is set for the memory usage during GC or when resources are excluded for a long time. To prevent this, tune the threshold values related to memory usage. For details about how to tune the threshold values, see *9.4 Setting the thresholds for GC control* in the *uCosminexus Application Server System Design Guide*.

2.9.3 Settings in the execution environment (batch server settings)

You must specify the required settings before using the GC control functionality.

You implement the batch server settings in the Easy Setup definition file. In this file, the section in which to specify GC control functionality definitions is the `<configuration>` tag for the logical J2EE server (j2ee-server). In the tag, specify the following parameter:

- `ejbserver.batch.gc.watch.threshold`

  This parameter specifies the threshold value for memory usage as a condition that triggers GC.

For details on the Easy Setup definition file and parameters, see *4.3 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide.*

2.10 Container extension libraries

On a batch server, if you want to use common processing between applications, you can use user-created libraries. You can extend application functionality by using user-created libraries. This section gives an overview and setting method of container extension libraries.

The following table describes the organization of this section.

Table 2–29: Organization of this section (Container extension libraries)

Category	Title	Reference location
Description	Overview of container extension libraries	*2.10.1*
Setup	Settings in the execution environment (batch server settings)	*2.10.2*

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

2.10.1 Overview of container extension libraries

A library that applications can commonly use is called *Container extension library*. You can commonly use these libraries between applications and invoke user-created libraries. Libraries, which are set up in the container extension library are loaded in the system class loader. For details, see *2.3.1 Overview of the batch application execution functionality*.

You can use container extension libraries on a batch server. However, you cannot set up and use a batch application in a container extension library.

You can specify an invocation of container extension libraries when stating and terminating the server by using *Server start/stop hook functionality*. You can initialize the JNI functionality used in container extension libraries.

For using container extension libraries, you compile the libraries in one JAR file and define the settings for using container extension libraries in `usrconf.cfg`. If the container extension libraries use JNI, you need to perform settings for using server start/stop hook functionality.

For an overview of using the container extension libraries, see *16.2 Using container extension libraries* in the *uCosminexus Application Server Common Container Functionality Guide*. For details on how to implement the server start/stop hook functionality, see *16.4.2 How to implement server start/stop hook functionality* in the *uCosminexus Application Server Common Container Functionality Guide*.

Important note

The following access permission is given for container extension libraries. You cannot change the access permissions.

`java.security.AllPermission`

However, the access permission of `setSecurityManager` of `java.lang.RuntimePermission` is not given.

2.10.2 Settings in the execution environment (Batch server settings)

You must perform batch server settings if you want to use container extension library functionality.

You implement the batch server settings in the Easy Setup definition file. You specify the definition of container extension library functionality in the `<configuration>` tag of a logical J2EE server (j2ee-server) of the Easy Setup definition file. The parameters to be specified are as follows:

- `add.class.path`
  Specify path of JAR file of the container extension library.
- `add.library.path`
  Specify search path of library for JNI.

For details on the Easy Setup definition file and parameters, see *4.3 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

For details on setting method for using container extension library functionality, see *16.3.3 Settings for using container extension library functionality* in the *uCosminexus Application Server Common Container Functionality Guide*.

2.11 JavaVM functionality

This section describes JavaVM functionality.

The following table describes the organization of this section.

Table 2–30: Organization of this section (JavaVM functionality)

Category	Title	Reference location
Description	Overview of JavaVM functionality	*2.11.1*
Setup	Settings in the execution environment (batch server settings)	*2.11.2*

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

2.11.1 Overview of JavaVM functionality

The processes of a batch server that operates with Application Server are executed on JavaVM.

JavaVM is an independent JavaVM provided by Cosminexus Developer's Kit for Java that is the component software. The following table describes JavaVM functionality. For details on the functionality, see *Reference location*.

Table 2–31: JavaVM functionality

Functionality	Explanation	Reference manual#	Reference location
Explicit Memory Management functionality	You can place Java objects for which Full GC can occur in the Explicit heap area. This can prevent Full GC from occurring due to Java objects used in the application.	This manual	*Chapter 7*
Class-wise statistics functionality	You can output size of all instances in the members, possessed by instance of each class, to extended thread dump as class-wise statistics. By outputting class-wise statistics multiple times, you can check, for example, how Java objects change when GC occurs and the states of short-life Java objects. The functions that output class-wise statistics include the following: instance statistical functionality, static member statistical functionality, reference-related information output functionality, function for selecting GC prior to statistics output, unused objects statistical functionality in the Tenured area, base object list output functionality for Tenured augmentation factors.	*Maintenance and Migration Guide*	*9.3*
Class-wise statistics analysis functionality	Based on class-wise statistics output to extended thread dump, you can output total size of instances for each class and number of instances for each class as two types CSV files.		*9.10*
Functionality for output age distribution information of Survivor area	The age distribution information for the Java objects in the Survivor area can be output to a Java VM log file when copy GC occurs. You can check usage status of Survivor area and use for tuning the memory size.		*9.11*
hndlwrap functionality	You can inhibit occurrence of logoff events of JavaVM during logoff.		*9.12*

uCosminexus Application Server is omitted in the manual name mentioned in *Reference manual*.

In JavaVM, log output contents are extended so that you can use the contents for analyzing the causes of failures and checking system status. This log is output to JavaVM log file. You can acquire a lot of troubleshooting information also from standard JavaVM. The availability of a system can be improved by using this log (extended `verbosegc` information) and performing appropriate tuning. For details on JavaVM log file, see *4.10 JavaVM log (JavaVM log file)* in the *uCosminexus Application Server Maintenance and Migration Guide*. For details on the JavaVM tuning, see *7. JavaVM Memory Tuning* in the *uCosminexus Application Server System Design Guide*.

2.11.2 Settings in the execution environment (Batch server settings)

You must perform batch server settings if you want to use JavaVM functionality.

You implement the batch server settings in the Easy Setup definition file. You specify the definition of JavaVM functionality in `<configuration>` tag of a logical J2EE server (j2ee-server) of the Easy Setup definition file.

The following table describes the definition of JavaVM functionality in Easy Setup definition file.

Table 2–32: Definition of JavaVM functionality in the Easy Setup definition file

Field	Parameter to be specified		Setting contents
	Parameter name	Parameter value	
Using the Explicit Memory Management functionality	`add.jvm.arg`	`-XX:+HitachiUseExplicitMemory`	If the Explicit Memory Management functionality is implemented in a batch application, you specify the use of the Explicit Memory Management functionality. For details on the JavaVM options that you can specify when using the Explicit Memory Management functionality, see *7.13.1 Common settings for using the Explicit Memory Management functionality (setting JavaVM options)*
Output age distribution information of Survivor area	`add.jvm.arg`	`-XX:+HitachiVerboseGCPrintTenuringDistribution`	The age distribution information for the Survivor area is output to a Java VM log file when copy GC occurs.
Acquiring JavaVM log (JavaVM log file)	`add.jvm.arg`	`-XX:+HitachiOutOfMemoryStackTrace`#	Specify output of exception information and stack trace to JavaVM log file.
		`-XX:+HitachiVerboseGC`#	Extended `verbosegc` information is output to a Java VM log file when GC occurs.
		`-XX:+HitachiJavaClassLibTrace`#	Specify output of stack trace of class libraries to JavaVM log file.

Note: For details on the Easy Setup definition file and parameters, see *4.3 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

#: Even if you specify any one, JavaVM log file is output. If you specify `-XX:+HitachiOutOfMemoryStackTrace`, `-XX:+HitachiOutOfMemorySize` and `-XX:+HitachiOutOfMemoryCause` are specified simultaneously.

2.12 Migrating from Java applications

You can execute Java applications, which you execute using the `cjclstartap` command provided with Application Server, as batch applications on batch servers. If you want to execute the Java applications as batch applications on batch servers, you might require to migrate the applications and the execution environment. This section describes the cases where migration is required and how to migrate applications and the execution environment.

The following table describes the organization of this section.

Table 2–33: Organization of this section (Migrating from Java applications)

Category	Title	Reference location
Implementation	Implementing batch applications (migrating from Java applications)	*2.12.1*
Setup	Settings of the execution environment (setting batch servers)	*2.12.2*

Note: There is no specific explanation of *Description, Operation*, and *Precautions* for this functionality.

2.12.1 Implementing batch applications (Migrating from Java applications)

This section describes the changes required in Java applications for migrating the Java applications to batch applications.

You must migrate the Java applications in the following cases:

- If you are implementing a process that corresponds to the points to be considered when using batch applications

  You must consider the operations of files and directories while implementing with the batch applications.

  How to migrate

  The processes of batch applications that require attention are described in *2.3.11(1) Processes that require attention*. Reference these contents, and modify Java application.

- If you are implementing a functionality that cannot be used with batch applications

  You cannot use many functionality with batch applications. For example, you cannot input from the standard input format or use JNI.

  How to migrate

  For the functionality that cannot be used with batch applications and the alternative methods for using the functionality, see *2.3.11(2) Functionality that you cannot implement in batch applications*. Reference these contents, and modify Java applications.

- If the unsupported properties are defined in `usrconf.properties` (user property file for batch applications)

  With batch applications, you can continue to use the `usrconf.properties` file (user property file for Java applications) that you use with the Java applications that are migrated.

  However, in the `usrconf.properties` file (user property file for Java applications), if you have defined the properties that are not supported with `usrconf.properties` (user property file for batch applications)# and you are referencing the values from the batch applications, you must modify the applications.

  How to migrate

  Modify the batch application in such a way so that you do not reference the properties that are not supported by `usrconf.properties` (user property file for batch applications).

#

This excludes the user-defined property. For details on the properties supported by `usrconf.properties` (user property file for batch applications), see *3.2.6 usrconf.properties (User property file for batch applications)* in the *uCosminexus Application Server Definition Reference Guide*.

2.12.2 Settings of the execution environment (Setting batch servers)

When you want to migrate Java applications to batch applications, you might need to change the settings of batch servers. This section describes the cases where you need to change setting of the batch servers.

You can use the following two files, which are used in the execution environment of Java applications until now, as it is in the execution environment of a batch server:

- `usrconf.cfg` (option definition file for Java applications)
- `usrconf.properties` (user property file for Java applications)

However, you must migrate the files, if corresponding to the following conditions:

- If you set up a storage location of `usrconf.cfg` (option definition file for Java applications) and `usrconf.properties` (user property file for Java applications) in the `CJCLUSRCONFDIR` environment variable

  How to migrate

  Specify a storage location of `usrconf.cfg` (option definition file for batch applications) and `usrconf.properties` (user property file for batch applications) in the `CJBATCHUSRCONFDIR` environment variable with an absolute path.

- If you specify an option other than `-cp`, `-classpath`, and `-D` in `add.jvm.arg` of `usrconf.cfg` (option definition file for Java applications)

  How to migrate

  Describe the option settings in `usrconf.cfg` (option definition file for batch applications). When you want to execute multiple batch applications in a sequence, on one batch server, you must adjust the definition settings. An example is given below. In the example, the value of application 2, for which a greater value is specified, is set up on a batch server.

  For example: If `add.jvm.arg=-Xmx512m` is set up in application 1 and `add.jvm.arg=-Xmx768m` is set up in application 2,

  specify `add.jvm.arg=-Xmx768m` on the batch server.

- If `ejb.client.log.directory` is specified in `usrconf.cfg` (option definition file for Java applications) and the log output location is changed from the default value

  How to migrate

  Specify `batch.log.directory` in `usrconf.cfg` (option definition file for batch applications) and set up an output location for logs other than the default location.

- If `ejb.client.ejb.log` or `ejb.client.log.appid` is specified in `usrconf.cfg` (option definition file for Java applications) and the log output location is changed from the default value

  How to migrate

  There is no method for migration. In the case of a batch server, you cannot specify a log output location that is specified by using `ejb.client.ejb.log` and `ejb.client.log.appid`.

- If ejb.client.directory.shareable=true is specified in usrconf.cfg (option definition file for Java applications) and multiple applications are executing concurrently

  How to migrate

    You cannot concurrently execute multiple batch applications on one batch server. Therefore, prepare the same number of batch servers as the maximum number of batch applications that will be concurrently executing. Change the server name specified in the cjexecjob command in such a way so that the batch applications operate on the respective batch servers.

- If the unsupported properties are defined in usrconf.properties (user property file for batch applications)

  In usrconf.properties (user property file for Java applications), if you have defined the properties not supported by usrconf.properties (user property file for batch applications)#, you must modify usrconf.properties (user property file for Java applications).

  How to migrate

    Delete the definition of the properties, not supported by usrconf.properties (user property file for batch applications), from usrconf.properties (user property file for Java applications).

#

This excludes the user-defined property. For details on the properties that are supported by usrconf.properties (user property file for batch applications), see *3.2.6 usrconf.properties (User property file for batch applications)* in the *uCosminexus Application Server Definition Reference Guide*.

2.13 Integrating with JP1/AJS

You can operate a system that executes batch applications, by integrating with JP1/AJS. You can also operate the system by using BJEX or JP1/Advanced Shell, besides using JP1/AJS. This section describes the settings for integrating with JP1/AJS, BJEX, and JP1/Advanced Shell.

The following table describes the organization of this section.

Table 2–34: Organization of this section (Integrating with JP1/AJS)

Category	Title	Reference location
Setup	Settings for integrating with JP1/AJS	*2.13.1*
	Settings for integrating with JP1/AJS, BJEX, and JP1/Advanced Shell	*2.13.2*

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

Reference note

For an overview of the systems integrated with JP1/AJS and the systems integrated with JP1/AJS, BJEX, and JP1/Advanced Shell, see *2.2.1 Systems executing batch applications* and *2.2.2 Procedure for operating batch servers and batch applications*.

2.13.1 Settings for integrating with JP1/AJS

This subsection describes the definition of JP1/AJS jobs, when systems are integrated with JP1/AJS.

Start a batch server in advance when executing a batch application from JP1/AJS.

(1) Starting a batch application

When you want to integrate the systems with JP1/AJS, you define the `cjexecjob` command as a UNIX job or PC job of JP1/AJS. Specify the following contents in **Script file name**, **Parameter**, and **User at the time of execution** fields on the window used for defining the properties of a JP1/AJS job.

- Script file name
  Specifies the `cjexecjob` command. For details on the path of the `cjexecjob` command, see *cjexecjob (Execute batch application)* in the *uCosminexus Application Server Command Reference Guide*.
- Parameter
  Specifies class name and arguments of the batch applications to be executed.
- User at the time of execution
  Specifies the user who executes a batch server.

For details on the settings of JP1/AJS, see the *JP1/Automatic Job Management System Operation Guide*.

(2) Forced termination of a batch application

When the system is integrated with JP1/AJS, and you forcefully terminate a job-net or a job, you define the `cjkilljob` command as a recovery job of JP1/AJS. However, when you want to forcefully stop a root job-net, the recovery job is not executed. Therefore, the batch application that is executing on a batch server continues as it is. In such cases, directly execute the `cjkilljob` command and forcefully stop the batch application.

For details on the settings of JP1/AJS, see the *JP1/Automatic Job Management System Operation Guide*.

2.13.2 Settings for integrating with JP1/AJS, BJEX, and JP1/Advanced Shell

This subsection describes the definition of JP1/AJS, BJEX, and JP1/Advanced Shell jobs, when integrating with AJS, BJEX and JP1/Advanced Shell.

Start the batch server in advance when executing the batch job applications of BJEX or JP1/Advanced Shell from JP1/AJS.

(1) Starting a batch application

When you want to integrate the systems with JP1/AJS, BJEX and JP1/Advanced Shell, specify the following contents for JP1/AJS, BJEX, and JP1/Advanced Shell respectively:

- Settings when integrating with BJEX
  You define the execution of the `cjexecjob` command in a batch job of BJEX. In such cases, define the `cjexecjob` command as a job step of the batch job.
  In addition, you define the following contents for the job definition XML of BJEX.
  - EXEC element
    Sets the definition for executing the `cjexecjob` command.
  - PGM property
    Defines the `cjexecjob` command.
  - PARM property
    Defines the arguments of the `cjexecjob` command. However, the maximum length of an argument conforms to the BJEX specifications.
    For details on the settings in BJEX, see the *uCosminexus Batch Job Execution Server Usage Guide*.
- Settings when integrating with JP1/Advanced Shell
  Use the `adshjava` command with JP1/Advanced Shell. By executing the `adshjava` command in the job definition script of JP1/Advanced Shell, the `cjexecjob` command is invoked and batch application is executed while processing the `adshjava` command. With the `adshjava` command, you can execute batch applications on a specific batch server, as you can specify a batch server name and schedule group name in addition to a class name of a batch application.
  For details on the `adshjava` command, see the manual *JP1/Advanced Shell*.
- JP1/AJS settings
  Defines the execution command of a batch job of BJEX or JP1/Advanced Shell as a job.
  For details on the settings in JP1/AJS, see the *JP1/Automatic Job Management System Operation Guide*.

(2) Forced termination of a batch application

When integrating with BJEX or JP1/Advanced Shell, you can force stop the running batch application automatically just by forcefully stopping the execution command of BJEX or JP1/Advanced Shell batch job. As a result, you do not need to define the recovery job.

3 Scheduling and Load Balancing of Requests Using CTM

This chapter describes the scheduling and load balancing of requests.

A business system needs to be reliable, able to maintain stable processing in the event of a local failure, and responsive to varying business processing demands, as required. To fulfill these requirements, the application server performs processing such as OLTP-based request scheduling and load balancing with clustered servers.

Note that the functionality described in this chapter can be used with only the products that include Component Transaction Monitor (CTM). For details about products compatible with this functionality, see *2.2.1 Mapping between products and the component software* in the manual *uCosminexus Application Server Overview*.

3.1 Topics covered by this chapter

This chapter describes the scheduling and load balancing of requests that CTM can implement. CTM improves system stability and operability by appropriately scheduling the executions of requests from clients, and then distributing these requests to multiple J2EE servers.

For an overview of request scheduling using CTM, see *3.2 Overview of request scheduling using CTM*. For the process configuration for using CTM, see *3.3 Process configuration for using CTM*.

Each section in this chapter describes a specific CTM function. The following table lists the functions described and their respective sections.

Table 3–1: CTM functions

Function name	See
Controlling the flow volume of requests	*3.4*
Controlling priority of requests	*3.5*
Dynamically changing the number of concurrent executions of requests	*3.6*
Controlling and blocking requests	*3.7*
Load balancing of requests	*3.8*
Monitoring the accumulation of requests in a queue	*3.9*
Connection with the TPBroker/OTM client by using the gateway functionality in CTM	*3.10*

You can also collect statistics on CTM operations. For details about how to collect statistics on CTM operations, see *Chapter 10. Collecting CTM Statistics* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

3.2 Overview of request scheduling using CTM

This section provides an overview of request scheduling using CTM.

The application server uses a software component called *Component Transaction Monitor* (CTM) to schedule the executions of requests. CTM controls requests by using a *queue*. The queue that CTM uses to schedule request executions is called a *schedule queue*.

3.2.1 Purpose of request scheduling

In a large-scale business system, many requests might concentrate on the J2EE server that is executing a specific J2EE application. To execute jobs smoothly with load on servers balanced and system availability maintained, the business system needs to distribute requests to multiple destinations and control the volume of requests that flow in any given period. In a configuration in which processing is distributed to multiple J2EE servers, to improve overall system performance, the business system needs to send an issued request to the J2EE server that is least heavily loaded.

Request scheduling allows a system to execute the above required processing, and is key to stable system operation and optimal use of each J2EE server. In addition, overall system availability improves because if a problem occurs in a specific J2EE server, J2EE application, or business-processing program (Enterprise Bean), the system can continue processing in reduced mode by isolating only the affected sections.

Request scheduling allows the application server to provide the following six functions:

- Controlling the flow volume of requests
  By limiting the number of threads that can run concurrently on each J2EE server, load is balanced between J2EE servers, for stable and high throughput.
- Controlling priority of requests
  By assigning priority levels to clients, requests from clients with a high priority can be processed first.
- Dynamically changing the number of concurrent executions of requests
  The maximum number of requests that can be executed concurrently can be changed temporarily without stopping the CTM daemon.
- Controlling and blocking requests
  Maintenance can be performed without stopping the system by stopping acceptance of requests for a specific J2EE application or by stopping the dequeuing of requests. As a result, system availability is improved.
- Load balancing of requests
  Processing is distributed to balance the load between J2EE servers. As a result, overall system performance and availability can be improved.
- Monitoring the accumulation of requests in a queue
  The number of requests contained in the schedule queue can be monitored.

3.2.2 Type of requests that can be controlled by CTM

The requests that can be scheduled by CTM are only calls issued to stateless session beans through a remote interface via RMI-IIOP communication.

Note that CTM cannot schedule the requests shown below.

Requests that cannot be scheduled by CTM:

- Calls to stateful session beans and entity beans
- Calls issued through the local interface and calls to message-driven beans (These calls are not issued via RMI-IIOP communication.)
- Calls to enterprise beans for EJB 3.0 or later

In the case where multiple business-processing programs in the same J2EE application are to be called by requests, use a remote interface only when you want those requests to be scheduled. If the scheduling of those requests is unnecessary, we recommend making calls by using the local interface, based on processing performance considerations.

Whether requests are to be controlled by CTM can be selected for each J2EE application or for each business-processing program (bean) in a J2EE application. For example, to exclude the business-processing programs that have a remote interface from control by CTM, change the settings by defining the relevant properties for the J2EE application. For details about the settings for request scheduling by CTM, see *3.4.2 Settings in the execution environment*.

3.2.3 Client applications that send requests

The following EJB clients can use CTM:

- EJB client applications
- JSP/servlets
- Other enterprise beans

Development of the above software does not require a special interface. Set them to look up the Global CORBA Naming Service linked to the CTM daemon (the CORBA Naming Service specified for the `-CTMINSRef` option of the `ctmstart` command).

Note that the software you develop must be able to switch the target CORBA Naming Service if a specific application server in the system fails. Therefore, code the software so that it resumes processing from `lookup` of JNDI if an exception occurs during processing of `lookup`, `create`, `invoke`, or `remove`.

3.2.4 Processing performed for using CTM

If CTM is enabled, the processing for using CTM is performed at the following times:

- At startup of a J2EE server
- At startup of a J2EE application
- At termination of a J2EE application
- At termination of a J2EE server

The following describes the processing performed at the above times.

(1) Processing performed at startup of a J2EE server

To start a J2EE application that is customized to use CTM, when the J2EE server is started, you must establish and initialize a connection to the CTM daemon as follows:

1. Specify the settings for using CTM.
2. Start the CTM daemon.
3. Start the J2EE server.

When the J2EE server starts, it establishes and initializes a connection to the CTM daemon. Make sure that the CTM daemon is started before you start the J2EE server.

For details about the settings for using the CTM daemon, see *3.4.2 Settings in the execution environment*. For details about how to start the CTM daemon and J2EE server, see *4.1.24 Starting the system (when using CUI)* in the *uCosminexus Application Server System Setup and Operation Guide*. If Smart Composer is used to start the system, the CTM daemon is started, and then the J2EE server is started.

If establishment and initialization of a connection to the CTM daemon fails during startup of the J2EE server, startup of the J2EE server fails. In this case, correct the cause of the failure, and then restart the J2EE server.

(2) Processing performed at startup of a J2EE application

When a J2EE application is started, the J2EE server requests the CTM daemon to activate a schedule queue with the specified queue name. In response to the request, the CTM daemon activates the queue, and then executes `create` on the J2EE server for any business-processing programs that the CTM daemon can process. The CTM daemon executes as many `create` instances as the number of concurrent threads (`Parallel Count`) for each business-processing program that is directly called by the CTM daemon.

Each time the EJB object reference that corresponds to a business-processing program is created, the EJB object reference is returned to the CTM daemon. The CTM daemon pools the received EJB object references, and then assigns them to requests that are input to the schedule queue. Thus requests are distributed to business-processing programs via the EJB object references.

(3) Processing performed at termination of a J2EE application

When a J2EE application is terminated, first, the CTM daemon is requested to lock (de-activate) the schedule queue managed by the CTM daemon to prevent the CTM daemon from distributing new requests. After de-activating the schedule queue, the CTM daemon executes `remove` on the J2EE server for any business-processing programs that the CTM daemon can process. The CTM daemon executes as many `remove` instances as the number of concurrent threads (`Parallel Count`) for each business-processing program that is directly called by the CTM daemon.

After that, J2EE application termination processing is executed in the same way as when CTM is not used.

(4) Processing performed at termination of a J2EE server

When the J2EE server is terminated, the connection between the J2EE server and CTM daemon is closed.

3.2.5 Basis on which to create schedule queues and sharing schedule queues

Queues can be created on a J2EE application basis or on a bean basis. This subsection describes the configuration of schedule queues and sharing of schedule queues. This subsection also describes the advantages of sharing queues and not sharing queues.

(1) Basis on which to create schedule queues

Execution of each request from clients is scheduled by using schedule queues managed by the CTM daemon. Schedule queues can be created on a J2EE application basis or on a bean basis. If schedule queues are created on a J2EE application basis, J2EE application names are used as the default queue names. If schedule queues are created on a bean basis, bean names are used as the default queue names.

(2) Sharing schedule queues

Business-processing programs or J2EE application beans that have different interfaces can share a schedule queue that is created on a J2EE application or bean basis. The requests controlled by using schedule queues are managed by using a combination of the EJB home reference name (registered in the global CORBA Naming Service) and the remote interface name (of the business-processing program).

For J2EE applications or beans to share a schedule queue, they must be associated with the same CTM daemon and must satisfy the following conditions.

For J2EE applications to share a schedule queue:

- The queue names must be the same.
- The J2EE applications must consist of the same business-processing programs. (The J2EE applications must contain exactly the same enterprise beans to the extent that CTM recognizes.)

For beans to share a schedule queue:

- The queue names must be the same.
- The beans must be the same.

J2EE applications for which different queue names are specified cannot share a schedule queue even if the J2EE applications consist of the same business-processing programs. Similarly, J2EE applications consisting of different business-processing programs cannot share a schedule queue even if the queue names specified for the J2EE applications are the same.

A schedule queue can be shared across J2EE servers. To share a schedule queue across J2EE servers, use the user-specified namespace functionality to assign an alias (optional name) to each enterprise bean (business-processing program). For details about this functionality, see *2.3 Binding and looking up objects in the JNDI name* in the *uCosminexus Application Server Common Container Functionality Guide*. Make sure that you assign an optional name as a J2EE application property.

Reference note

- If J2EE applications are imported with the default settings, the lookup names for business-processing programs are assigned in the following format: `/HITACHI_EJB/SERVERS/`*J2EE-server-name*`/EJB/`*J2EE-application-name*`/`*business-processing-program-name*. Because a specific J2EE server name is included in this format, J2EE applications having lookup names in this format cannot share a schedule queue across J2EE servers.
- It is impossible to share a schedule queue by importing multiple J2EE applications with the same name on one J2EE server.

(3) Advantages of sharing schedule queues

This subsection describes the advantages of sharing schedule queues on a J2EE application basis and on a bean basis, separately.

(a) Sharing on a J2EE application basis

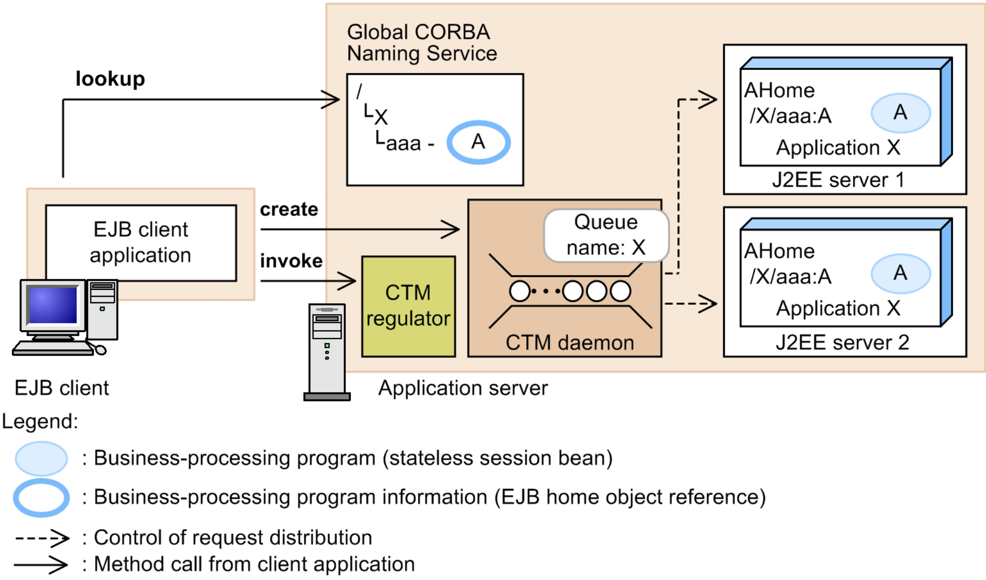
If a schedule queue is shared by J2EE applications, requests can be distributed to J2EE applications on multiple J2EE servers.

Advantages of sharing schedule queues are as follows:

- The number of threads running concurrently can be controlled between the J2EE applications that share a queue. Therefore, degradation of performance can be prevented when a specific J2EE application is heavily loaded. This improves the stability of system processing.
- If a J2EE server that shares a queue fails, the system can operate in reduced mode to process requests in the queue with the J2EE applications on other normally operating J2EE servers. This prevents business processing from stopping.

The following figure shows an example of sharing a schedule queue.

Figure 3–1: Example of sharing schedule queues (by J2EE applications)



The EJB client executes `lookup` for the global CORBA Naming Service. If a schedule queue is shared, the EJB client can obtain a reference to the queue (in this example, a reference to queue `X` can be obtained). When the EJB client executes `create` for that queue, the EJB client obtains a reference to the CTM regulator. When the EJB client executes `invoke` for that reference, schedule queue `X` distributes processing to J2EE server 1 or J2EE server 2.

(b) Sharing on a bean basis

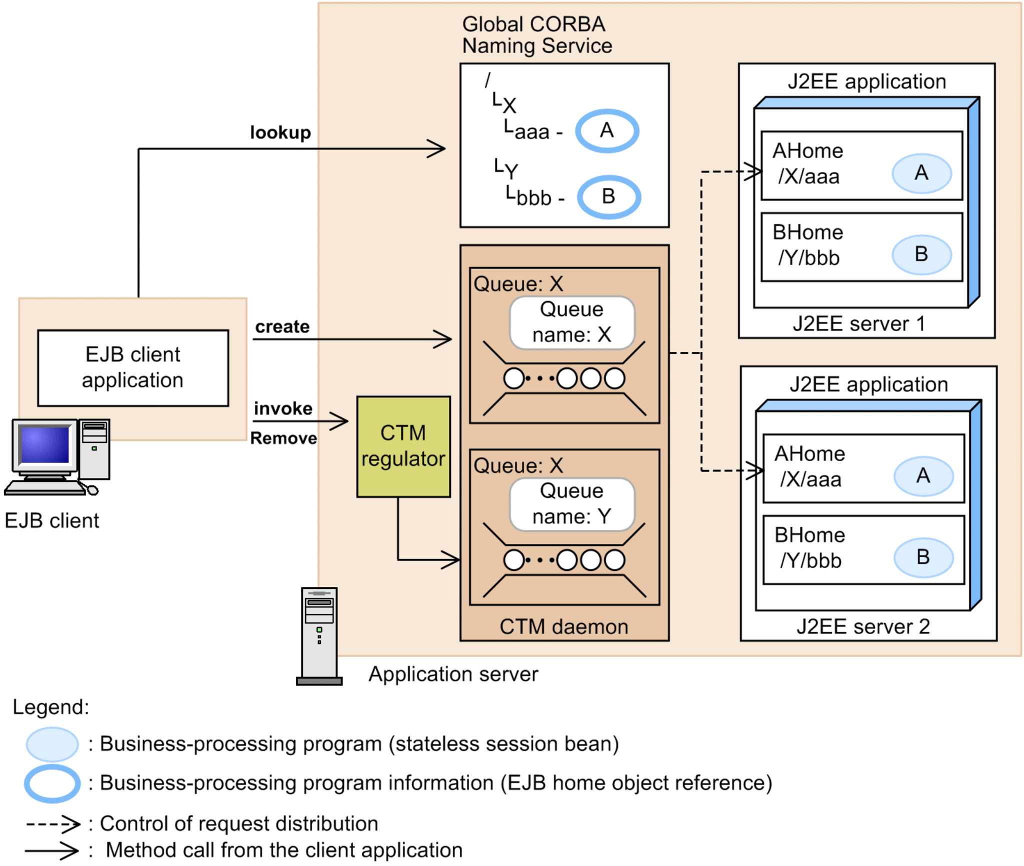
If a schedule queue is shared by beans, requests can be distributed to beans on multiple J2EE servers.

Advantages of sharing schedule queues are as follows:

- Queues can be assigned to specific types of beans, so as not to affect other beans in the same J2EE application.
- The number of threads running concurrently can be controlled between the beans that share a queue. Therefore, degradation of performance can be prevented when a specific bean is heavily loaded. This improves stability of system processing.
- If a J2EE server that shares a queue fails, the system can operate in reduced mode to process the requests in the queue with the beans on other normally operating J2EE servers. This prevents business processing from stopping.

The following figure shows an example of sharing a schedule queue.

Figure 3–2: Example of sharing schedule queues (by beans)



The EJB client executes `lookup` for the global CORBA Naming Service. If a schedule queue is shared, the EJB client can obtain a reference to the queue (in this example, a reference to queue `X` can be obtained). When the EJB client executes `create` for that queue, the EJB client obtains a reference to the CTM regulator. When the EJB client executes `invoke` for that reference, schedule queue `X` distributes processing to bean `A` on either J2EE server 1 or J2EE server 2.

(4) Advantages of not sharing schedule queues

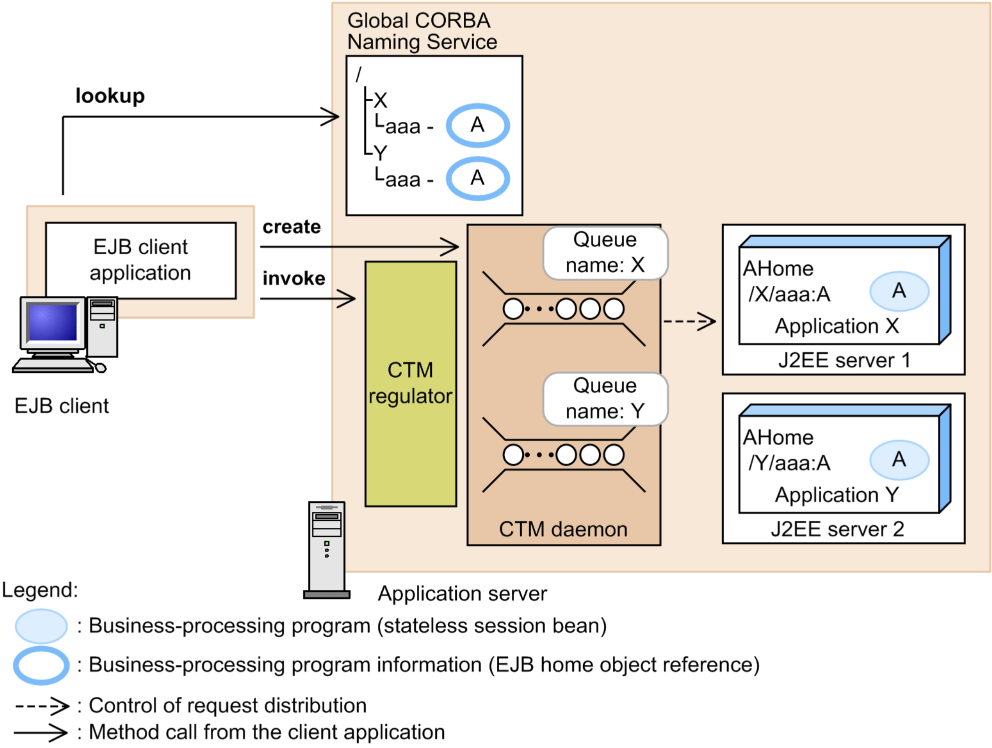
If no schedule queues are shared when the same J2EE applications have been imported on different J2EE servers or when the same beans exist on different J2EE servers, each queue individually controls requests to be executed on a certain server.

No sharing of schedule queues makes it impossible to use load balancing or reduced mode. However, accumulation of requests in a schedule queue does not affect execution of requests in other schedule queues. Therefore, requests in each queue are processed smoothly.

For schedule queues not to be shared, specify different lookup names for the business-processing programs in each J2EE application rather than specifying optional names.

The following figure shows an example of not sharing schedule queues.

Figure 3–3: Example of not sharing schedule queues



The EJB client executes `lookup` for the global CORBA Naming Service. If no schedule queues are shared, the EJB client can obtain a reference to the queue that controls the specified J2EE application (in this example, a reference to queue `X` can be obtained). When the EJB client executes `create` for that queue, the EJB client obtains a reference to the CTM regulator. When the EJB client executes `invoke` for that reference, processing is distributed to J2EE server 1 that is controlled by schedule queue `X`.

3.2.6 Length of a schedule queue

The length of a schedule queue can be set on the following bases:

- On a CTM daemon basis
- On a J2EE application basis
- On a session bean basis

For details about setting the schedule queue length on a CTM daemon basis, see *3.3.3(2) Registering requests in a schedule queue*.

To set the schedule queue length on a J2EE application basis or on a session bean basis, use the `<queue-length>` element in the `<scheduling>` element. For details about the request scheduling settings in CTM, see *3.4.2(3) Using server management commands to specify the settings*.

Note that because a schedule queue to be shared has already been created, specification of the length for that schedule queue does not take effect.

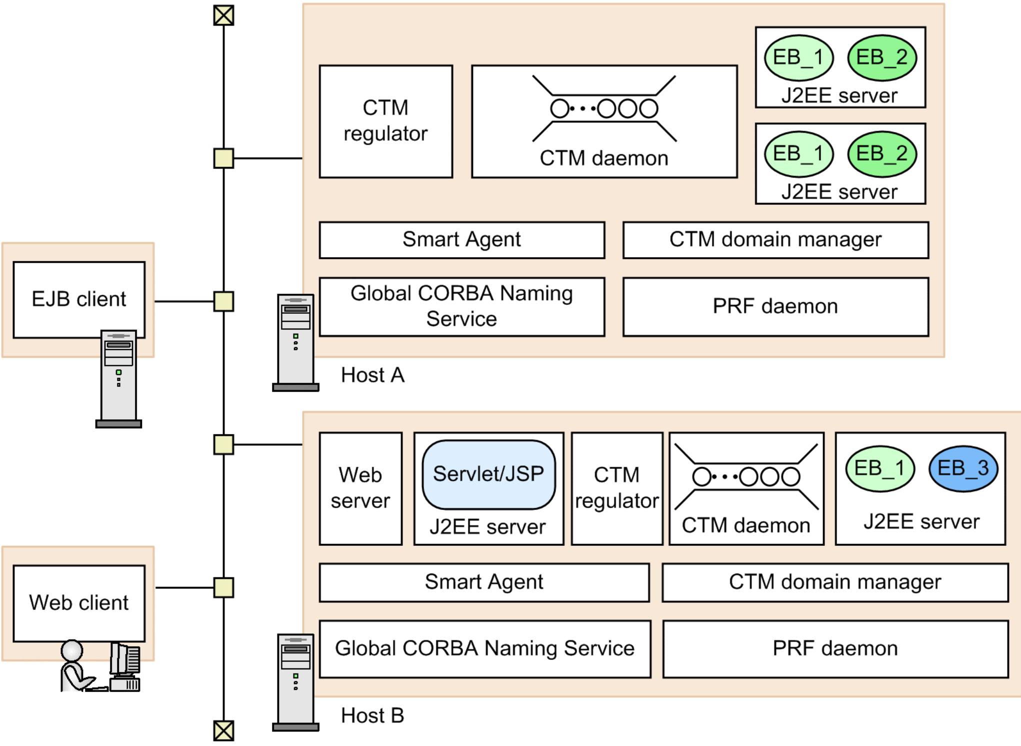
3.3 Process configuration for using CTM

This section describes the process configuration of an environment that uses CTM to schedule requests, and provides guidelines for deploying processes. This section also describes the function of each process.

3.3.1 Configuration and deployment of CTM processes

The following figure shows an example of deploying processes for using CTM.

Figure 3–4: Example of processes that make up CTM



The following table describes the main functionality of each process.

Table 3–2: Processes necessary for using CTM

Process	Description
CTM daemon	A process that manages a schedule queue that controls requests from clients
CTM regulator	A process that distributes and consolidates requests that concentrate on a CTM daemon
CTM domain manager	A process that manages a CTM domain. A CTM domain is made up of multiple CTM daemons, and is a range in which information can be shared and load can be balanced.
Global CORBA Naming Service	A naming service that manages the information about the business-processing programs on the hosts in the same CTM domain so that the information can be shared
PRF daemon (performance tracer)	An I/O process that receives performance analysis information from a CTM daemon and then outputs the information to a file. For details about the PRF daemon, see *7.5 Settings of execution environment* in the *uCosminexus Application Server Maintenance and Migration Guide*.

Process	Description
Smart Agent	A dynamic distributed directory service provided by TPBroker. CTM requires Smart Agent when scheduling requests. CTM also uses Smart Agent to distribute information to the CTM daemon in a different network segment.

3.3.2 Guidelines for deploying processes

This subsection provides the guidelines for deploying processes:

- Deploy one CTM daemon on one host.
- All hosts on which a J2EE server or CTM regulator is deployed require a CTM daemon.
- One CTM daemon can control multiple J2EE servers.
- Multiple CTM regulators can be deployed per CTM daemon. Note, however, that if 256 or more requests are simultaneously sent to one CTM regulator, performance might be degraded. In this case, deploy more CTM regulators.
- No CTM daemon is required on client hosts on which the EJB client is operating.
- Deploy one CTM domain manager on a host on which a CTM daemon is deployed. If you want multiple CTM daemons to participate in the same CTM domain, specify the same CTM domain manager name for the relevant hosts.
- Deploy a CTM daemon on the host to be used as the integrated naming scheduler server (although a J2EE server is not deployed on this host). You do not need to deploy a CTM regulator on this host. For details about the integrated naming scheduler server, see *(4) Configuration in which an independent integrated naming scheduler server is set up (integrated naming scheduler server model)*.

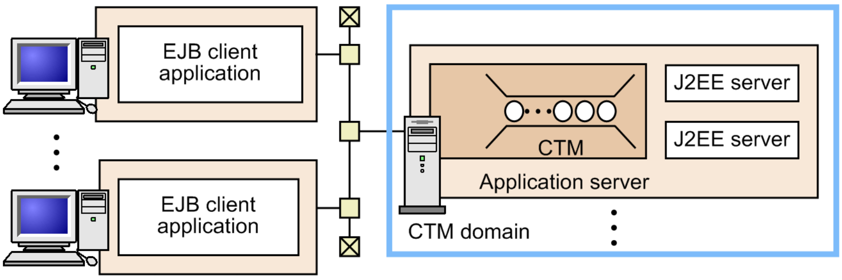
For details about how to start each process, see *Chapter 2. Starting and Stopping the System* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

The following subsections describe the deployment patterns of processes used for CTM.

(1) Configuration in which many EJB clients call J2EE servers

The following figure shows an example of the configuration in which many EJB clients call J2EE servers on application servers that are deployed in parallel.

Figure 3–5: Example of the configuration in which many EJB clients call J2EE servers

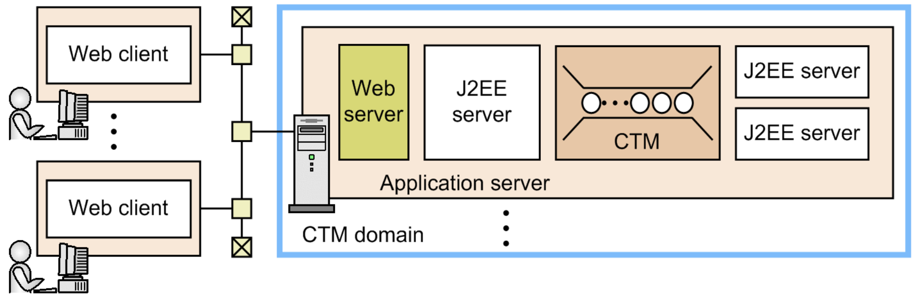


Note: *CTM* in this figure includes the CTM daemon, CTM regulator, Smart Agent, Global CORBA Naming Service, CTM domain manager, and PRF daemon.

(2) Configuration in which web browsers call J2EE servers (small-scale configuration)

The following figure shows an example of the configuration in which web browsers call J2EE servers via web containers on web servers or application servers that are deployed in parallel. In this configuration, a web server and application server are installed on the same host.

Figure 3–6: Example of the configuration in which web browsers call J2EE servers (small-scale configuration)

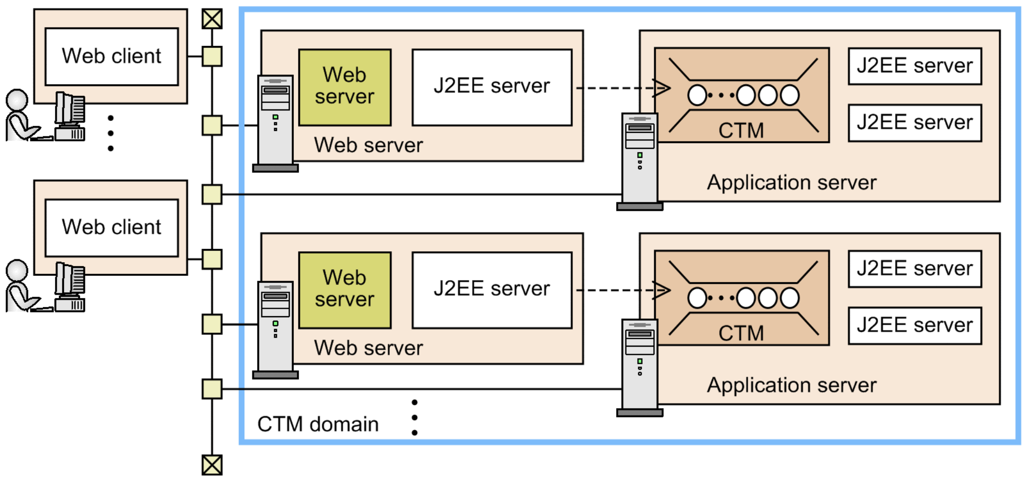


Note: *CTM* in this figure includes the CTM daemon, CTM regulator, Smart Agent, Global CORBA Naming Service, CTM domain manager, and PRF daemon.

(3) Configuration in which web browsers call J2EE servers (large-scale configuration)

The following figure shows an example of the configuration in which web browsers call J2EE servers on application servers via web containers on web servers. In this configuration, a web server and application server are installed on separate hosts. Therefore, web servers and application servers can be easily combined in a many-to-many relationship.

Figure 3–7: Example of the configuration in which web browsers call J2EE servers (large-scale configuration)



Note: *CTM* in this figure includes the CTM daemon, CTM regulator, Smart Agent, Global CORBA Naming Service, CTM domain manager, and PRF daemon.

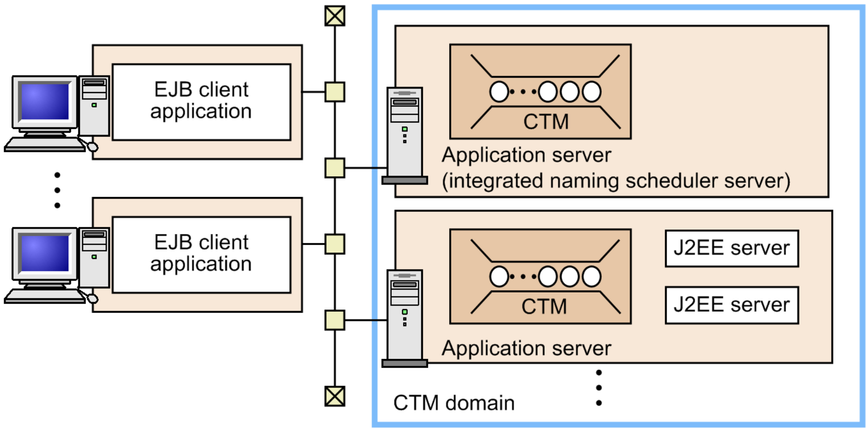
(4) Configuration in which an independent integrated naming scheduler server is set up (integrated naming scheduler server model)

In this configuration, the global CORBA Naming Service is deployed on a separate host. If replicas of the host are created, availability of the naming service can be improved. A host on which the global CORBA Naming Service is deployed is called an *integrated naming scheduler server*. No J2EE servers need to be installed on integrated naming scheduler servers.

However, to register (in the global CORBA Naming Service on an integrated naming scheduler server) information about business-processing programs on other hosts, a CTM daemon must also be deployed on the integrated naming scheduler server.

The following figure shows an example of the configuration in which an independent integrated naming scheduler server is set up.

Figure 3–8: Example of the configuration in which an independent integrated naming scheduler server is set up (integrated naming scheduler server model)



Note: *CTM* in this figure includes the CTM daemon, CTM regulator, Smart Agent, Global CORBA Naming Service, CTM domain manager, and PRF daemon.

Note that in this configuration, requests other than `create` are not sent to CTM daemons on integrated naming scheduler servers. Therefore, the CTM regulator does not need to be activated on integrated naming scheduler servers.

3.3.3 CTM daemon

A CTM daemon is a process that has scheduler functionality. It manages and schedules requests from clients.

Important note

To start a CTM daemon as a Windows service, specify `-Dvbroker.orb.isNTService=true` as a startup command option.

CTM daemons receive requests from clients via processes called *CTM regulators*. For details about the CTM regulator, see *3.3.4 CTM regulator*.

Note that the functionality of CTM daemons is configured by specifying arguments for the `ctmstart` command executed at startup of CTM daemons. In a system set up by using the management portal, configuration can be completed by using logical CTM beforehand.

A CTM daemon manages requests in the following sequence:

1. Distributing requests
2. Registering requests in a schedule queue

3. Calling business-processing programs

4. Returning results

The above steps are described below.

(1) Distributing requests

When a CTM daemon receives a request, it manages the request by itself or distributes the request to another CTM daemon, based on the load status of the CTM daemons.

The CTM daemons exchange their own load information with each other. When a CTM daemon receives a request, the CTM daemon determines which CTM daemon will manage the request based on the shared load information.

The CTM daemons in a certain range of area (called a *CTM domain*) share information about the business-processing programs contained in J2EE applications that the CTM daemons manage. The shared information is registered in the global CORBA Naming Service on the hosts on which the CTM daemons exist. If a CTM daemon receives a request to execute a business-processing program that the CTM daemon does not manage, the shared information allows the CTM daemon to distribute the request to the appropriate CTM daemon.

For details about the global CORBA Naming Service, see *3.3.6 Global CORBA Naming Service*. For details about CTM domains, see *3.3.5 CTM domains and CTM domain managers*.

CTM daemons distribute requests based on the `create`-based selection policy or schedule policy.

Both the `create`-based selection policy and schedule policy allow you to select which of the following types of distribution is used during startup of CTM daemons:

- A received request is distributed to the least heavily loaded CTM daemon.
- A received request is distributed to the CTM daemon that received the request.

  Note that if the CTM daemon that received a request is in a high load state or blocked state, another CTM daemon manages the request. The threshold that judges a high load state is calculated from the percentage of the used capacity of queues.

For when the `create`-based selection policy or schedule policy is applied, see *3.8 Load balancing of requests*.

The schedule policy is specified by using the `-CTMDispatchPolicy` argument of the `ctmstart` command. The `create`-based selection policy is specified by using the `-CTMCreatePolicy` argument of the `ctmstart` command.

Request transfer timeout

A timeout can be set for the request transfer processing between CTM daemons. The timeout can be specified by using the `-CTMDCSendTimeOut` option of the `ctmstart` command.

(2) Registering requests in a schedule queue

Requests distributed based on the schedule policy are registered in the schedule queue. The maximum number of requests that can be registered in the schedule queue is set at startup of CTM daemons. If the maximum number of transferable requests is exceeded, an error is returned to the client. If a maximum number of requests (queue length) is not set, 50 is set by default.

The length of the queue in which requests can be registered is specified by using the `-CTMMaxRequestCount` argument of the `ctmstart` command at startup of CTM daemons. In a system set up by using the management portal, the queue length can be set beforehand by using logical CTM. For details about the `ctmstart` command, see *ctmstart (start CTM daemon)* in the *uCosminexus Application Server Command Reference Guide*.

(3) Calling business-processing programs

Requests registered in the schedule queue call business-processing programs on J2EE servers managed by CTM daemons. These requests do not call business-processing programs on abnormally-terminated J2EE servers or hung business-processing programs.

(4) Returning results

After requests are processed, replies from business-processing programs (enterprise beans) are returned to the clients via CTM daemons. If the time period during which a request is in the schedule queue exceeds the request timeout, the request is discarded.

3.3.4 CTM regulator

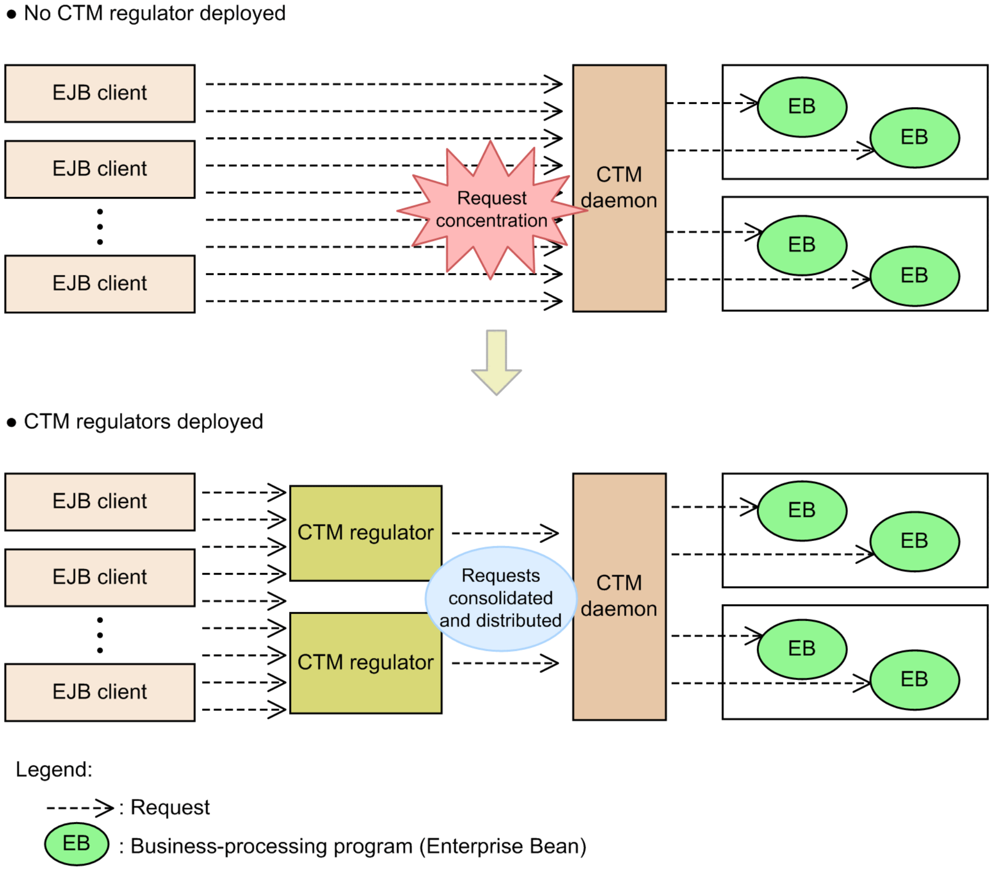
A CTM regulator is a process that solves a problem caused by request concentration on CTM daemons by regulating (consolidating) connections or requests. A CTM regulator is deployed at the front end of a CTM daemon, and distributes and consolidates connections or requests (`invoke` or `remove` requests) from EJB clients.

For example, in a large-scale system, if many clients issue many requests, the system might not operate stably or system-managed resources might become insufficient, preventing the system from operating normally. These phenomena are due to request concentration on CTM daemons that schedule requests. Request concentration increases the number of connections, causing processes to use a larger number of resources, such as opened files and sockets.

A CTM regulator is a special process that solves problems due to request concentration. CTM regulators consolidate connections from clients into one to control the number of connections established per CTM daemon. This control is called *connection regulation*. CTM regulators distribute resources to processes by regulating a large number of connections, so that a large-scale system can operate more stably.

The following figure shows how connections are regulated.

Figure 3–9: How connections are regulated



When a CTM regulator receives a request from an EJB client, the CTM regulator transfers the request to the corresponding CTM daemon, and then waits for a reply. Upon receiving a reply, the CTM regulator returns the reply to the EJB client.

In CTM, multiple CTM regulators can be deployed per CTM daemon, as required. If a CTM regulator receives 256 or more requests simultaneously, performance might be degraded. In such a case, increase the number of CTM regulators, regardless of the number of client processes. Note that CTM regulators and the corresponding CTM daemon must be deployed on the same host.

For the integrated naming scheduler server model in which a naming service and a J2EE server are deployed on separate hosts, the integrated naming scheduler server does not accept any requests other than `create`. Therefore, CTM regulators do not need to be activated on the integrated naming scheduler server.

3.3.5 CTM domains and CTM domain managers

A *CTM domain* is a range of area in which CTM daemons exchange information about registered business-processing programs and schedule queue load status with each other to share information and perform load balancing. Each CTM domain is identified by a CTM domain name. Requests are distributed and scheduled among the CTM daemons in the same CTM domain. The range of each CTM domain and the information about the CTM daemons in each CTM domain are managed by the CTM domain manager.

Tip

CTM domains are included in the management domain managed by Management Server.

Important note

Adding CTM domains increases information in the file system. For CTM domains that are no longer used, use the `ctmdminfo` command to delete the CTM domain information.

A *CTM domain manager* is a daemon process that manages the information about the CTM daemons that exist in the same CTM domain. A CTM domain manager is required on each host on which CTM daemons are deployed.

The way a CTM domain manager distributes information to other CTM domain managers differs depending on whether the other CTM domain managers are in other network segments.

Note that the functionality of CTM domain managers is configured by specifying arguments for the `ctmdmstart` command executed at startup of CTM daemon managers. In a system set up by using the management portal, configuration can be completed by using logical CTM beforehand. For details about the command, see *ctmdmstart (start CTM domain manager)* in the *uCosminexus Application Server Command Reference Guide*.

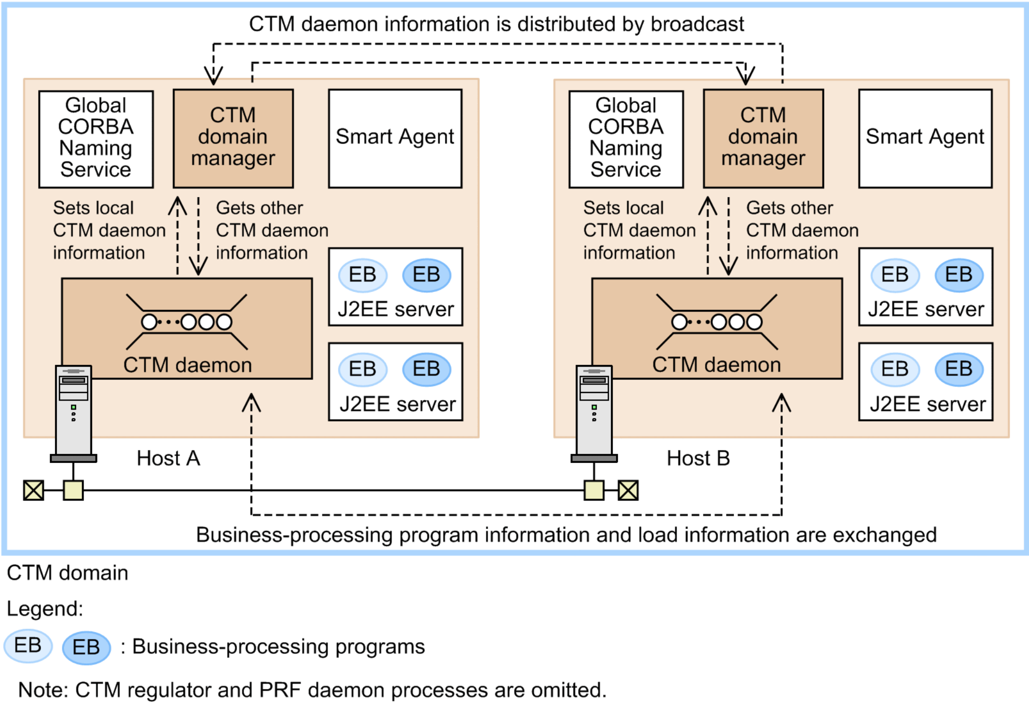
Important note

- To start a CTM domain manager as a Windows service, specify `-Dvbroker.orb.isNTService=true` as a startup command option.
- If a CTM daemon terminates abnormally in Windows, the CTM domain manager forcibly terminates the child processes of the CTM daemon.
- If a CTM domain manager terminates abnormally, execute the CTM domain manager normal startup command (`ctmdmstart`) with the `-CTMForceStart` or `-CTMAutoForce` option.

(1) Sharing information with CTM domain managers in the same network segment

A CTM domain manager distributes information about the CTM daemons that exist on the host to the CTM domain managers on other hosts by broadcast. The following figure shows how CTM domain managers in the same network segment share information.

Figure 3–10: Sharing information with CTM domain managers in the same network segment



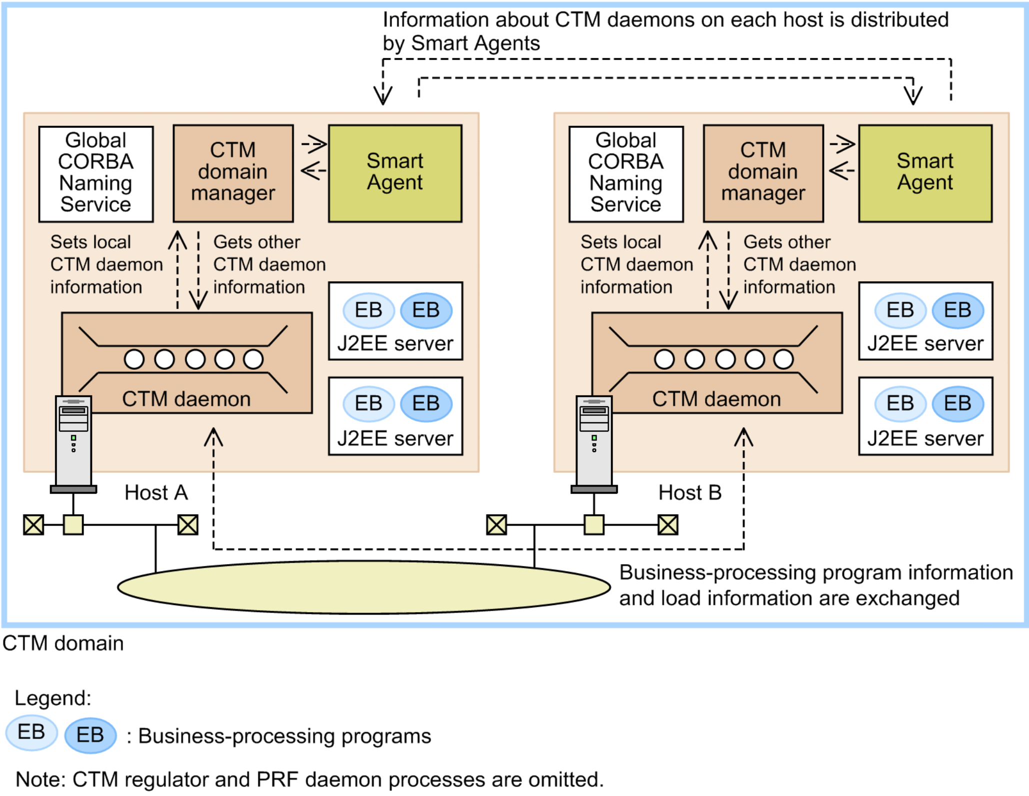
To add a new CTM daemon to an existing CTM domain, on a host in that CTM domain, start a CTM domain manager that has the same domain name and port number as other CTM domain managers. A new CTM daemon will then participate in the domain. You do not need to update the environment definitions and other information in the existing CTM domain. Therefore, you can easily scale out the system by simply copying the system environment.

(2) Sharing information with CTM domain managers in different network segments

Broadcast cannot send information beyond routers, and, therefore, cannot be used to share information between CTM domain managers in different network segments. For these CTM domain managers to share information, the information must be distributed by using Smart Agent.

The following figure shows how CTM domain managers in different network segments share information.

Figure 3–11: Sharing information with CTM domain managers in different network segments



The following shows the settings that are necessary to create a CTM domain with multiple network segments:

- When starting a CTM domain manager, specify the host name or IP address of the CTM domain manager that information is to be shared with.

  You can specify the queue in which requests can be registered by using the `-CTMSendHost` argument of the `ctmdmstart` command at startup of CTM domain managers. In a system set up by using the management portal, the queue can be set beforehand by using logical CTM.
- Connect the Smart Agent in the local network segment to the Smart Agent in the other network segment.

(3) Restarting only the CTM domain manager

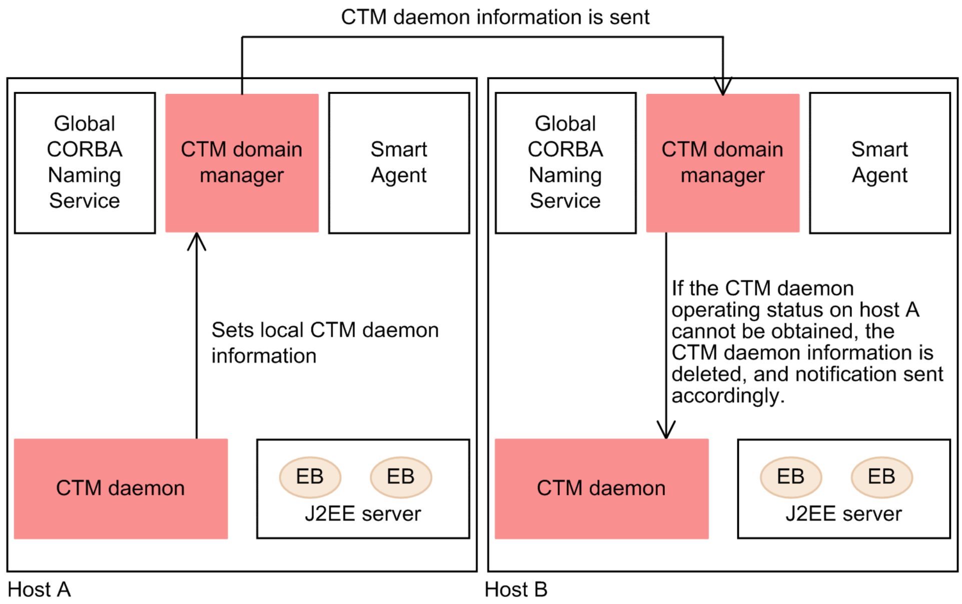
If a CTM domain manager terminates abnormally, you might be able to restart only the CTM domain manager when restart is attempted. Whether restart is possible is automatically determined. If impossible, the entire system terminates abnormally, and you must restart the entire system.

(4) Checking the operating status of CTM domain managers

A CTM domain manager checks whether the CTM domain managers on other hosts are running. The interval time at which a CTM domain manager performs this check can be changed. To change the check interval time, use the `-CTMAliveCheckCount` option of the `ctmdmstart` command.

If a CTM domain manager performs an operating status check and does not receive CTM node information, it judges that the CTM domain managers on those nodes are not running. The CTM domain manager then deletes the CTM information about those nodes. No requests will be distributed to the CTM daemons on those nodes. The following figure shows how a CTM domain manager checks the operating status of other CTM domain managers.

Figure 3–12: Checking the operating status of CTM domain managers



The host-B CTM domain manager waits for the information about the host-A CTM daemon from the host-A CTM domain manager. The host-B CTM domain manager waits for response for the following time: CTM daemon information transmission interval × dead-or-alive decision monitoring coefficient. If the wait times out, the host-B CTM domain manager deletes the information about the host-A CTM daemon, and notifies the host-B CTM daemon that the information about the host-A CTM daemon was deleted. As a result, the host-B CTM daemon will not distribute any requests to the host-A CTM daemon.

3.3.6 Global CORBA Naming Service

CTM-based request scheduling uses the global CORBA Naming Service as the naming service.

The *global CORBA Naming Service* is a naming service that manages information about the business-processing programs (stateless session beans) contained in the same CTM domain so that the information can be shared. The global CORBA Naming Service allows the hosts in the CTM domain to share the information about the EJB home object references registered on those hosts. The global CORBA Naming Service can be used to find J2EE servers on which a requested business-processing program is registered if that program is not registered on the J2EE server of the CTM daemon that received the request. With the global CORBA Naming Service, requests can be distributed to appropriate CTM daemons in this way.

A global CORBA Naming Service is deployed for each CTM daemon. CTM daemons exchange information with each other, including information about the business-processing programs on other hosts. Each CTM daemon registers this information in the global CORBA Naming Service on the local host. The information of the global CORBA Naming Services is thus shared within a CTM domain. Therefore, to obtain information about J2EE servers on other hosts, deploy a CTM daemon that runs only a global CORBA Naming Service (without running the J2EE server) on the integrated naming scheduler server.

The characteristics of a global CORBA Naming Service are as follows:

- System availability can be improved by minimizing the area affected by a failure.

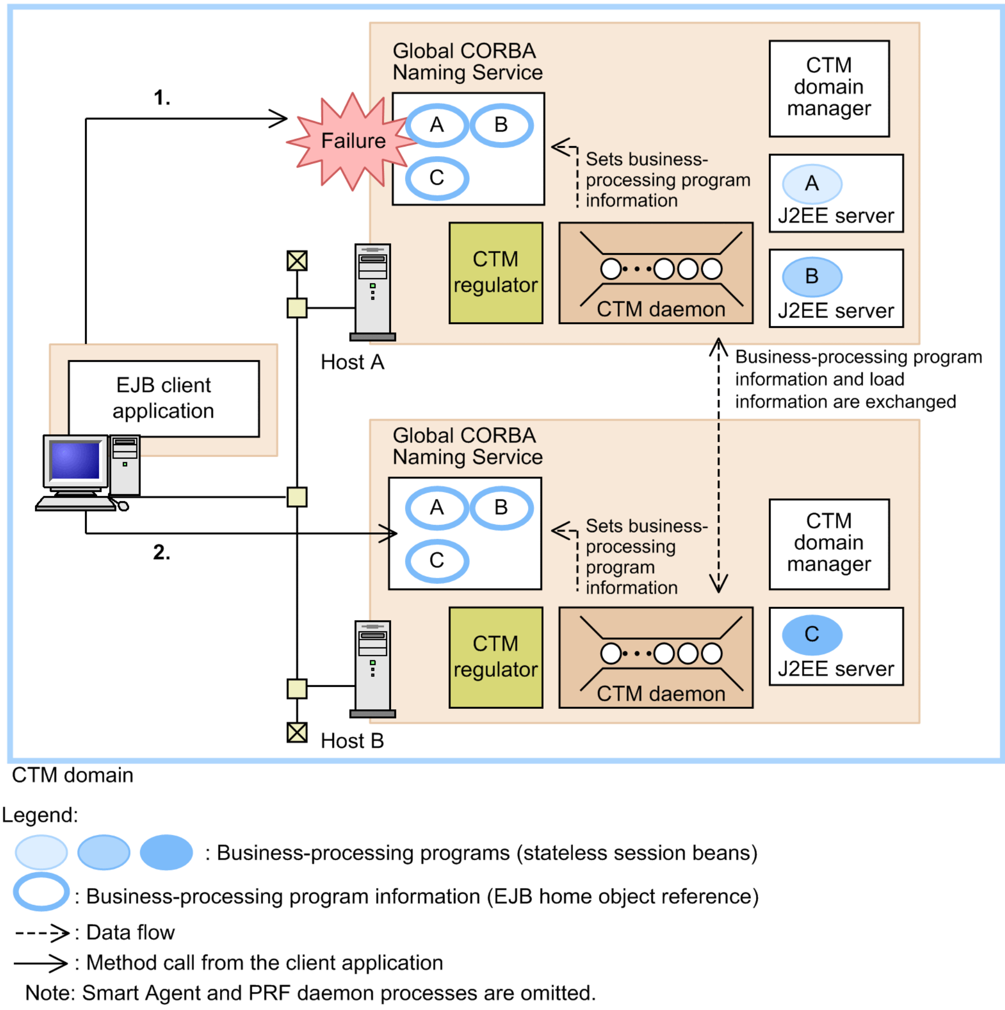
  Information can be shared within a domain by deploying a global CORBA Naming Service for each CTM daemon. Therefore, if a problem occurs in the global CORBA Naming Service on a host, operation can continue with the global CORBA Naming Service on another host. As a result, system availability can be improved.

- The lookup-target naming service does not need to be selected for each business-processing program.
  If load balancing is implemented by clustering, the same information about business-processing programs (EJB home object reference information) is registered in all global CORBA Naming Services in the CTM domain. Therefore, the lookup-target naming service does not need to be selected for each business-processing program to be executed. This prevents the load from concentrating on a specific naming service, thus enabling proper load balancing.

The below figure is an example of processing in a system that uses global CORBA Naming Services.

In this example, the CTM daemons on hosts A and B are registered in the same CTM domain. Business-processing programs A and B are registered in the J2EE server on host A. Business-processing program C is registered in the J2EE server on host B. Note that a failure has occurred on host A. Also note that the EJB client application was started by specifying a system property (`java.naming.factory.initial` key) that is set to perform a round-robin search.

Figure 3–13: Example of processing in a system that uses global CORBA Naming Services



The following describes the processing in the above figure:

1. For the EJB client application to start business-processing program C, first, an EJB home object reference to that program must be looked up from a global CORBA Naming Service. In this figure, the EJB client application executes `lookup` for the global CORBA Naming Service on host A, but an exception is thrown for the `lookup` because of a failure on host A.
2. If a global CORBA Naming Service fails when round-robin search is enabled with the EJB client application's system property, the application automatically switches the lookup destination to another global CORBA Naming

Service in the CTM domain. In this example, the EJB client application re-executes `lookup`, and obtains an EJB home object reference to business-processing program C from the global CORBA Naming Service on host B. Business-processing program C, which is installed on application server B, can then be executed regardless of the failure on application server A.

If no failure occurs on application server A, the global CORBA Naming Service on host A returns a reference in response to `lookup` in step 1. When the EJB client application uses the reference to request `create`, the CTM daemons on hosts A and B determine which CTM daemon manages the request. As a result, an EJB home object reference to business-processing program C on host B is returned to the EJB client application.

Important note

- If a problem occurs on a host on which a global CORBA Naming Service is registered, restart the application server on the host so that the CTM daemon re-registers schedule queue references in the global CORBA Naming Service.
- While a request is being processed, `CORBA::`*XXXX* exceptions might be sent to the standard output or standard error output. These exceptions are harmless if processing continues without changing the status.

3.4 Flow-volume control of requests

The flow-volume control function of CTM limits the number of requests that can be concurrently executed on each J2EE server to moderate the load on the J2EE servers. As a result, stable and high throughput is achieved.

The following table shows the structure of this section.

Table 3–3: Structure of this section (flow-volume control of requests)

Topic type	Title	Location
Description	Overview of flow-volume control of requests	*3.4.1*
Settings	Settings in the execution environment	*3.4.2*

Note: This section does not provide *Implementation*, *Operation*, and *Notes* types of topics that are specific to this function.

3.4.1 Overview of flow-volume control of requests

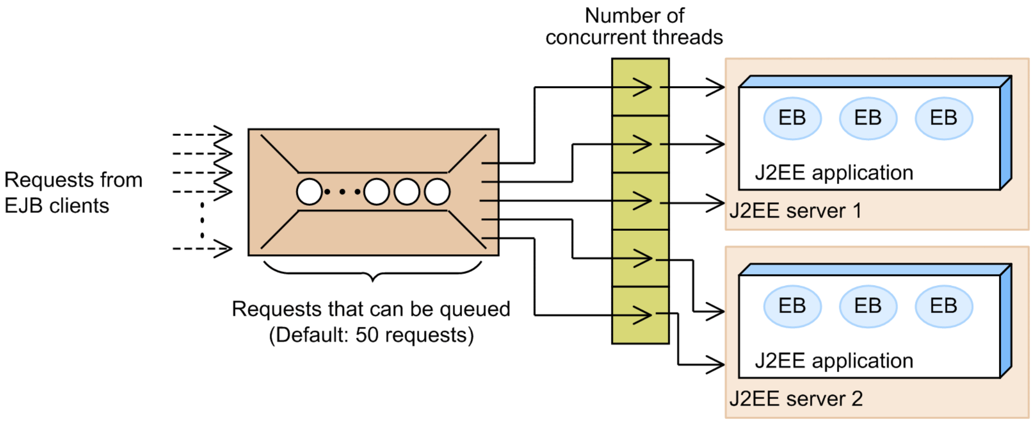
This subsection describes flow-volume control of CTM.

To control the number of requests that are executed concurrently, the flow-volume control function places a preset limit on the number of threads that can be concurrently generated on each J2EE server. This function moderates the load on the J2EE servers, thus providing stable and high throughput. This function can also prevent contention of a CPU or exclusive resource.

Flow-volume control of CTM is enabled by using a CTM daemon and the schedule queue managed by the CTM daemon.

The following figure shows an overview of flow-volume control of CTM, and provides an example of schedule queue sharing on a J2EE application basis.

Figure 3–14: Overview of flow-volume control of CTM



A CTM daemon adds received requests to a schedule queue, and executes as many requests as the maximum number of concurrent threads set for the schedule queue. The CTM daemon repeats this processing. If the number of requests from clients instantaneously increases, the CTM daemon controls the flow volume so that the number of requests executed by each J2EE server does not exceed the maximum number of concurrent threads. If the same J2EE application is installed on multiple J2EE servers and the schedule queue is shared by each instance of that J2EE application, the requested business-processing program can be executed in parallel. The degree of parallelism can be determined by the number of instances of that J2EE application and the maximum number of concurrent threads for each J2EE application. You can set the maximum number of requests that can be registered in a schedule queue. Note that if the maximum number of requests that can be registered is not set for a schedule queue, the setting for the CTM daemon is used by default. If the maximum is exceeded, an error is returned.

Note that the EJB container can also control the degree of parallelism for a 2EE application. The following describes the effect of combining the parallelism control of the EJB container and the flow-volume control of CTM.

- If the degree of parallelism reaches the upper limit for the EJB container on a J2EE server, requests can be transferred to another J2EE server. Even when the upper limit is not reached, if the load on the EJB container is high, requests can be transferred to other J2EE servers.
- Message queuing by CTM might limit the number of queued requests to the preset maximum. Therefore, if more requests than the maximum are sent to the EJB container, an error can be reported.
- Instance pooling of the EJB container can also be used simultaneously.

The maximum number of threads that can be controlled by CTM is specified by the `-CTMDispatchParallelCount` argument of the `ctmstart` command executed when the CTM daemon is started. The maximum number of requests that can be registered in a queue is specified by the `-CTMMaxRequestCount` argument of that command. If the system was set up by using the management portal, you can set these values in logical CTM beforehand. For details about the `ctmstart` command, see *ctmstart (start CTM daemon)* in the *uCosminexus Application Server Command Reference Guide*.

Important note

- In flow-volume control of CTM, the maximum number of concurrent threads (`Parallel Count`) is set for each schedule queue. The maximum number of instances that can be pooled (maximum of `Pooled Instances`) can be set for each stateless session bean called by CTM. Note that if the maximum number of instances that can be pooled is less than the number of concurrent threads for the schedule queue, instances might be insufficient when a stateless session bean is called.
- If CTM is used, EJB object references are registered in the local CORBA Naming Service, in addition to the global CORBA Naming Service, on each host. In some application configurations, therefore, enterprise beans can be called by directly executing `lookup` for the local CORBA Naming Service without using CTM. Note, however, that in this case, the number of concurrent threads specified by CTM is not guaranteed. Therefore, do not operate in such a manner.

3.4.2 Settings in the execution environment

Before you can use CTM functions, you must set up the system configuration in which CTM can be used. For details about the system configuration and setup procedure, see the *uCosminexus Application Server System Design Guide* and the *uCosminexus Application Server System Setup and Operation Guide*.

The settings for using CTM functions to schedule requests can be specified in the Easy Setup definition file. In this file, specify the CTM identifiers of CTM daemons, the lengths of CTM queues, and other settings for parameters whose names begin with `ejbserver.ctm`.

To schedule requests by using CTM, you must perform the following operations:

- Create execution-environment directories and specifying environment variable settings
- Specify the settings by using the Easy Setup definition file
- Specify the settings by using server management commands

(1) Creating execution-environment directories and specifying environment variable settings

When you set up a system without using Management Server, to use CTM in the system, you must create the execution-environment directories for CTM and the performance tracer, and then specify them for environment variables.

For details about creating execution-environment directories and specifying environment variables, see *Appendix H. System Environment Variables* in the *uCosminexus Application Server Command Reference Guide*.

Note that if you use Management Server to set up a system, you do not need to set the environment variables for using CTM.

Important note

In AIX, note the following points when setting environment variables:

- In the execution environment for Component Transaction Monitor, set `early` for the `PSALLOC` environment variable. If you do not set `early`, correct operation cannot be guaranteed when memory becomes insufficient.
- The `PSALLOC` environment variable, which specifies early paging space allocation mode in AIX, is set to `early`. In this mode, paging space estimation needs to be considered. For details, see *System Management Concepts: Operating System and Devices*, which is documentation for AIX.
- In the execution environment for Component Transaction Monitor, set `true` for the `NODISCLAIM` environment variable. If `PSALLOC` is `early`, `NODISCLAIM` must be `true`. If you do not set `true`, the response, throughput, and CPU usage rate might be severely degraded.
- To expand the user data area and shared memory area used by Component Transaction Monitor, set `MAXDATA=0x40000000` for the `LDR_CNTRL` environment variable. Then, specify 1 GB as the size of memory to be allocated.
- In the execution environment for Component Transaction Monitor, set `ON` for the `EXTSHM` environment variable. If you do not set `ON`, shared memory might not be accessible.

(2) Using the Easy Setup definition file to specify the settings

To schedule requests by using CTM, in the Easy Setup definition file, set the properties shown below in the `<configuration>` element for the logical J2EE server (`j2ee-server`). For details about the Easy Setup definition file, see *4.3 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

- `ejbserver.ctm.ActivateTimeOut`

  Specifies the time for which the J2EE server can wait before the schedule queue is activated at startup of the J2EE application that uses CTM.
- `ejbserver.ctm.DeactivateTimeOut`

  Specifies the time for which the J2EE server can wait before the schedule queue is de-activated (before requests being executed are completed) at termination of the J2EE application that uses CTM.
- `ejbserver.ctm.QueueLength`

  Specifies the length of the CTM queue generated by the J2EE server at startup of the J2EE application that uses CTM.
- `ejbserver.client.ctm.RequestPriority`

  Specifies the priority level of requests that the J2EE server sends to CTM.

(3) Using server management commands to specify the settings

This subsection describes the settings that can be specified by using server management commands. For details about operations that can be performed by using server management commands, see *Chapter 3. Basic Operations of Server Management Commands* in the *uCosminexus Application Server Application Setup Guide*.

- Settings for each J2EE application
  You use the application properties file to specify the following settings:
  - The `<managed-by-ctm>` element can be used to set whether to use CTM.
  - The `<scheduling>` element can be used to set the name, length, and other attributes of the schedule queue.
  - The `<scheduling-unit>` element can be used to select the schedule queue deployment basis (on a J2EE application basis or bean basis).
- Settings for each stateless session bean
  You use the session bean properties file to specify the following settings:
  - The `<enable-scheduling>` element can be used to specify the scheduling-target stateless session bean included in the J2EE application.
  - The `<maximum>` or `<minimum>` element in the `<pooled-instance>` element in the `<stateless>` element can be used to set the maximum or minimum number of instances to be pooled. Note that to dynamically change the degree of parallelism of CTM during operation, you must set `0` (indefinite) as the maximum.
  - The `<scheduling>` element can be used to set the name, length, and other attributes of the schedule queue.

After you use the `cjgetappprop` command to obtain a properties file, edit the file, and then use the `cjsetappprop` command to apply the new contents to the J2EE applications.

3.5 Controlling priority of requests

This section describes the CTM function that controls the priority of requests.

Priority levels can be assigned to requests controlled by CTM. If EJB clients are assigned priority levels, requests from EJB clients with a higher priority level are dequeued and processed earlier than requests from EJB clients with a lower priority level.

The priority of requests is set as a property of a J2EE server operating as an EJB client, a web container server, or an EJB client application. CTM processes requests from EJB clients with a smaller priority value earlier.

3.6 Dynamically changing the number of concurrent executions of requests

When CTM performs flow-volume control of requests, it can also dynamically change the number of concurrent executions of requests for a schedule queue without stopping CTM daemons. This function can temporarily increase or decrease the number of concurrent executions according to the processing of services managed by schedule queues.

The following table shows the structure of this section.

Table 3–4: Structure of this section (dynamically changing the number of concurrent executions of requests)

Topic type	Title	Location
Description	Mechanism of dynamically changing the number of concurrent executions	*3.6.1*
Settings	Values that can be specified for the number of concurrent executions	*3.6.2*
Operation	Checking the operating status of CTM schedule queues	*3.6.3*
	Changing the maximum number of concurrent executions for a CTM schedule queue	*3.6.4*

Note: This section does not provide *Implementation* and *Notes* types of topics that are specific to this function.

Dynamic change of the number of concurrent executions in CTM is executed by using the `ctmchpara` command. For details about changing the number of concurrent executions for a schedule queue, see *3.6.4 Changing the maximum number of concurrent executions for a CTM schedule queue*. For details about the `ctmchpara` command, see *ctmchpara (change the number of concurrent executions for schedule queues)* in the *uCosminexus Application Server Command Reference Guide*.

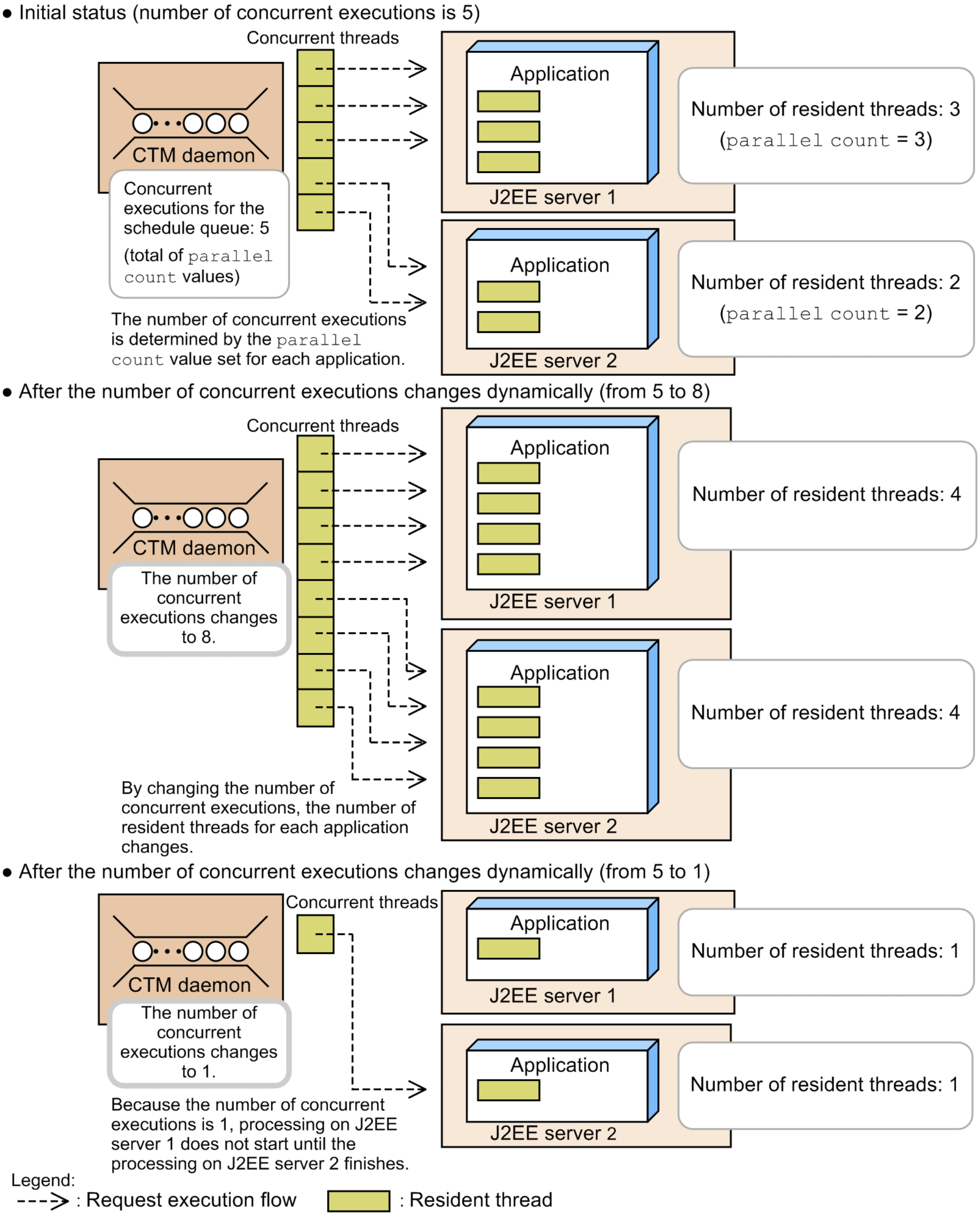
Tip

The number of concurrent executions for a schedule queue changed by the `ctmchpara` command is effective until the CTM daemon is terminated. The change is not applied to `parallel count` set for an individual J2EE application. If a J2EE application is restarted by restarting the CTM daemon, the value of `parallel count` set for that J2EE application takes effect.

3.6.1 Mechanism of dynamically changing the number of concurrent executions

The following figure shows an overview of dynamically changing the number of concurrent threads in CTM.

Figure 3–15: Overview of dynamically changing the number of concurrent executions in CTM



The following subsections describe Figure 3-15.

(1) Initial status (number of concurrent executions is 5)

This subsection describes the status existing when a J2EE server is started before the number of concurrent executions is changed dynamically. The schedule queue managed by the CTM daemon is shared by the J2EE applications on J2EE servers 1 and 2.

For the J2EE application on J2EE server 1, the number of concurrent executions (`parallel count`) is set to `3` as a stateless session bean property. For the J2EE application on J2EE server 2, the number of concurrent executions (`parallel count`) is set to `2` as a stateless session bean property. In this case, the number of concurrent executions for the schedule queue is 5 (3 + 2).

When the CTM daemon receives a request, threads for executing the request are generated as needed on both J2EE servers. No more threads than the value set by `parallel count` for the J2EE application can be generated. The generated threads are made resident in memory (not deleted).

Note that the `parallel count` value can be set or changed by using server management commands.

(2) After the number of concurrent executions changes dynamically (from 5 to 8)

This subsection describes how the system behaves if the number of concurrent executions for a schedule queue dynamically changes to 8.

If the number of concurrent executions for a schedule queue increases dynamically, the number of resident threads that process the requests for each J2EE application increases accordingly.

Note that if the number of resident threads changes, the number of resident threads for each J2EE application that shares the schedule queue is balanced. This process is called *balancing the number of resident threads*. For example, assume that three J2EE servers contain J2EE application instances whose `parallel count` values are 40, 30, and 60, and as many resident threads as those values have been generated. In this case, if the number of concurrent executions for a schedule queue is changed to 120, the number of resident threads for each J2EE server changes to 40 as a result of balancing (120 / 3).

For the case in Figure 3-15, because there are 2 J2EE servers when the number of concurrent executions for a schedule queue is 8, 4 resident threads are generated for each J2EE server.

(3) After the number of concurrent executions changes dynamically (from 5 to 1)

This subsection describes how the system behaves if the number of concurrent executions for a schedule queue is reduced from 5 to 1.

As in the case where the number of concurrent executions increases, when the number of concurrent executions decreases, the number of resident threads that process the requests for each J2EE application also decreases accordingly. In this case, the number of resident threads for each J2EE application is balanced.

However, if the number of concurrent executions for a schedule queue decreases to fewer than the number of J2EE application instances that share the schedule queue, simple balancing causes some J2EE servers to not receive requests. To prevent this, at least one resident thread is generated for each instance.

For the case in Figure 3-15, because there are 2 J2EE servers when the number of concurrent executions is 1, a minimum of 1 resident thread is generated for each J2EE server. Even in this case, however, no more requests than the number of concurrent executions can be processed concurrently. Therefore, the thread for J2EE server 1 does not execute processing until processing with the thread for J2EE server 2 finishes.

3.6.2 Values that can be specified for the number of concurrent executions

This subsection describes the value that can be specified for the number of concurrent executions if the number of concurrent executions dynamically changes.

The value that can be specified for the number of concurrent executions is an integer from 1 to the number of J2EE application instances that share the schedule queue × 127. 127 is the maximum value for the number of concurrent executions for a J2EE application (`parallel count`).

Note, however, that you cannot specify a value larger than the value that was specified for `-CTMDispatchParallelCount` when the CTM daemon was started.

You cannot specify the following values. If you specify one of the following values, an error is output, and the number of concurrent executions does not change.

- 0
- Value larger than the number of J2EE application instances that share the schedule queue × 127
- Value larger than the value specified for `-CTMDispatchParallelCount` of the `ctmstart` command

3.6.3 Checking the operating status of CTM schedule queues

This subsection describes how to check the operating status of CTM schedule queues. The operating status of CTM schedule queues can be checked by using the `mngsvrutil` management command.

To check the operating status of CTM schedule queues, execute the management command by specifying the `get` subcommand with the `queueApps` argument. When this command is executed, you can obtain information such as the number of concurrent executions specified at startup of a J2EE application and the number of resident threads currently generated for a J2EE application.

The following shows the format and an execution example of the above command.

Format:

```
mngsvrutil -m Management-Server-host-name[:port-number] -u management-user
-ID -p management-password -t logical-server-name get queueApps
```

Execution example:

```
mngsvrutil -m mnghost -u user01 -p pw1 -t myServer get queueApps
```

For details about the `mngsvrutil` command, the subcommands of that command, and the information that can be obtained by using that command, see *mngsvrutil (Management Server management command)* in the *uCosminexus Application Server Command Reference Guide*.

3.6.4 Changing the maximum number of concurrent executions for a CTM schedule queue

This subsection describes how to change the maximum value for the dynamically changed number of concurrent executions of a J2EE application for a schedule queue.

The procedure for changing the maximum value for the dynamically changed number of concurrent executions of a J2EE application for a CTM schedule queue is as follows:

1. Check the current value for the maximum number of concurrent executions for the target CTM schedule queue.

To do this, use the `ctmlsque` CTM command. For details, see *(1) Checking the maximum operating status of CTM schedule queues*.

2. Change the maximum number of concurrent executions for the target CTM schedule queue.
   To do this, use the `ctmchpara` CTM command. For details, see *(2) Checking the number of concurrent executions for a CTM schedule queue*.
3. Check the new value for the maximum number of concurrent executions for the target CTM schedule queue.
   To do this, use the `ctmlsque` CTM command. For details, see *(1) Checking the operating status of CTM schedule queues*.

Note that you can change the maximum number of concurrent executions for a CTM schedule queue when the state of the schedule queue is one of the following:

- A: Scheduling is possible.
- H: Schedule queue is locked.
- C: Locked but scheduling is possible.

(1) Checking the operating status of CTM schedule queues

To check the operating status of CTM schedule queues, execute the `ctmlsque` command with `-CTMAppInfo` specified. This command outputs the information about the J2EE applications that share schedule queues. The following shows the format and an execution example of the above command.

Format:

```
ctmlsque -CTMDomain CTM-domain-name -CTMID CTM-identifier -CTMAppInfo
```

Execution example:

```
ctmlsque -CTMDomain domain01 -CTMID CTM01 -CTMAppInfo
```

For details about the `ctmlsque` command and the information output by the command, see *ctmlsque (output schedule queue information)* in the *uCosminexus Application Server Command Reference Guide*.

(2) Checking the maximum number of concurrent executions for a CTM schedule queue

To change the maximum number of concurrent executions for a CTM schedule queue, execute the `ctmchpara` command. The following shows the format and an execution example of the above command.

Format:

```
ctmchpara -CTMDomain CTM-domain-name -CTMID CTM-identifier -CTMQueue regis
tered-schedule-queue-name -CTMChangeCount number-of-concurrent-executions
```

Execution example:

```
ctmchpara -CTMDomain domain01 -CTMID CTM01-CTMQueue que01 -CTMChangeCount
10
```

After executing the command, confirm that the change has been applied. For details about how to check the status of schedule queues, see *(1) Checking the operating status of CTM schedule queues*.

For details about the `ctmchpara` command and the information output by the command, see *ctmchpara (change the number of concurrent executions for schedule queues)* in the *uCosminexus Application Server Command Reference Guide*.

3.7 Locking and controlling requests

Locking and controlling requests (service lock) is a function that improves system availability by stopping reception of requests to a specific J2EE application, or by enabling replacement or restart of a J2EE application without stopping the entire system.

The following table shows the structure of this section.

Table 3–5: Structure of this section (locking and controlling requests)

Topic type	Title	Location
Description	Overview of locking and controlling requests	*3.7.1*
	Replacing a J2EE application while the system is online	*3.7.2*
	Locking and controlling requests for a J2EE application	*3.7.3*
	Locking and controlling requests for a schedule queue	*3.7.4*
	Holding requests if a J2EE server terminates abnormally	*3.7.5*
Settings	Specifying settings in the execution environment	*3.7.6*

Note: This section does not provide *Implementation*, *Operation*, and *Notes* types of topics that are specific to this function.

3.7.1 Overview of locking and controlling requests

While CTM is scheduling requests, it can lock and control requests for a specific schedule queue. Locking and controlling requests for a schedule queue enables *service lock*, which allows you to replace a specific J2EE application without stopping the system.

Locking and controlling requests of CTM provide the following functions:

- Replacing a J2EE application while the system is online
  You can replace a J2EE application without clearing requests from the schedule queue.
- Locking and controlling requests for a J2EE application
  The system waits for requests to finish before locking a J2EE application.
- Locking and controlling requests for a schedule queue
  The system immediately locks a schedule queue. At this time, the user can select whether to discard the requests in the queue.
- Holding requests if a J2EE server terminates abnormally
  The system holds requests in the schedule queue for a certain length of time if the J2EE server terminates abnormally.

To perform locking and controlling requests, use the `mngsvrutil` management command. For details about this command, see *mngsvrutil (Management Server management command)* in the *uCosminexus Application Server Command Reference Guide*.

3.7.2 Replacing a J2EE application while the system is online

You can replace a J2EE application while the system is online.

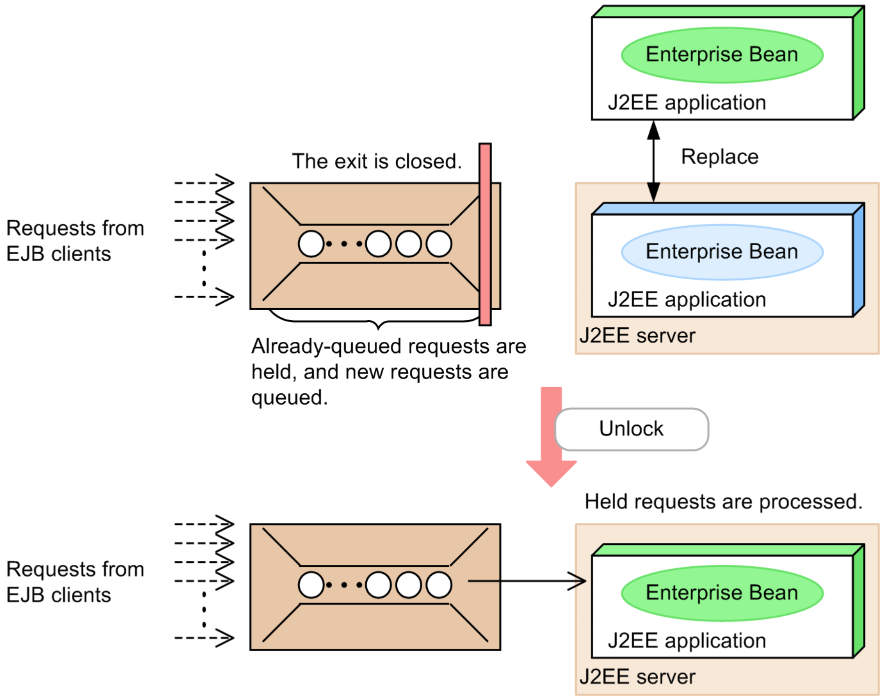
This subsection provides an overview of replacement and describes the replacement procedure.

(1) Overview of replacement

Replace a J2EE application after the CTM daemon closes the exit of the schedule queue. While the exit is closed, requests from clients can be added to the schedule queue. Therefore, system operation can continue without causing the requests for the relevant application to fail. However, if an attempt is made to add a request to the schedule queue when the schedule queue is full, an error is returned to the relevant client.

The following figure shows an overview of replacing a J2EE application while the system is online.

Figure 3–16: Overview of replacing a J2EE application while the system is online



(2) Replacement procedure

Before you can replace a J2EE application while the system is online, the exit of the schedule queue for the J2EE application must be closed. To close the exit and replace the J2EE application, use the `mngsvrutil` management command.

In addition to replacing a specific J2EE application, you can also replace J2EE applications on a host basis or on a management domain basis.

To close the exit of a schedule queue, execute the `mngsvrutil` command with the `hold` subcommand specified. While the exit of a schedule queue is closed, requests from clients can be added to the schedule queue. However, if an attempt is made to add a request when the schedule queue is full, an error is returned to the relevant client.

When you have replaced a J2EE application, unlock the schedule queue. To unlock the schedule queue, execute the `mngsvrutil` command with the `release` subcommand specified. When the schedule queue is unlocked, the J2EE application restarts processing the queued requests.

The following procedure shows how to replace a J2EE application while the system is online by using CTM.

1. Close the exit of the CTM schedule queue for the J2EE application that you want to replace.

The following shows the format and an execution example of the `mngsvrutil` command that is executed when a J2EE application is replaced.

Format:

```
mngsvrutil -m Management-Server-host-name[:port-number] -u management-u
ser-ID -p management-password -t CTM-name hold queue queue-name out
```

Execution example:

```
mngsvrutil -m mnghost -u user01 -p pw1 -t ctm01 hold queue App1 out
```

2. Replace the J2EE application.

   Stop the J2EE application, and then replace it with a new one. After that, start the new J2EE application.

   For details about how to replace a J2EE application, *5.6.3 Replacing and Maintaining a J2EE Application* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

3. Unlock the CTM schedule queue by executing the `mngsvrutil` command with the `release` subcommand specified.

   The following shows the format and an execution example of the `mngsvrutil` command executed in this step.

   Format:

```
mngsvrutil -m Management-Server-host-name[:port-number] -u management-u
ser-ID -p management-password -t CTM-name release queue queue-name
```

   Execution example:

```
mngsvrutil -m mnghost -u user01 -p pw1 -t ctm01 release queue App1
```

For details about the `mngsvrutil` command, see *mngsvrutil (Management Server management command)* in the *uCosminexus Application Server Command Reference Guide*.

3.7.3 Locking and controlling requests for a J2EE application

When you stop a J2EE application, the system can wait until all requests in the schedule queue are processed. If the J2EE application that you stop is the last of the J2EE applications that share the schedule queue, there might be requests in the schedule queue. If the system waits, these requests will be processed successfully.

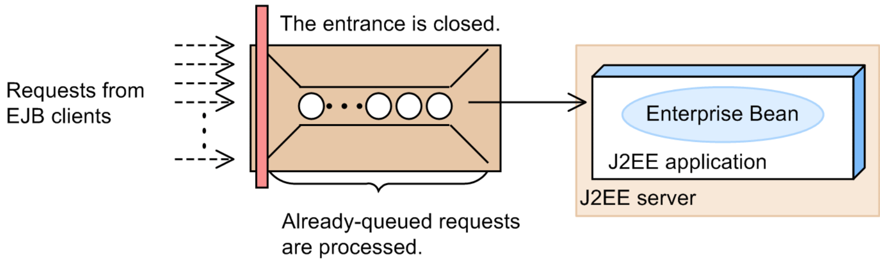
This subsection provides an overview of locking and controlling requests for a J2EE application, and describes the procedure for locking a schedule queue.

(1) Overview of locking and controlling requests for a J2EE application

When the last of the J2EE applications that share a schedule queue stops, the CTM daemon closes the entrance of the schedule queue to perform a service lock so that the queue receives no more requests. After that, the system waits until all requests in the schedule queue are processed, and then stops the J2EE application.

The following figure shows an overview of locking and controlling requests for a J2EE application.

Figure 3–17: Overview of locking and controlling requests for a J2EE application



When CTM executes the locking and controlling of requests for a J2EE application, the following operations take place:

- Reception of new requests stops.
- Processing of already-queued requests that have been distributed to J2EE servers continues.
- For already-queued requests that have not yet been distributed to J2EE servers, a `java.rmi.RemoteException` error is returned.

(2) Procedure for locking schedule queues

Use the management command to lock schedule queues.

The following shows the format and execution examples of the management command executed when all J2EE applications on a specific host are stopped. For details about the management command, see *mngsvrutil (Management Server management command)* in the *uCosminexus Application Server Command Reference Guide*.

Format:

```
mngsvrutil -m Management-Server-host-name[:port-number] -u management-user
-ID -p management-password -t host-name -k host hold queues in:request-com
pletion-wait-time-(sec.)
```

Execution examples:

- To perform a service lock and wait for all requests to be processed, execute the following command:
  `mngsvrutil -m mnghost -u user01 -p pw1 -t host01 -k host hold queues in:0`
- To perform a service lock, continue processing of requests for 5 minutes, and discard requests that are still running, execute the following command:
  `mngsvrutil -m mnghost -u user01 -p pw1 -t host01 -k host hold queues in:300`
- To perform a service lock and immediately discard requests, execute the following command:
  `mngsvrutil -m mnghost -u user01 -p pw1 -t host01 -k host hold queues in:-1`

To unlock schedule queues, execute the `mngsvrutil` command with the `release` subcommand specified. The following shows the format and an execution example of the `mngsvrutil` command.

Format:

```
mngsvrutil -m Management-Server-host-name[:port-number] -u management-user
-ID -p management-password -t host-name -k host release queues
```

Execution example:

```
mngsvrutil -m mnghost01 -u user01 -p pw1 -t host01 -k host release queues
```

3.7.4 Locking and controlling requests for a schedule queue

There are the following two types of locking and controlling requests for a schedule queue:

- Forced locking
- Timeout-triggered locking

This subsection provides an overview of locking and controlling requests for a schedule queue. This subsection also describes the procedures for performing a forced locking and timeout-triggered locking.

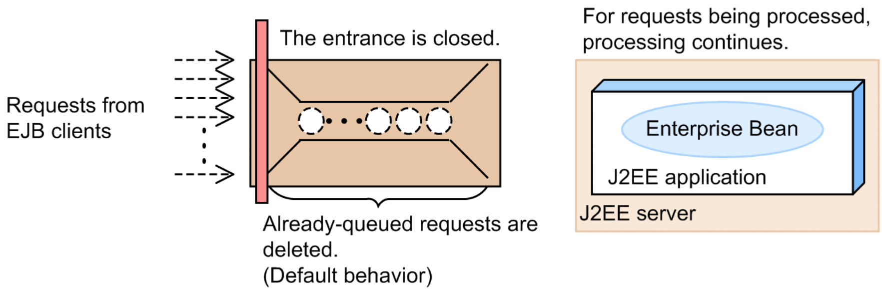
(1) Overview of locking and controlling requests for a schedule queue

You can directly lock a schedule queue. This allows you to simultaneously stop all J2EE applications that share the schedule queue. You can select whether to discard requests remaining in the schedule queue immediately, or continue processing them for a certain length of time. If you choose to continue processing, you can specify the timeout value to forcibly discard the requests that are not processed in the specified time. Processing of requests that are being executed on J2EE servers continues.

In response to a request to lock a schedule queue, the CTM daemon closes the exit of the schedule queue to perform a service lock so that the queue receives no more requests. Already-queued requests are handled (discarded or processed) as pre-specified, and then the lock of the schedule queue is completed. If queued requests are discarded, they are returned to the clients as errors. If queued requests are processed, after processing continues for a preset time, any requests whose processing does not end within the time are returned as errors.

The following figure shows an overview of locking and controlling requests for a schedule queue.

Figure 3–18: Overview of locking and controlling requests for a schedule queue



In a back-end system that uses CTM, to simultaneously stop all J2EE applications on a host or all J2EE applications that share a schedule queue, directly lock the schedule queue for those J2EE applications. You can then stop the J2EE applications.

If you use the management command to directly lock schedule queues for J2EE applications, you can stop J2EE applications for each schedule queue that is shared by J2EE applications, for each host, or for each management domain. When you do so, you can select whether queued requests are to be discarded immediately or processed for a certain length of time. Any queued requests that you choose to discard are returned to the clients as errors. If you choose to process queued requests for a certain length of time, requests whose processing does not end in the specified time are returned to the clients as errors.

How you can use the management command to perform a CTM service lock is described below. The following subsections describe two types of locks: forced locking and timeout-triggered locking. Use forced locking in cases such as when you must lock a queue immediately because the CTM daemon is in a high load state.

(2) Forced locking of a schedule queue

You can lock a schedule queue without it communicating with the CTM daemon. This is called *forced locking of a schedule queue*. Forced locking can be used to immediately lock a queue in such a case where the load on the CTM daemon is high. If you use normal locking, the schedule queue communicates with the CTM daemon, and then discards the requests in the queue. In this case, if the CTM daemon is in a high-load state, communication with the CTM daemon takes time. Therefore, time is also needed before the queue is locked.

If you use forced locking, you can immediately lock the queue because communication between the queue and the CTM daemon is skipped. Note that discarding of queued requests takes place the next time the CTM daemon exchanges the load information with other CTM daemons.

To perform forced locking, execute the `mngsvrutil` management command with the `hold` subcommand and `queue force` argument specified. When you perform forced locking of a schedule queue, the requests in the queue are discarded after a certain length of time. If you do not want to discard the requests, execute the `ctmholdque` command with the `-CTMRequestLeave` option specified.

You can unlock a schedule queue locked by forced locking in the same way as you unlock a schedule queue locked in the ordinary manner. For details about the `mngsvrutil` command, see *mngsvrutil (Management Server management command)* in the *uCosminexus Application Server Command Reference Guide*.

The following shows the format and an execution example of the `mngsvrutil` command executed to perform forced locking.

Format:

```
mngsvrutil -m Management-Server-host-name[:port-number] -u management-user
-ID -p management-password -t host-name -k host hold queues force
```

Execution example:

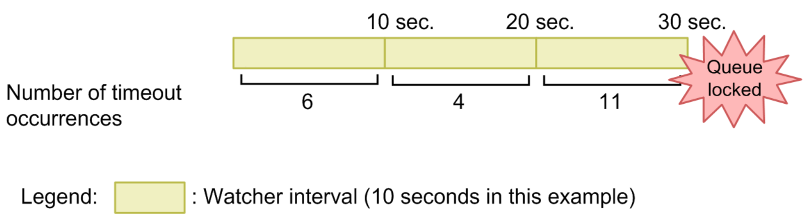
```
mngsvrutil -m mnghost -u user01 -p pw1 -t host01 -k host hold queues force
```

(3) Timeout-triggered locking of a schedule queue

The CTM daemon of a schedule queue monitors EJB client timeouts at regular intervals, and locks the schedule queue when the number of timeout occurrences exceeds a preset value. This is called *timeout-triggered locking of a schedule queue*.

The following describes timeout occurrences. First, see the following figure.

Figure 3–19: How timeout-triggered locking of a schedule queue occurs



In the above figure, the number of timeout occurrences is watched every 10 seconds. Counting of timeout occurrences continues only within each watcher interval. When a watcher interval ends, the counter is reset, and the next watcher interval starts.

For example, assume that the timeout count threshold is set to 10. In this case, if 10 or more timeouts occur in a 10-second watcher interval, the queue is locked. Note that if 10 timeout occurrences are detected, locking of a queue takes place when the next watcher interval starts. In the above figure, 11 timeouts are detected 30 seconds after the start of watching. Therefore, the queue is locked after 30 seconds has elapsed since watching started.

Timeout-triggered locking for a schedule queue is set by using an option specified during startup of the CTM daemon. Specify the `-CTMWatchRequest` option when executing the `ctmstart` command.

3.7.5 Holding requests if a J2EE server terminates abnormally

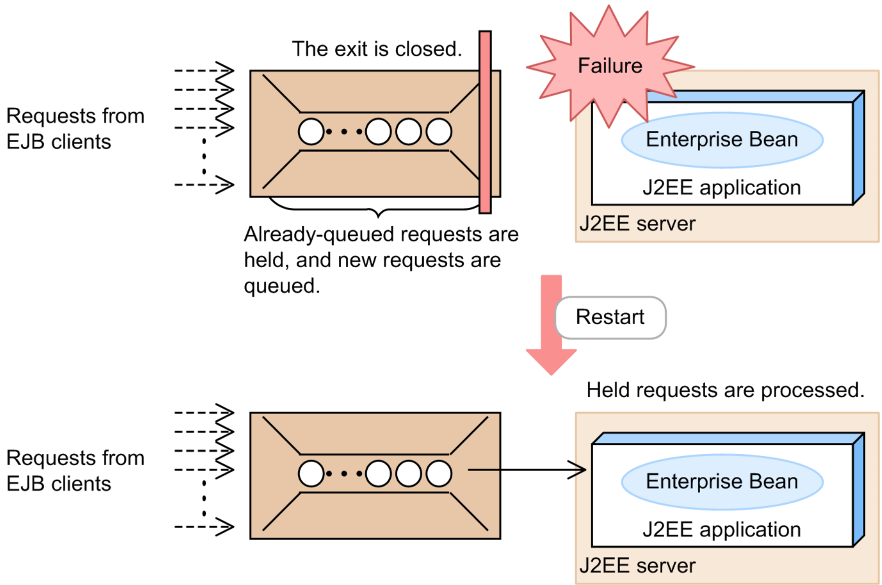
If a J2EE server terminates abnormally, the requests in the schedule queue are held for a certain length of time.

For this reason, errors are not immediately returned to users when a J2EE server terminates abnormally. In addition, the schedule queue continues to receive requests from clients until the J2EE server restarts. The schedule queue can receive requests while it is not full. Therefore, if a J2EE server fails, you can continue operation without clients noticing the failure by restarting the J2EE server immediately. Note, however, that errors are returned to the clients if the number of queued requests exceeds the maximum.

To set this function, specify the `-CTMQueueDeleteWait` option when executing the `ctmstart` command. For details about this command, see *ctmstart (start CTM daemon)* in the *uCosminexus Application Server Command Reference Guide*.

The following figure shows an overview of holding requests if a J2EE server terminates abnormally.

Figure 3–20: Overview of holding requests if a J2EE server terminates abnormally



3.7.6 Specifying settings in the execution environment

Before you can use timeout-triggered locking of a schedule queue, you must specify the necessary CTM daemon settings.

To specify the CTM daemon settings, use the Easy Setup definition file. Timeout-triggered locking is related to load-balancing of requests. The location in which to define load-balancing of requests is the `<configuration>` element for logical CTM (component-transaction-monitor) in the Easy Setup definition file.

The following table describes the parameters related to timeout-triggered locking for a schedule queue in the Easy Setup definition file.

Table 3–6: Parameters related to timeout-triggered locking for a schedule queue in the Easy Setup definition file

Parameter	Description
`ctm.RequestCount`	Specifies the number of timeouts that can occur before the queue is locked.
`ctm.RequestInterval`	Specifies the time interval in which the number of timeouts that occurred is counted.
`ctm.WatchRequest`	Specifies whether to lock the queue when transmission of a request to the J2EE server times out.

For details about the Easy Setup definition file and the parameters that can be specified in the file, see *4.3 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

3.8 Load balancing of requests

Load balancing is a function that improves overall system availability by equally distributing processing to parallel J2EE servers (typically seen in a cluster configuration). Load balancing can distribute `create` and `invoke` requests from clients to servers, processes, and threads.

The following table shows the structure of this section.

Table 3–7: Structure of this section (load balancing of requests)

Topic type	Title	Location
Description	Times when load balancing takes place	*3.8.1*
	Watching the load status	*3.8.2*
Settings	Specifying settings in the execution environment	*3.8.3*

Note: This section does not provide *Implementation*, *Operation*, and *Notes* types of topics that are specific to this function.

Load balancing can be performed across J2EE applications that share a schedule queue. By exchanging load information among CTM daemons, load balancing can also be performed for business-processing programs included in J2EE applications controlled by different schedule queues.

3.8.1 Times when load balancing takes place

CTM performs load balancing at the following two times:

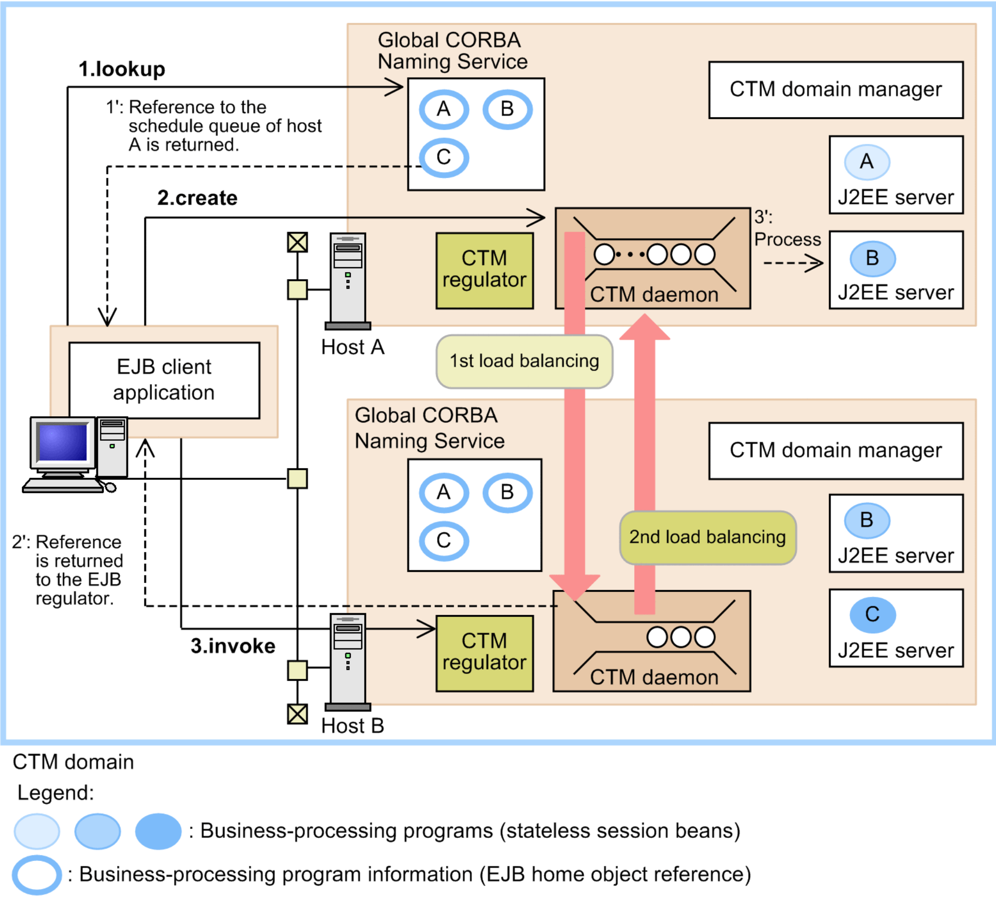
- When an EJB object reference is obtained by a `create` request

  If load balancing takes place at this time, the `create`-based selection policy decides whether processing is assigned preferentially to the CTM daemon that received the request, or to the CTM daemon that is least loaded.
- When a business method is executed through a remote interface by an `invoke` request

  If load balancing takes place at this time, the schedule policy decides whether processing is assigned preferentially to the CTM daemon that received the request, or to the CTM daemon that is least loaded.

The following figure shows an overview of how a client calls business-processing programs and the times when load balancing takes place.

Figure 3–21: How an EJB client calls business-processing programs and the times when load balancing takes place



Note: The processes of Smart Agent and the PRF daemon are omitted.

The following describes the processing illustrated in the above figure.

1. The EJB client executes `lookup` for one of the global CORBA Naming Services deployed on hosts.

   In the above figure, `lookup` is executed for host A.

   In each global CORBA Naming Service, references to schedule queues are registered. In the above figure, host A returns a registered schedule queue reference.

2. The EJB client uses the obtained reference to execute `create`.

   In the above figure, `create` is executed for the CTM daemon on host A.

   At this time, the first load balancing takes place.

   The `create`-based selection policy decides how to balance the load.

   The CTM daemon that received the `create` request returns either of the following references to the EJB client based on the `create`-based selection policy:

   - Reference to the CTM regulator that corresponds to the CTM daemon on the host that received the `create` request
   - Reference to the CTM regulator that corresponds to the CTM daemon that is least loaded in the CTM domain

   In the above figure, a reference to the CTM regulator on host B is returned.

3. The EJB client uses the obtained reference to execute the `invoke` or `remove` request defined in the remote interface.

   In the above figure, the `invoke` request is executed for the CTM regulator on host B. The CTM regulator then sends the request to the CTM daemon.

   At this time, the second load balancing takes place.

   The schedule policy decides how to balance the load when the `invoke` request is executed.#

   In the above figure, processing is assigned to the CTM daemon on host A, which received the request. The request is then registered in the schedule queue. When the request is executed, it is associated with the already-pooled reference to an EJB object, and the relevant business-processing program on the J2EE server is called. At this time, a J2EE server that was terminated abnormally or a business-processing program that timed out due to a hang is never called.

   #

   The schedule policy is not used for execution of `remove`.

A reply from a business-processing program is returned to the EJB client via the CTM daemon that received the request.

3.8.2 Watching the load status

CTM can watch the load status of schedule queues. CTM performs watching of the load status at an interval specified for each J2EE server. Use an argument to specify the interval when executing the `ctmstart` command to start a CTM daemon. If the system you are using is a system that was set up by using the management portal, you can set the interval in logical CTM beforehand. For details about the `ctmstart` command, see *ctmstart (start CTM daemon)* in the *uCosminexus Application Server Command Reference Guide*.

3.8.3 Specifying settings in the execution environment

Before you can use load balancing of requests, you must specify the necessary CTM daemon settings.

To specify the CTM daemon settings, use the Easy Setup definition file. Load-balancing of requests is defined by the `<configuration>` element for logical CTM (component-transaction-monitor) in the Easy Setup definition file.

The following table describes the parameters for load balancing of requests in the Easy Setup definition file.

Table 3–8: Parameters for load balancing of requests in the Easy Setup definition file

Type	Parameter	Description
Times when load balancing takes place	`ctm.CreatePolicy`	Specifies the CTM node selection policy for the `create` request. This policy is used at the first load balancing.
	`ctm.DispatchPolicy`	Specifies the request schedule policy. This policy is used at the second load balancing.
Watching the load status	`ctm.LoadCheckInterval`	Specifies the time interval at which to watch the load status of the schedule queue.

For details about the Easy Setup definition file and the parameters that can be specified in the file, see *4.3 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

3.9 Monitoring and retaining request queues

The following table shows the structure of this section.

Table 3–9: Structure of this section (monitoring and retaining request queues)

Topic type	Title	Location
Description	Overview of monitoring requests remaining in a schedule queue	*3.9.1*
	Example of monitoring a schedule queue	*3.9.2*
Settings	Specifying settings in the execution environment	*3.9.3*
Notes	Notes	*3.9.4*

Note: This section does not provide *Implementation* and *Operation* types of topics that are specific to this function.

On a J2EE server, if it takes time for the CTM daemon to dequeue requests from the schedule queue, requests might remain in the queue for a long time. These requests are monitored by using the *schedule queue monitoring function*. This function is described below.

3.9.1 Overview of monitoring requests remaining in a schedule queue

The schedule queue monitoring function monitors the number of requests in a schedule queue. If the number of queued requests exceeds a certain percentage, the CTM daemon outputs a message and terminates abnormally.

A schedule queue is monitored as follows:

1. The schedule queue monitoring function starts when the preset threshold percentage value of used capacity of the queue is exceeded.
2. When the function starts, it checks the filling status of the schedule queue at the specified time interval.
3. When the function performs a check, if the following expression is true, the CTM daemon terminates abnormally:

   Schedule queue monitoring expression:

   $(P / C_{n-1}) < (M1 / 100)$

   P: Number of requests processed during the period from the previous check to the current time

   C_{n-1}: Number of queued requests at the check before the last check (C_n)

   M1: Threshold for stopping the system (system processing percentage)

To use the schedule queue monitoring function, specify the `-CTMWatchQueue` option when executing the `ctmstart` command. For details about the `ctmstart` command, see *ctmstart (start CTM daemon)* in the *uCosminexus Application Server Command Reference Guide*.

3.9.2 Example of monitoring a schedule queue

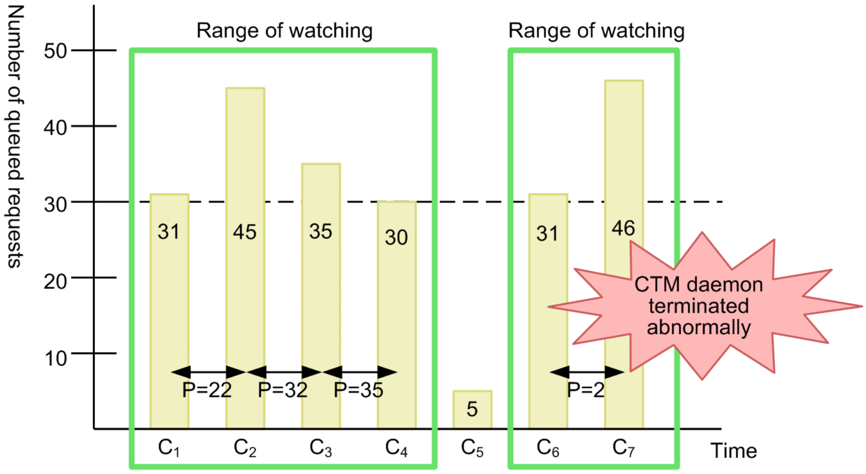
This subsection provides an example to explain the monitoring of a schedule queue.

The example in this subsection assumes that the following settings have been specified:

- Queue filling percentage triggering schedule queue monitoring: 60%

- Threshold for stopping the system: 70%
- Schedule queue check interval: 1 second

Figure 3–22: Example of schedule queue monitoring



Legend: *Cn*: Watching point P: Number of processed requests

In this example, the threshold of the system processing percentage for stopping the system is set to 70%. Therefore, the right side of the schedule queue monitoring expression (M1 / 100) is 70 / 100 (= 0.7). Therefore, the expression in this example is as follows:

Schedule queue monitoring expression in this example:

$(P / C_{n-1}) < 0.7$

In this case, the CTM daemon terminates abnormally when the left side (P / C_{n-1}) is less than 0.7.

This example also assumes that the maximum number of requests that can be contained in the schedule queue is 50. Therefore, 60% of schedule queue filling percentage means 30 queued requests. Therefore, monitoring of the schedule queue starts when the number of queued requests exceeds 30.

The following describes the processing of schedule queue monitoring for each check point:

C_1

At check point C_1, 31 requests are in the schedule queue. Because the schedule queue filling percentage exceeds 60% (or, 30 requests), monitoring of the schedule queue starts.

C_2

At check point C_2, P (number of requests processed from C_1 to C_2) = 22. Therefore, the left side of the schedule queue monitoring expression (P / C_{n-1}) is as follows:

$(P / C_1) = 22 / 31 = 0.7$

Because the resulting value (0.7) is equal to the threshold for stopping the system (70%), the CTM daemon does not stop.

Also, the number of queued requests at C_2 is 45. Because this number exceeds the queue filling percentage triggering schedule queue monitoring, 60% (or, 30 requests), monitoring of the schedule queue continues.

C_3

At check point C_3, P (number of requests processed from C_2 to C_3) = 32. Therefore, the left side of the schedule queue monitoring expression (P / C_{n-1}) is as follows:

$(P / C_2) = 32 / 45 = 0.71$

Because the resulting value (0.71) is greater than the threshold for stopping the system (70%), the CTM daemon does not stop.

Also, the number of queued requests at C_3 is 35. Because this number exceeds the queue filling percentage triggering schedule queue monitoring, 60% (or, 30 requests), monitoring of the schedule queue continues.

C_4

At check point C_4, P (number of requests processed from C_3 to C_4) = 35. Therefore, the left side of the schedule queue monitoring expression (P / C_{n-1}) is as follows:

$(P / C_3) = 35 / 35 = 1$

Because the resulting value (1) is greater than the threshold for stopping the system (70%), the CTM daemon does not stop.

Also, the number of queued requests at C_4 is 30. Because this number is equal to the queue filling percentage triggering schedule queue monitoring, 60% (or, 30 requests), monitoring of the schedule queue stops.

C_5

At check point C_5, P (number of requests processed from C_4 to C_5) = 5. Therefore, the left side of the schedule queue monitoring expression (P / C_{n-1}) is as follows:

$(P / C_4) = 5 / 30 = 0.16$

Although this value is less than 0.7 (70%), the threshold for stopping the system, the CTM daemon does not stop because the schedule queue is not monitored at C_5.

C_6

At check point C_6, 31 requests are in the schedule queue. Because the queue filling percentage triggering schedule queue monitoring, 60% (or, 30 requests), is exceeded, monitoring of the schedule queue starts.

C_7

At check point C_7, P (number of requests processed from C_6 to C_7) = 2. Therefore, the left side of the schedule queue monitoring expression (P / C_{n-1}) is as follows:

$(P / C_6) = 2 / 31 = 0.06$

Because this value is less than 0.7 (70%), the threshold for stopping the system, the CTM daemon terminates abnormally.

3.9.3 Specifying settings in the execution environment

Before you can monitor the number of requests in a schedule queue, you must specify the necessary CTM daemon settings.

To specify the CTM daemon settings, use the Easy Setup definition file. Schedule queue monitoring is related to load-balancing of requests. The location in which to define load-balancing of requests is the `<configuration>` element for logical CTM (component-transaction-monitor) in the Easy Setup definition file. For the `ctm.QueueRate` parameter, specify the queue filling percentage as the threshold that triggers start of schedule queue monitoring.

For details about the Easy Setup definition file and the parameters that can be specified in the file, see *4.3 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

3.9.4 Notes

- For a request queue that is monitored, if the queue locking command (`ctmholdque`) is used to discard queued requests, they are treated as requests that were already processed.
- If the queue locking command (`ctmholdque`) is executed for a monitored request queue, the monitoring state changes as follows:
  - If the queue is locked normally (`ctmholdque` without options)

    Because all requests in the queue are discarded, the number of queued requests decreases. As a result, monitoring of the queue ends.
  - If the entrance of the queue is locked (`ctmholdque` with the `-CTMRequestLeave` option)

    Because all requests in the queue are processed by servers, monitoring of the queue continues.
  - If the exit of the queue is locked (`ctmholdque` with the `-CTMChangeServer` option)

    The requests in the queue are not processed. Because the system processing percentage is 0, the system stops. As a result, monitoring of the queue ends.

3.10 Connection with the TPBroker/OTM client by using the gateway functionality in CTM

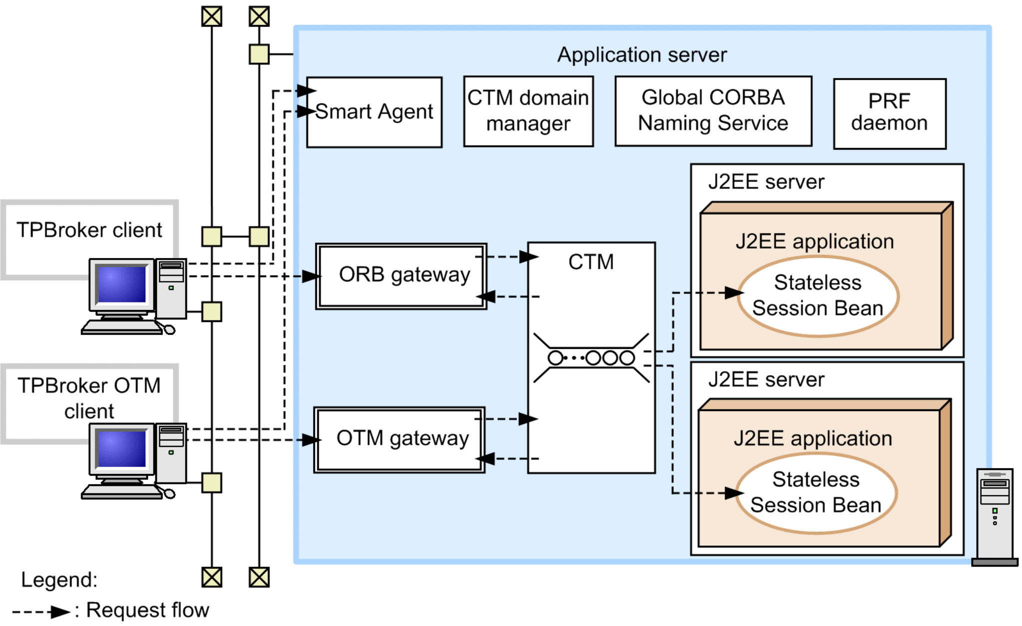
CTM provides gateway functionality, which allows the following types of clients to call J2EE applications that operate on application servers:

- TPBroker clients
  Client applications developed by using TPBroker Version 5 or later.
- TPBroker OTM clients
  Client applications developed by using TPBroker Object Transaction Monitor.

Also, CTM can output analysis information about the gateways that send and receive requests. This information can be converted into CSV format. The converted information can then be used for further analysis with analysis information output by other J2EE server functions. For details about performance analysis trace output, see the description of performance trace analysis in *Chapter 7. Performance Analysis by Using Trace Based Performance Analysis* in the *uCosminexus Application Server Maintenance and Migration Guide*.

The following figure shows an overview of calling J2EE applications from a TPBroker client or TPBroker OTM client by using the CTM gateway functionality.

Figure 3–23: Overview of calling J2EE applications from a TPBroker client or TPBroker OTM client



When the TPBroker client sends a request to the J2EE application on a J2EE server, the request passes through an ORB gateway. When the TPBroker OTM client sends a request to the J2EE application on a J2EE server, the request passes through an OTM gateway. The ORB and OTM gateways are processes provided by CTM, and are started when the CTM daemon is started.

The following describes how TPBroker and TPBroker OTM clients can send requests to J2EE applications, and how references can be resolved.

- For TPBroker clients
  If `1` is specified for the `-CTMAgent` or `-CTMIDLConnect` option of the `ctmregltd` command, the CTM regulator enables the ORB gateway functionality. If `1` is specified for `-CTMAgent`, a CORBA reference is registered in the smart agent by using the EJB lookup name as an object name. Therefore, the TPBroker client resolves

the reference destination by specifying the EJB lookup name as an argument for `_bind()`. If `1` is specified for `-CTMIDLConnect`, the TPBroker client resolves the reference destination by using the `ctmgetior` command to obtain the IOR string.

- For TPBroker OTM clients

  If `1` or a larger value is specified for the `-CTMTSCGwStart` option of the `ctmstart` command, the OTM gateway starts. On the TPBroker OTM client, specify the EJB lookup (registration) name as a TSC acceptor name for an argument of the constructor that generates a TSC user proxy. Note that the TSC acceptor name cannot be omitted. Select *TSC regulator* as the connection protocol.

You might want to develop client applications that allow TPBroker clients or TPBroker OTM clients to call applications on J2EE servers. For details about how to develop such applications, see the documentation for TPBroker or TPBroker Object Transaction Monitor.

4 Scheduling of Batch Applications

If you use the scheduling functionality of batch applications, you can control the execution requests of batch applications. As a result, you can receive multiple execution requests for batch applications without changing the number of batch servers. This chapter gives an overview of how to schedule batch applications. This chapter describes how to execute batch applications by using the scheduling functionality and the settings required for using the scheduling functionality.

You can use the scheduling functionality only in the products that contain Cosminexus Component Transaction Monitor in the component software. For details on the products that you can use, see *2.2.1 Mapping between products and the component software* in the manual *uCosminexus Application Server Overview*.

4.1 Organization of this chapter

The *batch application scheduling functionality* is the functionality that uses CTM to control the execution requests of batch applications to be executed on batch servers. Hereafter, this functionality is called *scheduling functionality*.

The following table describes the organization of this chapter.

Table 4–1: Organization of this chapter (Scheduling of batch applications)

Classification	Title	Reference location
Description	Overview of the scheduling functionality	*4.2*
	Systems using the scheduling functionality	*4.3*
	Setting and operating the batch application execution environment when using the scheduling functionality	*4.4*
	Executing batch applications by using the scheduling functionality	*4.5*
	Migrating to the environment using the scheduling functionality	*4.6*
Setup	Settings of the execution environment	*4.7*
Notes	Points to be considered when using the scheduling functionality	*4.8*

Note

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

For details on the functionality provided with batch servers and the generation of batch applications, see *2. Executing Applications by Using Batch Servers*.

4.2 Overview of the scheduling functionality

This section gives an overview of the scheduling functionality.

With Application Server, use CTM for scheduling batch applications. The CTM uses a queue to control the execution of batch applications. This queue is called a *schedule queue*.

4.2.1 Advantages of scheduling batch applications

This subsection describes the advantages of using the scheduling functionality.

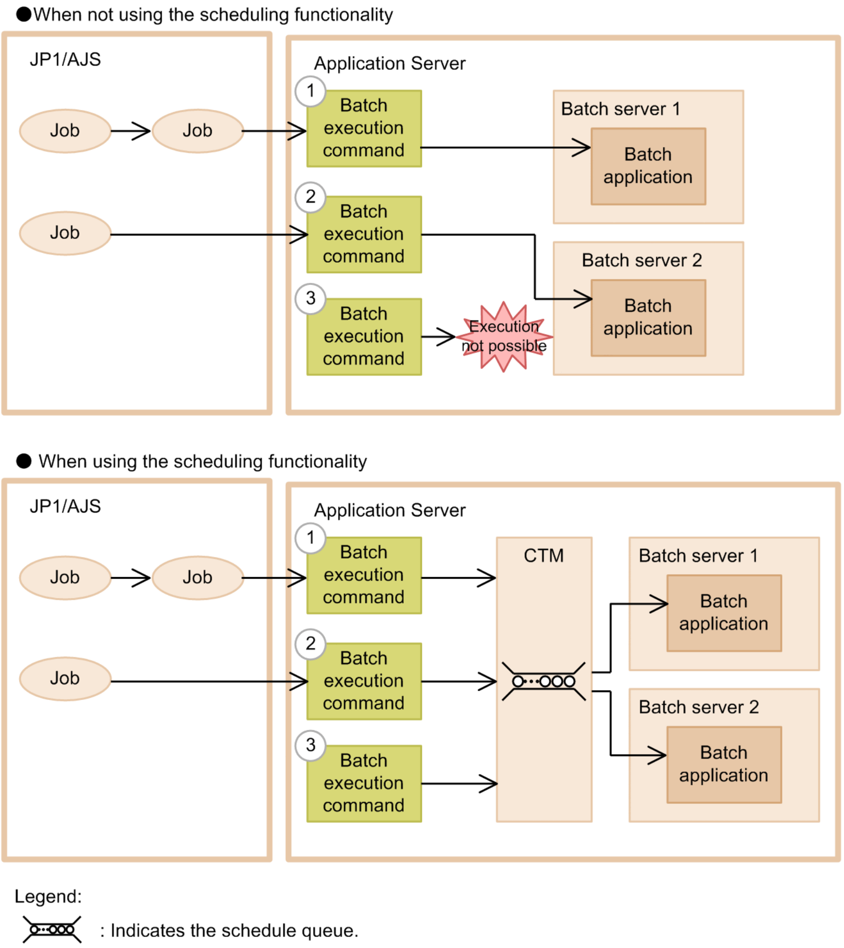
With a batch server, you can execute one batch application at a time. You use the batch execution commands provided with Application Server to start batch applications. The batch server after receiving a batch application execution request from the batch execution command, starts the batch application.

If you do not use the scheduling functionality, the batch application execution requests, exceeding the number of batch servers, cannot be received. In such cases, the requests that are not received, give error. Moreover, in the batch execution command, you are required to define the batch server on which you want to execute an application.

If you use the scheduling functionality, the batch application execution requests, exceeding the number of batch servers, are accumulated in a schedule queue by using CTM, and an error does not occur. The accumulated requests are distributed to batch servers by using CTM. As a result, you can execute the batch execution commands irrespective of the number of batch servers. Also, the CTM distributes the batch application execution requests to batch servers, so you need not define the batch server on which you want to execute an application, in the batch execution command.

The following figure shows the flow of execution of batch applications when you use and do not use the scheduling functionality.

Figure 4–1: Flow of execution of batch applications when using and not using the scheduling functionality



This figure shows an example in which batch servers concurrently execute the batch execution commands from the JP1/AJS jobs or a direct machine, for two systems.

If you do not use the scheduling functionality, you cannot concurrently execute the batch execution commands shown in steps 2 and 3, in the above figure. If you use the scheduling functionality, you can concurrently execute the batch execution commands shown in steps 2 and 3, because the batch application execution requests are distributed to batch servers by using CTM.

4.2.2 Prerequisites for using the scheduling functionality

This subsection describes the prerequisites for using the scheduling functionality.

If you use the scheduling functionality, using CTM is a prerequisite. For details on the CTM, see *3. Scheduling and Load Balancing of Requests Using CTM*.

To use CTM, you must build a system which is configured to use CTM. For details on the configuration for using the CTM, see *4.3 Systems using the scheduling functionality*.

4.2.3 Procedure for executing the batch applications using the scheduling functionality

This subsection describes the procedure for executing batch applications.

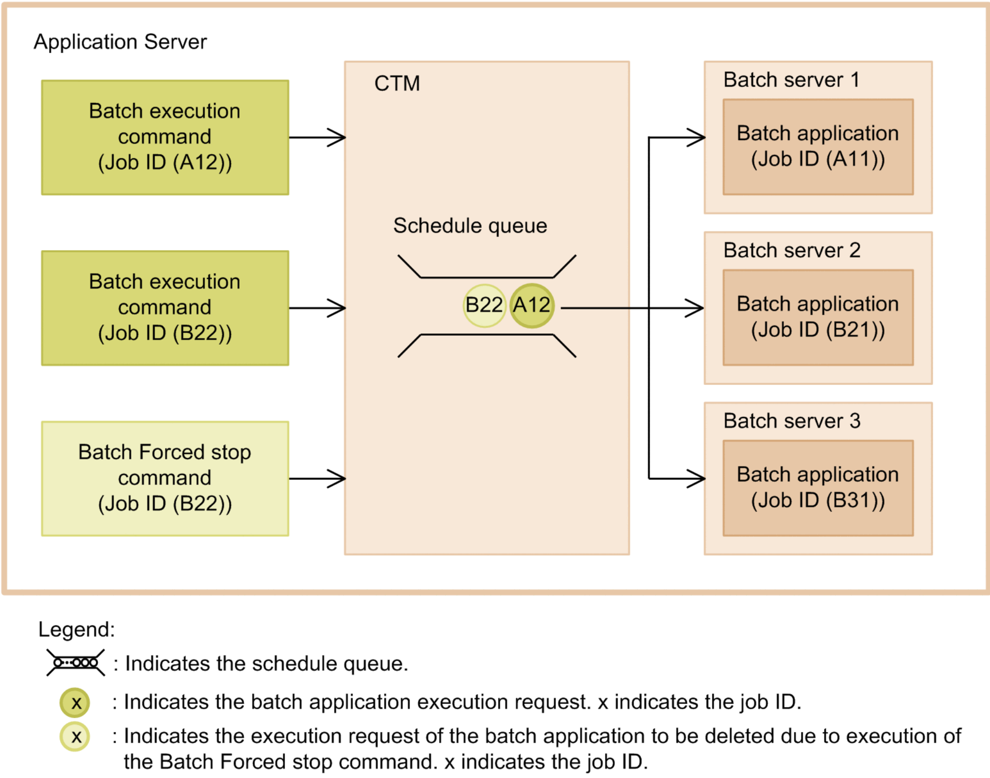
If you use the scheduling functionality, the batch applications executed on batch servers are differentiated by job IDs. A *Job ID* is a string used for differentiating execution requests for the batch applications to be executed. You can specify any value when executing a command. If you omit a job ID while executing a command, the scheduling functionality automatically generates a job ID. This Job ID is managed by CTM.

A batch server group, to which batch applications are distributed using CTM, is called a *schedule group*. A schedule queue is created for respective schedule groups. Specify the schedule group, if you want to control the number of concurrent operations for respective business classifications of batch applications. Set up a unique schedule group in a system. You must set up the schedule groups separately even if CTM is different for each machine. When specifying the schedule groups, you must specify with the batch execution commands and on the batch servers. For details on how to perform settings, see *4.7 Settings of the execution environment*.

Even when you use the scheduling functionality, you can integrate the execution environment of batch applications with JP1/AJS.

The following figure shows the flow of execution of batch applications by using the scheduling functionality.

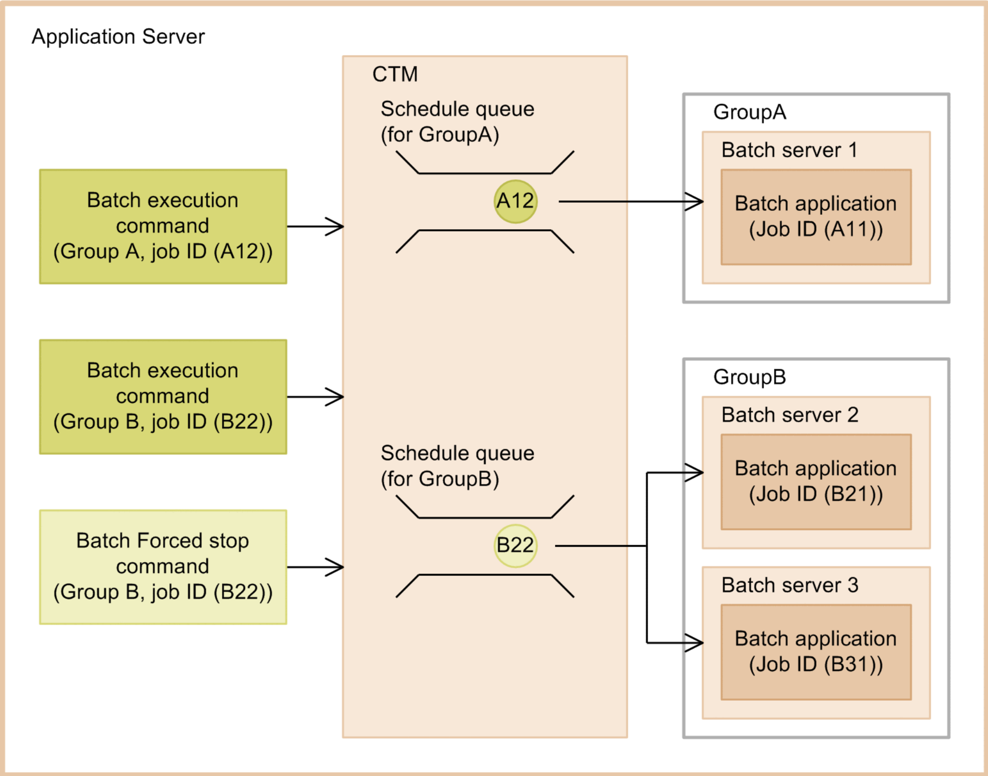
Figure 4–2: Flow of execution of batch applications by using the scheduling functionality



In this figure, the batch application execution requests are distributed from batch server 1 through batch server 3, which belong to the same schedule group, by using CTM. The batch application execution requests which overflow from the schedule queue give an error.

The following figure shows the flow of execution of batch applications by using multiple schedule groups.

Figure 4–3: Flow of execution of batch applications by using multiple schedule groups



Legend:

: Indicates the schedule queue.

x : Indicates the batch application execution request. x indicates the job ID.

x : Indicates the execution request of the batch application to be deleted due to execution of the Batch Forced stop command. x indicates the job ID.

This figure shows an example where two schedule groups `GroupA` and `GroupB` are specified, and two schedule queues are created. You use a command to define the schedule group to be used. The batch applications are distributed to schedule queues according to the schedule group settings of the command. In this figure, batch applications are running on batch servers of schedule groups, so the batch applications received by CTM are on standby in a schedule queue.

4.3 Systems using the scheduling functionality

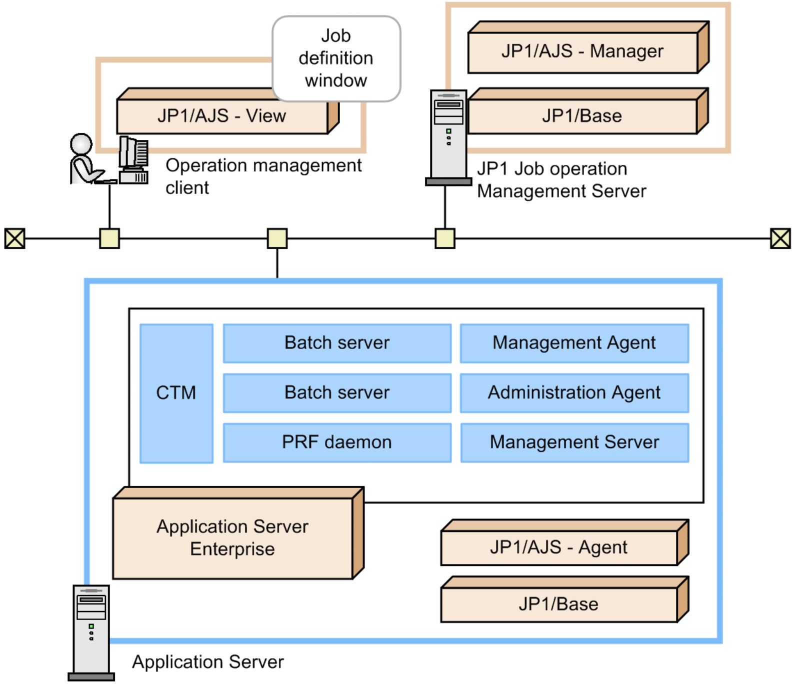
This section describes the system configuration and processes required for a system using the scheduling functionality.

4.3.1 Configuring a system using the scheduling functionality

This subsection describes the configuration of a system using the scheduling functionality.

The following figure shows an example of configuring a system using the scheduling functionality.

Figure 4–4: Example of configuring a system using the scheduling functionality



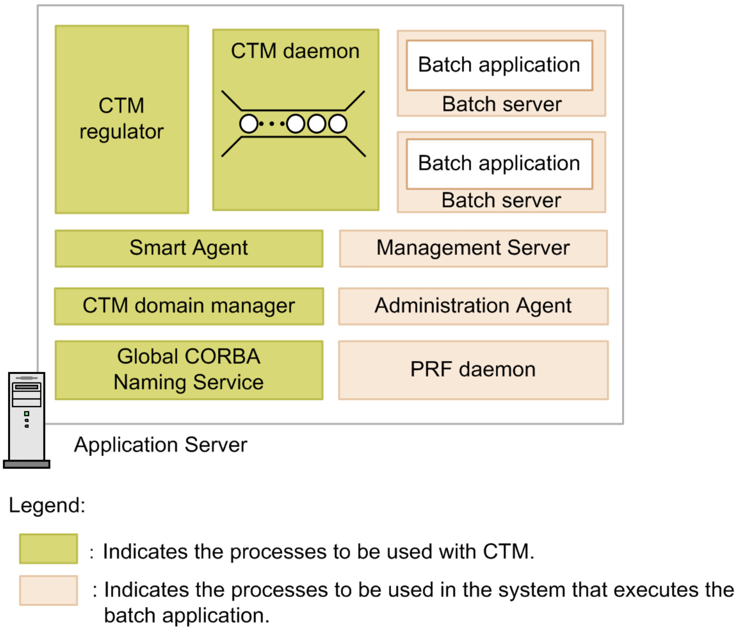
In this figure, the system that is executing batch applications is integrated with JP1/AJS. If you do not integrate the system with JP1/AJS, the management client, JP1 Job Management Server, and JP1/AJS - Agent and JP1/Base of Application Server, shown in the above figure, are not required. For the operation procedures of batch servers and batch applications, see *2.2.2(1) Systems integrated with JP1/AJS* and *2.2.2(3) Systems not integrated with JP1/AJS, BJEX, and JP1/Advanced Shell*.

4.3.2 Processes required for the scheduling functionality

This subsection describes the processes required for the scheduling functionality.

When using the scheduling functionality, you use CTM. The following figure shows an example of the process configuration of Application Server using the scheduling functionality.

Figure 4–5: Example of process configuration of Application Server (For using the scheduling functionality)



The following table gives an overview of the processes used in CTM.

Table 4–2: Overview of the processes used in CTM (For using the scheduling functionality)

Process	Explanation
CTM daemon	This process manages the schedule queues for controlling the batch application execution requests.
CTM regulator	This process distributes and consolidates the batch application execution requests accumulated in a CTM daemon.
CTM domain manager	This process manages the information of CTM daemons in the same CTM domain.
Global CORBA Naming Service	This Naming Service shares and manages the batch application information on the hosts that are included in the same CTM domain.
Smart Agent	This process provides the dynamically distributed directory services that are provided with Cosminexus TPBroker. This process is used when distributing the information to CTM daemons in different network segments. For details, see the *Borland(R) Enterprise Server VisiBroker(R) Developers Guide*.

For details on the guidelines to arrange the processes and for the respective processes, see *3. Scheduling and Load Balancing of Requests Using CTM*.

The *PRF daemon* (performance tracer) is also used in CTM as an I/O process to output the performance analysis information, which is output by the CTM daemon. For details, see *7.2.1 Overview of the trace based performance analysis of Application Server* in the *uCosminexus Application Server Maintenance and Migration Guide*.

4.4 Setting and operating the batch application execution environment when using the scheduling functionality

This section describes how to set up and operate an execution environment for batch applications.

Even if you use the scheduling functionality, you use the Smart Composer functionality and server management commands to build an execution environment for batch applications. In this case, the same operation procedures and usable operation functionality of the batch application execution environment are used as when the scheduling functionality is not used.

For the procedure of building a batch application execution environment, see *2.2.3(1) Setting up the batch application execution environment*. However, when using the scheduling functionality, you are required to set up an environment and build CTM and Smart Agent in addition to defining and setting up a batch server. You also use the Smart Composer functionality when building CTM and Smart Agent. For details, see *4.6 Setting up a system for executing batch applications* in the *uCosminexus Application Server System Setup and Operation Guide*. For details on the operations that can be performed in a batch application execution environment and the operation procedures, see *2.2.3(2) Operating batch application execution environment*.

You can integrate with JP1 and cluster software even in a system that uses the scheduling functionality. For details, see *2.2.3(3) Integrating with other programs*.

4.5 Executing batch applications by using the scheduling functionality

This section describes the execution of batch applications by using the scheduling functionality. For details on the batch application execution functionality, see *2.3.1 Overview of the batch application execution functionality*.

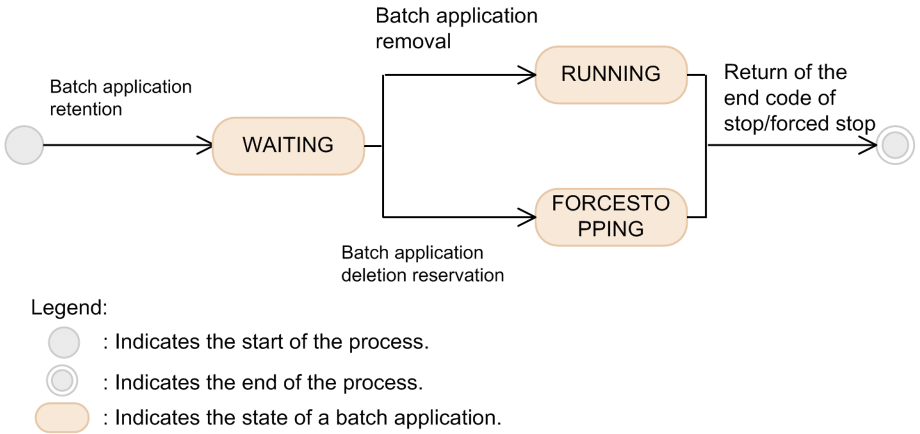
The same execution logs of batch applications are output by batch servers as those that are output when using the scheduling functionality. For details on the execution logs of batch applications, see *2.3.5 Log output of a batch application*.

4.5.1 Status transition of batch applications using the scheduling functionality

This subsection describes the status transition of batch applications using the scheduling functionality.

The following figure shows the status transition of batch applications.

Figure 4–6: Status transition of batch applications (Using the scheduling functionality)



The following table describes the respective status of batch applications.

Table 4–3: Description of the respective status of batch applications

Status of batch application	Explanation
`WAITING`	This is a standby status in a schedule queue because another batch application is running on the batch sever.
`RUNNING`	In this status, a batch application is running on a batch server.
`FORCESTOPPING`	In this status, the batch application, which is in the schedule key, is reserved for deletion by the batch forced stop command.

You can confirm the status of batch applications from the batch application information. For details on how to display the batch applications information, see *4.5.4 Displaying a list of batch application information*.

4.5.2 Executing batch applications

You use the `cjexecjob` command to start batch applications. You can use the following two methods to execute the `cjexecjob` command:

1. Directly executing the `cjexecjob` command

   If you do not use JP1/AJS, you can use this method to start a batch application.

2. Defining the `cjexecjob` command as a JP1/AJS job, and executing from JP1/AJS

   If you use JP1/AJS, you can use this method to start a batch application.

For details on the definition of JP1/AJS jobs when starting batch applications by using the *method 2*, see *2.13 Integrating with JP1/AJS*. Start a batch server in advance when you want to execute a batch application from JP1/AJS.

The same points are to be considered when starting, ending and executing the batch applications which are considered when the scheduling functionality is not used. For details, see *2.3.2 Executing batch applications*.

4.5.3 Forced stopping of batch applications

You use the `cjkilljob` command for forced stopping of batch applications. If you use the scheduling functionality, you specify a job ID in the `cjkilljob` command.

If a batch application execution request specified by a job ID is on standby status in a schedule queue, the request is reserved for deletion. The batch application execution request, which is reserved for deletion, is deleted when the request is removed from a schedule queue by using CTM.

If the batch application execution request specified by the job ID is running on a batch server, the application is forcefully stopped.

You use the same method to forcefully stop batch applications and consider the same points when forcefully stopping and executing batch applications as in the case of not using the scheduling functionality. For details, see *2.3.3 Forcefully stopping a batch application*.

4.5.4 Displaying a list of batch application information

You can display a list of items such as the status of batch applications (`running` or `standby`) and the start time of batch execution commands as the batch application information. This subsection describes the list display of the batch application information.

(1) How to display a list of batch application information

To display a list of batch application information, directly execute the `cjlistjob` command, irrespective of whether you are using JP1/AJS.

You can acquire the batch application information in the following units:

- Schedule groups specified in the command argument
- All schedule groups

In the argument of the `cjlistjob` command, you specify the schedule group name, to which the batch server (for which you want to acquire the batch application information) belongs, or the `-all` option. You can specify multiple schedule group names. If you specify the `-all` option, you can acquire the batch application information of all the schedule groups that are used by the batch servers of the same machine.

(2) Processing for displaying a list of batch application information

If you execute the `cjlistjob` command, you can acquire information of the running batch applications in the schedule group, specified in the argument or in `batch.schedule.group.name key of usrconf.cfg` (option definition file for batch applications). The batch application information is output to the standard output.

The following table describes the batch application information that you can acquire.

Table 4–4: Batch application information that you can acquire

Item in batch application information that can be acquired	Content
Schedule group name	The name of a schedule group, in which the batch application execution requests are distributed, is output.
Status of batch application	`running`, `waiting`, or `forceStopping` is output. `running`, `waiting`, and `forceStopping` indicate that the status of batch applications is `RUNNING`, `WAITING`, and `FORCESTOPPING` respectively. For details on the status of batch applications, see *4.5.1 Status transition of batch applications using the scheduling functionality*.
Batch application name	The class name of a batch application, specified in the `cjexecjob` command, is output.
Root application information of the performance analysis trace	The communication number of a root application of the performance analysis trace is output. You can check the status of batch applications by mapping with the root application information output to the performance analysis trace file.
Execution time of the batch execution commands	The time of the `cjexecjob` command execution is output in the following format. Note that Δ indicates a single byte space. `yyyy/mm/ddΔhh:mm:ss.ssssss` `yyyy`: Christian year, `mm`: Month, `dd`: Date, `hh`: Hour, `mm`: Minute, `ss`: Second, `ssssss`: Microsecond
Standby start time, execution start time, and forced stop reception time of batch applications	For every status of batch applications, the time at which the status of a batch application starts is output with the following format. Note that Δ indicates a single byte space. `yyyy/mm/ddΔhh:mm:ss.ssssss` `yyyy`: Christian year, `mm`: Month, `dd`: Date, `hh`: Hour, `mm`: Minute, `ss`: Second, `ssssss`: Microsecond
Job ID	The job Id of a batch application is output.
Batch server name on which batch applications are running	The name of the batch server, on which batch applications are running, is output. When the status of a batch application is `waiting`, "-" is output.

If a batch application does not exist, nothing is output even if you execute the `cjlistjob` command. In this case, the `cjlistjob` command ends successfully.

The following example describes the format and the output of the `cjlistjob` command. Note that Δ indicates a single byte space.

Output format of the cjlistjob command

```
Schedule-group-name Δ State-of-batch-application Δ Batch-application-name Δ
Root-application-information-of-performance-analysis-trace Δ Execution-time-
of-batch-execution-command Δ Standby-start-time-,-execution-start-time-and-f
orced-stop-reception-time-of-batch-application Δ Job-ID Δ Batch-server-name
```

```
-on-which-batch-application-is-running
Schedule-group-name Δ State-of-batch-application Δ Batch-application-name Δ
Root-application-information-of-performance-analysis-trace Δ Execution-time-
of-batch-execution-command Δ Standby-start-time-,-execution-start-time-and-f
orced-stop-reception-time-of-batch-application Δ Job-ID Δ Batch-server-name
-on-which-batch-application-is-running
:
```

Output example of the cjlistjob command

```
JOBGROUP running com.hitachi.mypackage.batchApp1 0x0000000000123456 2008/04/
14 17:27:35.689012 2008/04/14 17:27:37.182777 HOGE MybatchServer1
JOBGROUP running com.hitachi.mypackage.batchApp2 0x0000000002345678 2008/04
/14 17:45:20.123456 2008/04/14 19:21:56.271354 102 MybatchServer2
JOBGROUP running com.hitachi.mypackage.batchApp3 0x0000000034567890 2008/0
4/14 18:15:54.397890 2008/04/14 19:00:00.123447 #5HL390_G3CV7 MybatchServer3
JOBGROUP waitting com.hitachi.mypackage.batchApp4 0x0000000045678901 2008/
04/14 18:30:24.125444 2008/04/14 18:30:25.006220 112345 -
```

This example shows that in the schedule group `JOBGROUP`, batch applications are running on `MybatchServer1`, `MybatchServer2`, and `MybatchServer3`. Also, an execution request of the batch application `batchApp4` is on standby.

4.5.5 Executing the commands used in batch applications

The same types of commands used in batch applications, status of batch servers and execution of commands are used as in the case when a scheduling functionality is not used, except the points described below. The differences are:

- You can execute the `cjexecjob` command even when you are processing the `cjexecjob` command on a batch server.
- When the status of a batch server is any one of the following, and if you execute the `cjexecjob`, `cjkilljob`, or `cjlistjob` command, the KDJE55046-E message is output:
  - When a batch server is starting
  - When a batch server is stopping
  - After the batch server is stopped
- Between the `cjexecjob` command and the batch server, you can set up a time until a timeout occurs between the `cjkilljob` or `cjlistjob` command and CTM. Set up a timeout by using the `batch.request.timeout` key in `usrconf.cfg` (option definition file for batch applications). For details on how to set up a timeout, see *4.7.3 Settings for the commands to be used with batch applications*.

For the points other than these differences, see *2.3.6 Executing commands used in a batch application*.

This subsection describes the countermeasures that you need to take in the case of an abnormal end during the processing of a command used in a batch application, and the points to be considered when executing the commands.

(1) When a batch server ends abnormally while processing a command

When the `cjexecjob`, `cjkilljob`, or `cjlistjob` command is processing on a batch server, if the batch server ends abnormally, the KDJE55021-E message is output. Confirm the status of the batch server and re-execute the command.

(2) When a CTM daemon or a CTM regulator ends abnormally while processing a command

When the `cjexecjob`, `cjkilljob`, or `cjlistjob` command is processing, if the CTM daemon or the CTM regulator ends abnormally, the KDJE55047-E message is output. This message is output if a process ends abnormally while communicating with the CTM daemon or the CTM regulator, after acquiring the schedule group name from Smart Agent. Confirm the status of the CTM daemon and the CTM regulator, and re-execute the command.

(3) Points to be considered when executing commands

The following points are to be considered when executing the commands:

- On a machine having multiple IP addresses, if an IP address is not specified in `usrconf.cfg` (option definition file for batch applications) or in an environment variable, the IP address to which the ORB gateway connects is automatically determined.
- If you use the scheduling functionality, you execute batch applications on the batch server to which CTM distributes the applications. As a result, you cannot directly execute the `cjexecjob` command for batch servers.
- If you execute the `cjkilljob` command for a standby batch application, CTM reserves the batch application for deletion. The batch application, which is reserved for deletion, is deleted when removed from the schedule queue. In such cases, the batch application, which is reserved for deletion, remains in the schedule queue, so consider the following points:
    - You cannot use a job ID that duplicates with the batch application reserved for deletion.
    - Due to execution of the `cjexecjob` command, if the number of batch application execution requests exceeds the number that you can register in a schedule queue, the KDJE55060-E message is output and the batch server ends abnormally.
- If you execute the `cjkilljob` command for a standby batch application, the `cjexecjob` command does not end until the batch application is taken out from the schedule queue.
- If no batch server exists, when you execute the `cjexecjob`, `cjkilljob`, or `cjlistjob` command, a message is output and the command ends abnormally. An output message varies according to the specification of the `batch.ctm.enabled` key in `usrconf.cfg` (option definition file for batch applications).
    - If `true` is specified,

      The KDJE55010-E or KDJE55046-E message is output.
    - If `false` is specified,

      The KDJE55010-E message is output.
- When executing the `cjexecjob` command, the command might end abnormally depending on the specification in the Easy Setup definition file and `usrconf.cfg` (option definition file for batch applications).
    - If `true` is specified in the `ejbserver.ctm.enabled` parameter of the Easy Setup definition file, and if `false` is specified using the `batch.ctm.enabled` key in `usrconf.cfg` (option definition file for batch applications), the KDJE55067-E message is output, and the command ends abnormally.
    - If `false` is specified in the `ejbserver.ctm.enabled` parameter of the Easy Setup definition file, and if `true` is specified using the `batch.ctm.enabled` key in `usrconf.cfg` (option definition file for batch applications), the KDJE55046-E message is output, and the command ends abnormally.
- When executing the `cjlistjob` command, if `false` is specified in the `ejbserver.ctm.enabled` parameter of the Easy Setup definition file, and if `true` is specified using the `batch.ctm.enabled` key in `usrconf.cfg` (option definition file for batch applications), the command is not received on the batch server. In such cases, the batch application information is not output.

4.6 Migrating to the environment using the scheduling functionality

This section describes how to migrate from an environment where the scheduling functionality is not used. When migrating an execution environment of batch applications, from an environment that does not use the scheduling functionality to an environment that uses the scheduling functionality, you cannot use the environment in use, as it is.

In the environment which is in use, you must edit the definition files. The following table describes the files, for which you need to edit the settings, when migrating the environment.

Table 4–5: Files for which you need to edit the settings, when migrating the environment

File	Main key to be edited	Settings	Required or optional
`usrconf.properties` (User property file for batch servers)	`ejbserver.ctm.enabled`	`true`	Required
	`vbroker.agent.enableLocator`	`true`#	Optional
	`ejbserver.batch.schedule.group.name`	A schedule group name	Optional
	`ejbserver.batch.queue.length`	The length of the created schedule queue	Optional
`usrconf.cfg` (option definition file for batch applications)	`batch.ctm.enabled`	`true`	Required
	`batch.schedule.group.name`	A schedule group name	Optional
	`batch.request.timeout`	A timeout between a batch execution command and a batch sever, and a timeout between a batch forced stop command or a batch list display command and CTM	Optional
	`batch.vbroker.agent.port`	A port number being used by Smart Agent	Optional

Legend:

Required: You must specify

Optional: Specify as and when required

Note: This section describes the main keys to be edited when migrating to an environment that uses the scheduling functionality. For details on the `usrconf.properties` file (user property file for batch servers) and the keys, see *3.2.2 usrconf.properties (User property file for batch servers)* in the *uCosminexus Application Server Definition Reference Guide*.

For details on the `usrconf.cfg` file (option definition file for batch applications) and the keys, see *3.2.5 usrconf.cfg (Option definition file for batch applications)* in the *uCosminexus Application Server Definition Reference Guide*.

#: `false` is specified by default. However, when integrating with CTM, `true` is specified automatically.

For details on the parameters to be edited in respective files, see the *uCosminexus Application Server Definition Reference Guide*.

4.7 Settings of the execution environment

You are required to perform the following settings for using the scheduling functionality:

- Batch servers
- CTM
- Commands to be used with batch applications

This section describes the respective settings. Note that you also need to specify the definition of the batch application execution functionality for using the scheduling functionality. For details on the definition of the batch application execution functionality, see *2.3.10 Settings in the execution environment (batch server settings)*.

4.7.1 Settings of batch servers

You execute the batch server settings in the Easy Setup definition file. You specify the definitions of the scheduling functionality in the `<configuration>` tag of logical J2EE servers (`j2ee-server`) in the Easy Setup definition file.

The following table describes the definitions of the scheduling functionality in the Easy Setup definition file.

Table 4–6: Definitions of scheduling functionality in Easy Setup definition file

Item	Parameter to be specified	Settings	Required or optional
Settings for using the scheduling functionality	`ejbserver.ctm.enabled`	Specify whether the scheduling functionality is to be used. `true` is specified by default. If you specify `ctm-tier` in the `<tier-type>` tag, `true` is automatically specified when building a system.	Optional
Settings for using Smart Agent	`vbroker.agent.enableLocator`	Specify that Smart Agent is to be used. `false` is specified by default. However, `true` is automatically specified when integrating with CTM. Therefore, you need not change the parameter value to `true`.	Optional
Settings for the schedule group name	`ejbserver.batch.schedule.group.name`	Specify the schedule group name of a batch server group managed by CTM. `JOBGROUP` is specified by default. CTM schedules the execution of batch applications for respective schedule groups. If you divide a schedule queue by using multiple schedule groups, specify the schedule group names for respective batch servers.	Optional
Settings for the length of a schedule queue	`ejbserver.batch.queue.length`	Specify the length of a schedule queue that is created by CTM. `50` is specified by default.	Optional

Legend:

Optional: Specify as and when required

Note: This section describes the main parameters to be specified when using the scheduling functionality. When using the scheduling functionality, you can also optionally specify the following parameters starting with `ejbserver.ctm`:

- `ejbserver.ctm.ActivateTimeOut`
- `ejbserver.ctm.CTMDomain`
- `ejbserver.ctm.CTMID`
- `ejbserver.ctm.CTMMyHost`
- `ejbserver.ctm.DeactivateTimeOut`

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

4.7.2 Settings of CTM

You execute the CTM settings with the Easy Setup definition file. You specify the definition of the scheduling functionality in the `<configuration>` tag of the logical CTM (`componenttransaction-monitor`) in the Easy Setup definition file. The following parameters are to be specified. Specify these parameters without fail.

- `ctm.Agent`
  When using the scheduling functionality, you use the ORB gateway functionality of the CTM regulator. Specify the value of parameter as `1` without fail.

For details on the Easy Setup definition file and parameters, see *4.3 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

4.7.3 Settings for the commands to be used with batch applications

You execute the settings of the commands to be used with batch applications, in `usrconf.cfg` (option definition file for batch applications). You specify command options in `usrconf.cfg` for definitions of the scheduling functionality.

The following table describes the definitions of the scheduling functionality in `usrconf.cfg`.

Table 4–7: Definitions of the scheduling functionality in usrconf.cfg

Item	Key to be specified	Settings	Required or optional
Settings for using the scheduling functionality	`batch.ctm.enabled`	Specify whether the scheduling functionality is to be used. Specify the parameter value as `true` without fail.	Required
Settings for the schedule group name	`batch.schedule.group.name`	Specify the schedule group name of a batch server group managed by CTM. `JOBGROUP` is specified by default. CTM schedules the execution of batch applications for respective schedule groups.	Optional
Setting the maximum time for connecting to CTM	`batch.request.timeout`	Specify a timeout between the batch execution command and a batch sever, and a timeout between the batch forced stop command or the batch list display command and CTM. `0` (no timeout) is specified by default.	Optional
Settings for a port number used by Smart Agent	`batch.vbroker.agent.port`	Specify a port number used by Smart Agent. `14000` is specified by default.	Optional

Legend:
  Required: You must specify
  Optional: Specify as and when required

Note: This section describes the main keys to be specified when using the scheduling functionality. For details on `usrconf.cfg` (option definition file for batch applications) and the keys, see *3.2.5 usrconf.cfg (Option definition file for batch applications)* in the *uCosminexus Application Server Definition Reference Guide*.

4.8 Points to be considered when using the scheduling functionality

You consider the following points when using the scheduling functionality:

- With the CTM daemon used for the scheduling functionality, do not perform load balancing of the requests from clients for J2EE servers.
- Do not perform load balancing of requests for batch servers between multiple CTM daemons. Specify different schedule group names on the batch servers connected to multiple CTM daemons.

  If you execute the load balancing of requests for batch servers between multiple CTM daemons, though the batch application execution requests are accepted, the following problems might occur:

  - You cannot view the list of batch application information (you cannot acquire the batch application information).
  - Forced stopping of batch applications fails.
- If you execute the batch forced stop command while passing a batch application execution request from a schedule queue to a batch server, the KDJE55016-W message is output and you cannot forcefully stop the batch application. In such cases, you execute the batch list display command and confirm the status of the batch application. If the batch application state is `running`, re-execute the batch forced stop command.
- If a timeout occurs between CTM and a batch server, the KDJE55061-E message is output. In such cases, CTM does not manage the batch application execution requests and the running batch applications. In such cases, when performing the list display or forced stopping for the running batch applications, execute commands by specifying batch server names. Execute the batch list display command after changing the settings for not using the scheduling functionality. To specify the settings for not using the scheduling functionality, specify `false` in the `batch.ctm.enabled` key in `usrconf.cfg` (option definition file for batch applications).

  You can identify the batch server names specified in commands, in the message (KDJE55066-I) for the batch execution commands.
- Before a batch application starts on a batch server, if the batch execution command is terminated using **Ctrl**+**C** or a timeout, the KDJE55007-E message is output to a message log, and an attempt to start the batch application fails. In such cases, the KDJE40062-E message might be output to the standard error output.

5 Inheriting Session Information Between J2EE Servers

This chapter describes the overview, prerequisites, and memory estimation of the session failover functionality, which is functionality for inheriting session information between J2EE servers.

5.1 Organization of this chapter

Use the session failover functionality for inheriting session information between J2EE servers. This section gives an overview of the session failover functionality and types of the session failover functionality.

The following table describes the organization of this chapter.

Table 5–1: Organization of this chapter (the session failover functionality)

Category	Title	Reference location
Description	Overview of the session failover functionality	*5.2*
	Session management by using a global session	*5.3*
	Prerequisites	*5.4*
	Database session failover functionality	*5.5*
	A functionality that you can set when using the session failover functionality	*5.6*
	A functionality executed when using the session failover functionality	*5.7*
	Estimating the memory	*5.8*
Notes	Notes	*5.9*

Note: There is no specific description on the *Implementation*, *setup*, and *operations* for this functionality.

5.2 Overview of the session failover functionality

The session failover functionality inherits objects registered in the `HttpSession` object on a J2EE server when a software failure, hardware failure, or network failure occurs on a J2EE sever or a Web server.

If you use the session failover functionality and if a failure occurs on a specific J2EE server in a system, you can continue the operations on another J2EE server by inheriting session information before failure. Thus, you can improve the availability of the system.

The following subsections describe the benefits of using the session failover functionality and the types of functionality.

5.2.1 Benefits of using the session failover functionality

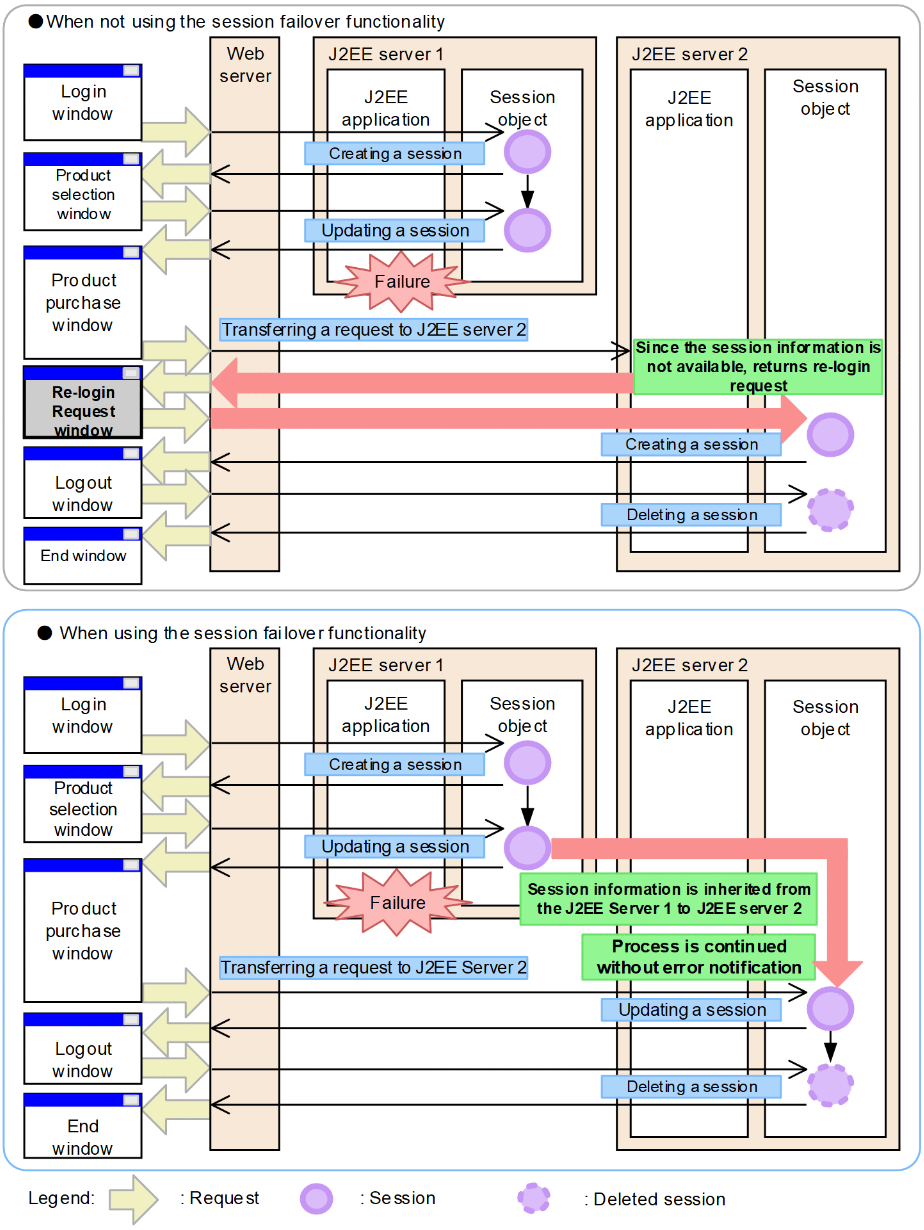
The `HttpSession` object is retained in the memory of a J2EE server. The `HttpSession` object is lost if a failure occurs on the J2EE server. In the case of a system configured from multiple J2EE servers, if a failure occurs on one J2EE server, requests are transferred to another J2EE server. However, because the `HttpSession` object is lost, the information registered in the `HttpSession` object (*session information*) is not inherited. As a result, the session is treated as a new session in a J2EE application on the J2EE server to which the requests are transferred. For example, if a failure occurs on a window after user authentication processing, you must login again.

If you use the session failover functionality, you can manage the session information and if a failure occurs on a J2EE server, you can pass the managed information to another J2EE server. As a result, even when a failure occurs on a J2EE server and requests are transferred to another J2EE server, you can continue operations in the state before failure.

You can also inherit the login state on other J2EE servers by using the session failover functionality even if you are using integrated user management.

The following figure shows the flow of processing when using and not using the session failover functionality.

Figure 5–1: Flow of processing when using and not using the session failover functionality



If a failure occurs on a server when you are not using the session failover functionality, you must login again because the session information is lost.

If you use the session failover functionality, session information is inherited between servers, and hence you can continue the processing without noticing a failure on a server when performing user operations in a browser.

5.2.2 Types of session failover functionality

There are the following types of the session failover functionality depending on the storage location of session information:

- The database session failover functionality

  This functionality stores the session information in a database and manages the information.

For details on the overview of the database session failover functionality, see *5.5.1 Overview of the database session failover functionality*.

For details on the application procedure, flow of processing, and settings, see *6. Database session failover functionality*.

5.3 Session management using a global session

This section describes the global session information managed by using the session failover functionality. This section also describes the conditions and precautions for the attributes of HTTP sessions, which are inherited as the global session information.

5.3.1 Global session information

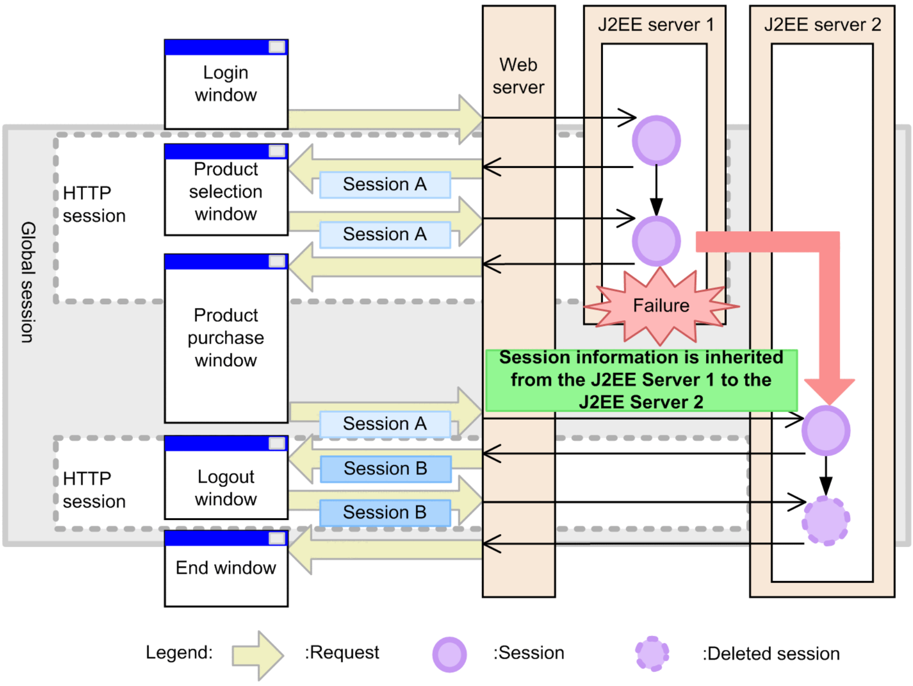
With the session failover functionality, a J2EE server can inherit the information of the objects registered in the HttpSession object on another J2EE server.

Invoke a session that you can inherit and use between multiple J2EE servers as a *global session*. An HTTP session is lost if a failure occurs on the J2EE server, which handles that session. However, a global session is not lost in this situation because the session is managed by a process that is not the J2EE server.

If you are using a global session, the information of the `HttpSession` object that is inherited on another J2EE server, is called the *global session information*.

For details on the scope of the HTTP session and global session, see the following figure.

Figure 5–2: Scope of HTTP session and global session



With the session failover functionality, because the global session information is inherited when a failure occurs on a J2EE server, you can continue operations in the state before the failure occurrence without reporting an error to the user.

5.3.2 Information included in the global session information

If you use the database session failover functionality, the global session information is stored in records of the session information storage table created in a database. When the global session information is stored, one record is assigned to each HTTP session.

The information described in the following table is included in the global session information.

Table 5–2: Information included in the global session information

No.	Replication target	Explanation
1	Session ID	A session ID that manages the global session information.
2	HTTP session attribute information	Information that converts the objects of an attribute name and attribute value into serialized byte arrays for the attributes registered in an HTTP session and all the associated attributes.
3	Creation time of an HTTP session	The time of creating an HTTP session. When inheriting a global session, the creation time of an HTTP session before inheritance is used as is.
4	Expiration date of an HTTP session	An expiration date set in an HTTP session.
5	Last access time	The time when requests using an HTTP session are sent for the last time.
6	Identifier of J2EE server which owns HTTP session	A server ID of the J2EE server that creates or inherites an HTTP session.

Reference note

- Database tables used in the database session failover functionality include an application information table that stores the setting information of Web applications, a session information storage table that stores the global session information, and a blank record information table that manages unused records in the session information storage table.

5.3.3 HTTP session attributes that are inherited as global session information

This subsection describes the following items related to the HTTP session attributes that can be inherited when a failure occurs.

- Conditions for the HTTP session attributes that can be inherited
- Objects supported as target for inheriting
- Can or cannot inherit the session information depending on object contents
- Notes on serialize processing when inheriting HTTP session attributes
- Notes on deserialize processing when inheriting HTTP session attributes
- Notes for when HTTP session attributes are inherited

(1) Conditions for the HTTP session attributes that can be inherited

In the session failover functionality, serialization of objects occurs in update processing of the global session information and deserialization of objects occurs in inherit processing. Hence, attributes to be registered in an HTTP session must satisfy the following condition:

- It is an object of serializable class that has implemented the `java.io.Serializable` interface.

(2) Objects supported as targets for inheriting

With the session failover functionality, the following objects of serializable classes are supported as targets for inheriting:

- Objects of the classes provided by a J2EE application.
- Objects of the classes provided on J2SE.

However, with inheritance processing, it is not checked whether an object of a serializable class, which is registered in an HTTP session, is supported by the session failover functionality.

(3) Conditions for inheriting the session information depending on object contents

The following table describes whether you can or cannot inherit the session information depending on the contents of objects registered in an HTTP session.

Table 5–3: Conditions for inheriting the session information depending on the contents of objects registered in an HTTP session

No.	Contents of objects registered in an HTTP session		Can or cannot inherit the session information	Storing global session information
	Implementation status of the java.io.Serializable interface	Serialization successful/failed		
1	The `java.io.Serializable` interface implemented	Serialization successful	Can be inherited.	Information after serialization is stored in a database.
2		Serialization failed	Cannot be inherited because HTTP sessions containing attributes, which failed in serialization, are not targeted for inheriting a global session.	The KDJE34318-E or KDJE34411-E message is output and the global session information is not stored in a database. After completing request processing the next time, the global session information is stored in a database when objects registered in an HTTP session become serializable.
3	The `java.io.Serializable` interface not implemented	(Cannot be serialized)	Cannot be inherited because attributes that cannot be serialized cannot be targeted for inheriting the global session.	If there are objects that cannot be serialized, the KDJE34317-W or KDJE34410-W message is output and the global session information that is created with the attributes excluding attributes that cannot be serialized, is stored in a database.

(4) Notes on serialize processing when inheriting HTTP session attributes

Notes on serialize processing are as follows:

(a) Impact of serialize processing on performance

The serialize processing is executed not only for the objects targeted for inheriting but also for all the objects that are referenced from the objects targeted for inheriting. Hence, if you register a class containing information, which need not be inherited, in an HTTP session, performance might deteriorate.

(b) When the java.lang.OutOfMemoryError error occurs

In the serialize processing, data after serialization is temporarily created exceeding the number of `HttpSession` objects set in the application. As a result, if you register huge objects in an HTTP session, the `java.lang.OutOfMemoryError` error might occur while creating the global session information.

(c) When serialization fails and its measures

In the following cases, the `KDJE34317-W`, `KDJE34318-E`, `KDJE34410-W`, or `KDJE34411-E` message is output and serialization fails.

- If objects referenced from the objects registered in an HTTP session (objects of serializable classes) include the objects for which classes other than serializable classes are implemented.
- If the `writeObject` (`java.io.OutputStream out`) method is implemented in objects and if an exception occurs when serializing.

If serialization fails, processing for updating and inheriting the global session information is not executed. To execute the processing, you must take one of the following actions:

- Cancel the registration of objects that failed in serialization, in an HTTP session.
- Change the objects that failed in serialization and eliminate the cause of failure.

(5) Notes on deserialize processing when inheriting HTTP session attributes

Deserialization fails in the following cases:

- If you add changes that cause failure in deserialization in a Web application and if the Web application is different than in the case of serialization.
- If the `readObject` (`java.io.OutputStream out`) method is implemented in objects and if an exception occurs when deserializing.

If deserialization of session information fails when receiving a request or in the processing of inheriting global session information when starting a Web application, global session information and session information is deleted, and `KDJE34326-E` or `KDJE34413-E` is output. Because inheriting of the session fails, the request is processed in the absence of an HTTP session.

(6) Notes for when HTTP session attributes are inherited

In a configuration in which the integrity mode is not used and multiple requests are executed for one session, thread-unsafe objects must not be stored in the session. (Contention with Cosminexus-side processing cannot be avoided even if a thread-safe implementation is achieved by using, for example, the `synchronized` clause in the user program.)

5.4 Prerequisites

This section describes the prerequisites for using the session failover functionality.

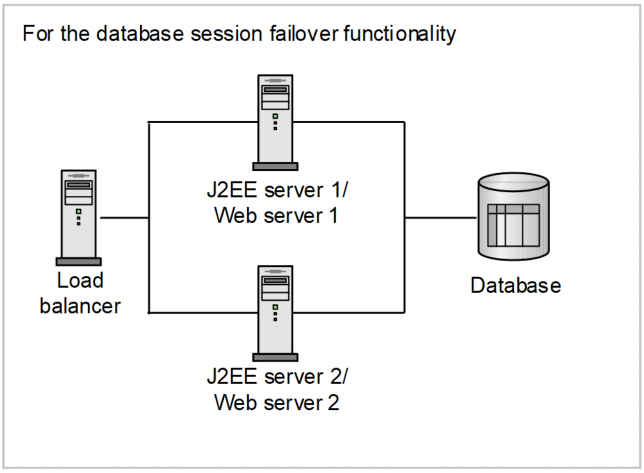
5.4.1 Prerequisite configuration

If you want to use the session failover functionality, a system configuration that uses a load balancer and distributes requests on multiple J2EE servers, is a prerequisite. Moreover, it is necessary to deploy a database for storing HTTP session information that is created on each J2EE server.

The database session failover functionality does not support Oracle Database connections based on Oracle RAC.

The following figure shows the prerequisite configuration for using the database session failover functionality.

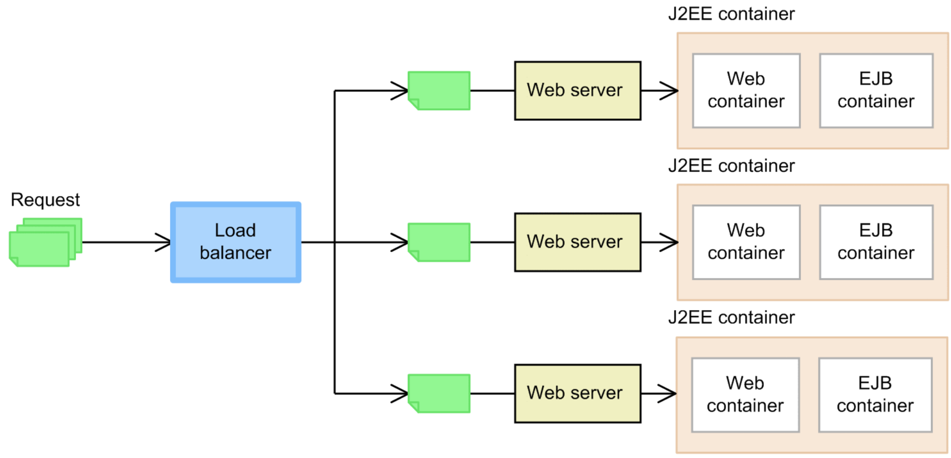
Figure 5–3: Prerequisite configuration for the session failover functionality



- Load balancer
  For using the session failover functionality, use of a load balancer is a prerequisite.

> **Reference note**
>
> **Distributing a request with a load balancer**
>
> Requests are distributed with a *load balancer*. This distributes the load, achieving stable operations of the system and improving processing performance.
>
> The load balancing performed by using a load balancer has the advantage that the load related to the load balancing processing is not applied to the Web server and J2EE server. Methods for the request distribution vary according to the load balancer.
>
> The following figure shows an example of distributing requests by using a load balancer.

Figure 5–4: Example of request distribution by using a load balancer



- The J2EE server or Web server

  For using the session failover functionality, arrange one or more J2EE servers and Web servers in one system. We recommend that you arrange two or more servers in the preparation for a J2EE server failure.

- Database

  When using the database session failover functionality, a database is required as a storage location of session information. The following table describes mapping of the databases, which you can use as a storage destination of session information, JDBC drivers, and resource adapters.

Table 5–4: Mapping of databases, JDBC drivers, and resource adapters which you can use

Database	JDBC driver	Resource adapter#
HiRDB	HiRDB Type4 JDBC Driver	`DBConnector_HiRDB_Type4_CP.rar`
Oracle	Oracle JDBC Thin Driver	`DBConnector_Oracle_CP.rar`

DB Connector is the resource adapter used in the database session failover functionality. For details on the settings required for DB Connector used in the database session failover functionality, see *6.6.4 DB Connector settings*.

For details on the system configuration for using the database session failover functionality, see *3.10.1 Configuration using a database (database session failover functionality)* in the *uCosminexus Application Server System Design Guide*.

If the configuration that uses the database session failover functionality satisfies the conditions described here, you need not redesign from the system configuration. You can use the database session failover functionality if you implement the functionality settings and perform parameter tuning.

5.4.2 Prerequisite settings

This subsection describes the prerequisite settings for using the session failover functionality.

(1) Common prerequisite settings of the session failover functionality

The following settings are required when using the database session failover functionality.

- **Adding a server ID to a session ID by using the server ID addition functionality of HttpSession**

This functionality adds a server ID to a session ID of HTTPSession. This functionality must be enabled if you want to use the database session failover functionality (disabling integrity mode). Set different server IDs for each replicated J2EE server.

If you disable the server ID addition functionality of HttpSession, the KDJE34371-E or KDJE34404-E error message is output to the message log when starting the Web application, and the Web application fails to start. If you do not set a different server ID for each replicated J2EE server, global session information might be inherited on an unintended J2EE server and the integrity of global session information might be lost.

For details on the functionality of adding server ID to session ID of HttpSession, see *2.7.6 Adding server ID to session ID and Cookie* in the *uCosminexus Application Server Web Container Functionality Guide*.

- **Setting sticky of an HTTP session**

  To use the session failover functionality in the environment used by the load balancer, you must set sticky for the HTTP session.

  If you do not set sticky for the HTTP session, distribution destination of the requests that retain HTTP session is not fixed. As a result, an HTTP session is inherited every time you receive a request that retains an HTTP session, and this might result in the performance deterioration.

- **Setting the host time**

  For using the session failover functionality, set the same time on each host, on which the J2EE servers in the system run.

  Information such as the creation time and last access time of an HTTP session is included in the session information to be stored in a database. If the time set on each host is different, the information that is different from the setting time of local host is included in the session information. As a result, if you inherit a session, problems might occur when controlling the HTTP session.

(2) Prerequisite settings of the database session failover functionality

The following settings are required when using the database session failover functionality.

- **Deleting invalid session IDs retained by the Web client**

  This functionality deletes the information of HTTP Cookie, which is retained in the Web client when disabling an HTTP session, and inhibits sending of a session ID to disabled HTTP sessions. For using the database session failover functionality, you must enable this functionality.

  If you have disabled deletion of HTTP Cookie that indicates the session ID of an HTTP session, the `KDJE34339-E` error message is output to the message log when starting a Web application, and the Web application fails to start. For details on deleting HTTP Cookie that indicates the session ID of an HTTP session, see *2.7.4 Deleting invalid session IDs retained by a Web client* in the *uCosminexus Application Server Web Container Functionality Guide*.

- **Specifying the upper limit of the number of HttpSession objects**

  This functionality sets the upper limit of the number of a valid `HttpSession` object. Set this functionality when enabling the integrity mode.

  If you have set to cancel the Web application start process in the case of failure in the negotiation processing that is executed when an application starts a, you must set a valid value (1 or above) as an upper limit If you have not specified the upper limit of the number of `HttpSession` objects, the `KDJE34303-E` error message is output to the message log when starting an application, and the application fails to start.

  However, if you have set to continue the Web application start process in the case of failure in the negotiation process, setting the upper limit of the number of `HttpSession` objects is optional. You can also specify -1 (unlimited) as the upper limit. If you set -1 (unlimited) as the upper limit of the number of `HTTPSession` objects or if you specify a value greater than the number of records in the session information storage table of the database, the operation performed when the number of `HttpSession` objects exceeds the number of records in the session information storage table is as follows:

When an integrity mode is disabled (optional)

The corresponding HTTP sessions reduce and request processing continues.

When an integrity mode is enabled

The `KDJE34380-E` error message is output to the message log and corresponding HTTP session is not created.

For details on settings of the upper limit of the number of `HttpSession` objects, see *2.7.5 Setting upper limit of the number of HttpSession objects* in the *uCosminexus Application Server Web Container Functionality Guide*.

For details on negotiation processing, see *6.4.1 Processing when starting an application*.

For details on integrity mode, see *5.5.1(4) Operation mode of the database session failover functionality*.

For details on reduction of HTTP sessions when an integrity mode is disabled, see *5.7.3 Reducing an HTTP session*.

- **Setting default pending queues and pending queues in the Web application unit, and pending queues in the URL group unit**

  If the functionality for controlling the number of concurrently executed threads in an Web application unit is enabled, and if vacancies in the default pending queue, pending queues in the Web application unit, and pending queues in the URL group unit become insufficient, specify whether the `503` error is to be returned to the client, in the `webserver.dbsfo.thread_control_queue.enabled` parameter in the Easy Setup definition file. Note that by default the `503` error is returned to the client.

  If you set not to return the `503` error to client, set a sufficiently large value in the pending queue size.

  If you set to return the `503` error to client, do not perform the following HTTP session updates on the error page specified in `web.xml`.

  - Creating an HTTP session

    If the Web application creates an HTTP session, the `com.hitachi.software.web.dbsfo.SessionOperationException` exception is thrown at the invocation source of the `getSession` method in the `javax.servlet.http.HttpServletRequest` interface and an HTTP session is not created.

  - Changing the expiration date of an HTTP session (invoking the `setMaxInactiveInterval` method in the `javax.servlet.http.HttpSession` interface)

    If the Web application changes the expiration date of an HTTP session, the expiration date of the global session on the database does not change. If the global session is inherited, the expiration date returns to the state before change.

  - Changing the attribute information of the HTTP session

    If the Web application changes the HTTP session attribute information, the global session information on the database does not change. If the global session is inherited, the attribute information returns to the state before change.

  - Disabling an HTTP session (invoking the `invalidate` method in the `javax.servlet.http.HttpSession` interface)

    If theWweb application invokes the `invalidate` method in the `javax.servlet.http.HttpSession` interface, the `com.hitachi.software.web.dbsfo.SessionOperationException` exception is thrown.

- **The user-specified namespace functionality**

  When using the database session failover functionality, the system considers that the look up of J2EE resources in optional names that are given by using the user-specified name space functionality, is already performed.

  Hence, if you have specified the following parameter in the properties of the J2EE server and are using the round-robin search functionality, you cannot use the database session failover functionality.

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

If you have specified this parameter, the KDJE34305-E error message is output to the message log when starting a Web application, and the Web application fails to start.

If round-robin search is required in the J2EE application to be operated on the J2EE server, do not specify the classes delegated to implement the InitialContextFactory, in the properties of the J2EE server. You must specify the classes in an argument when generating InitialContext for each application. For details on the round-robin search functionality, see*2.7 Searching CORBA naming services by using round-robin policy* in the *uCosminexus Application Server Common Container Functionality Guide*.

5.5 Database session failover functionality

The database session failover functionality are used for inheriting session information between J2EE servers. This section describes an overview of the database session failover functionality.

5.5.1 Overview of the database session failover functionality

The database session failover functionality manages session information in a database and inherits session information between J2EE servers when a failure occurs. When a failure occurs, you can re-create the session based on the session information stored in the database and can continue the normal operations.

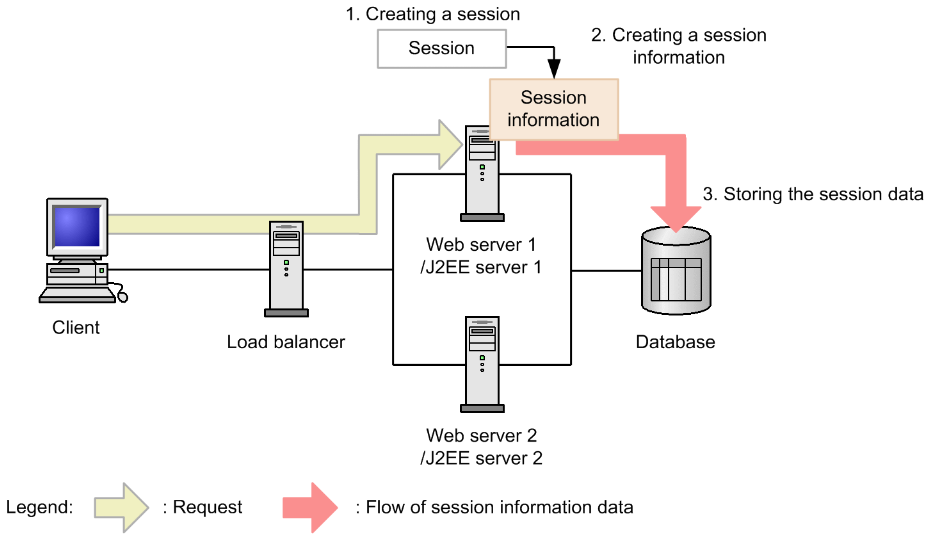
This subsection describes an overview and the operation mode of the processing of the database session failover functionality.

(1) Procedure for storing the session information

If you use the database session failover functionality and a session creation processing is generated by a request, the processing is extended and the session information is stored in a database.

The following figure shows the flow of storing session information.

Figure 5–5: Flow of storing session information (the database session failover functionality)



No. corresponds to the numbers in the figure.

1. If the Web server receives a request requiring the creation of a session, from the client, a session is created on the J2EE server.
2. The session information is created for the session.
3. The session information is stored in the database.

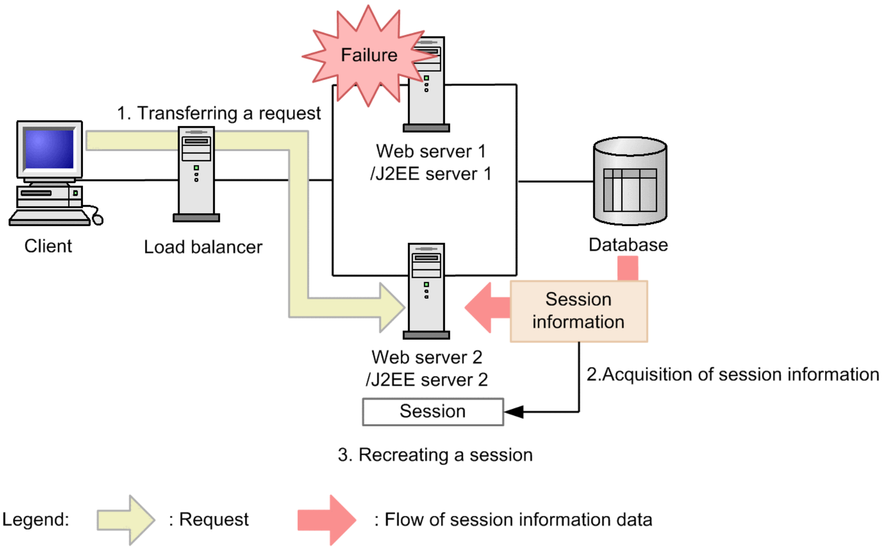
If a failure occurs in the Web server 1 or J2EE server 1, the session information stored in the database is inherited by the web server 2 or J2EE server 2, and you can continue the operations in the state before failure.

(2) Flow of processing when a failure occurs on a Web server or a J2EE server

If a failure occurs on a Web server or a J2EE server, you can re-create a session on another J2EE server based on the session information stored in the database and continue the normal operations.

The following figure shows the flow of processing when a failure occurs on a Web server or a J2EE server:

Figure 5–6: Flow of processing when a failure occurs on a Web server or a J2EE server (the database session failover functionality)



1. If a failure occurs on the Web server 1, the load balancer transfers the request to the Web server 2.
2. Because the session associated with the request does not exist when processing the request on the J2EE server at the transfer destination, it inherits the session information from the database.
3. The session is re-created.

The session is successfully inherited and you can continue operations in the state before failure.

When you restart the J2EE server 1 and Web server 1 recovering from failure, the requests are again sent to the Web server 1.

(3) Flow of processing when a failure occurs in a database

If a failure occurs in a database, you can continue operations by operating only the session information on the J2EE server. When the database recovers from the failure and you can access the database in session operations after that, the functionality updates the database with the session information operated on the J2EE server.

As a result, the client can continue operations without recognizing the database failure.

(4) Operation mode of the database session failover functionality

If multiple requests with the same session ID for the global session information that is stored in the database are concurrently sent, you can concurrently process multiple requests by default. Thus, you can control the degradation of the processing performance caused by the use of the database session failover functionality.

However, a prerequisite of this operation is that processing such as concurrently updating global session information of the same session ID from multiple replicated J2EE servers should not occur. If global session information with the same session ID is updated from multiple J2EE servers, consistency of global session information might be lost. You must enable a mode for maintaining consistency of global session information in the systems in which such cases cannot be allowed.

The mode that maintains consistency of global session information is called *Integrity mode*. If you enable this mode, a lock is set to the database whenever you update a global session. If multiple requests with the same session ID are concurrently sent, the requests are serially processed and the global session information does not become inconsistent. However, request processing performance might be affected because multiple requests cannot be concurrently executed and the lock set up and release processing occurs whenever global session information is stored.

As a result, when using the database session failover functionality, you need to examine the mode to perform the operations depending on the purpose and characteristics of the system.

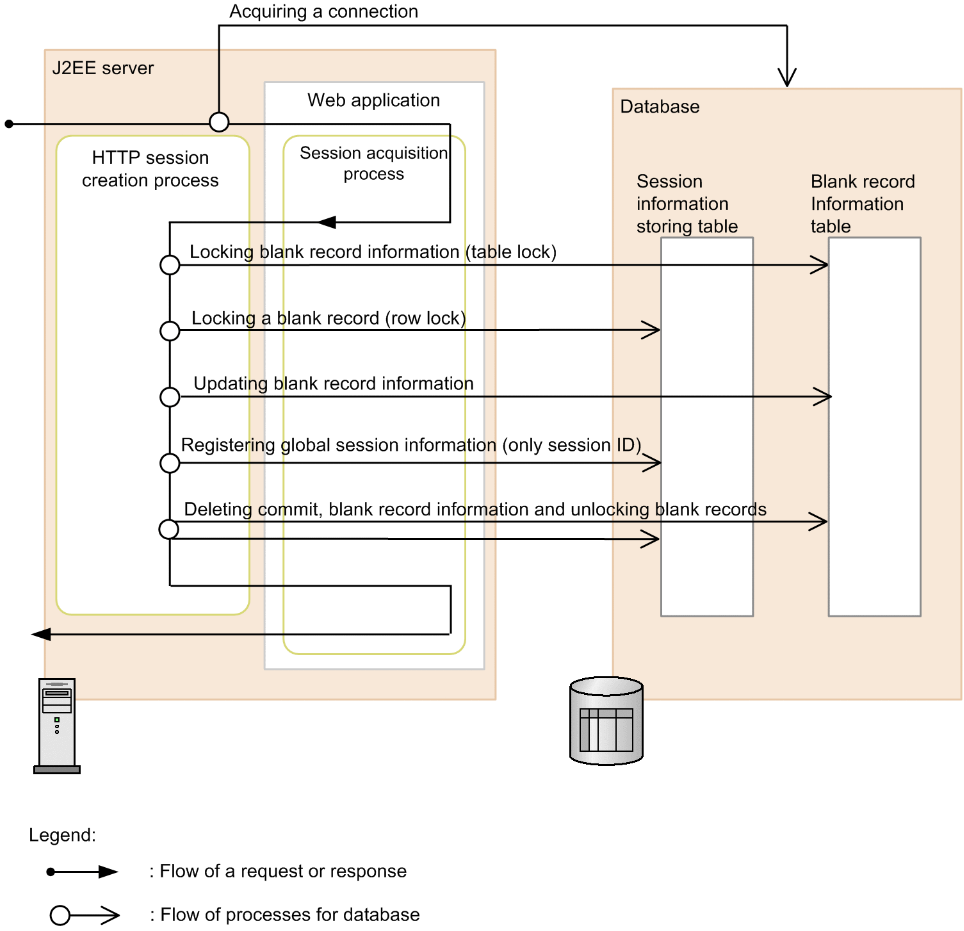
The following table describes the main differences depending on enabling or disabling the integrity mode.

Table 5–5: Main differences depending on enabling or disabling the integrity mode

Items to be compared	Integrity mode	
	Disabled	Enabled
Characteristics of the appropriate system	Suitable for a system in which performance is highly important.	Suitable for a system in which assured inheriting of session information is required even if performance reduces.
Request processing performance	Performance is excellent because you can concurrently process multiple requests with the same session ID.	Performance degrades because it is necessary to serially process the requests.
Integrity of global session information	Integrity is not maintained if you concurrently update global session information with the same session ID.	Maintains the integrity.
Behavior when a failure occurs in the database	Uses session information on the J2EE server and continues the processing (reduced operations of the database session failover functionality).	Outputs an error message and stops the processing.

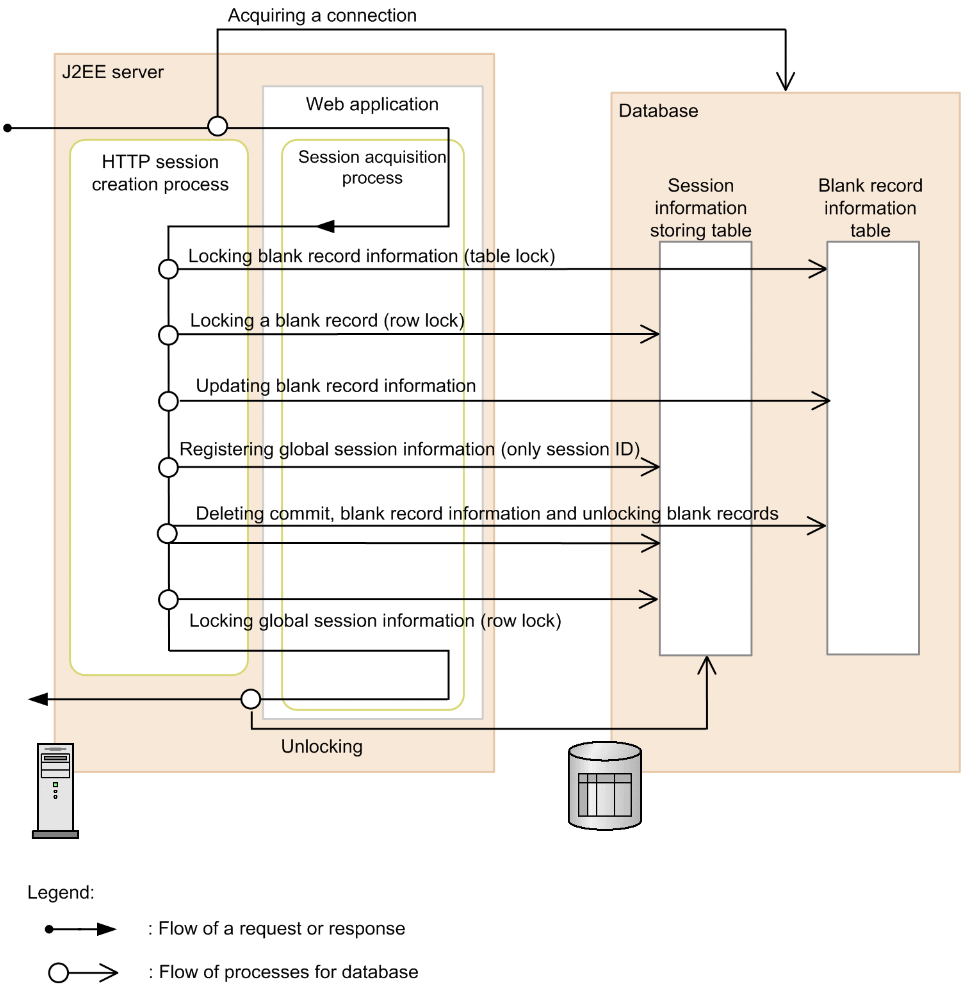
The following figure shows the flow of request processing in each mode:

Figure 5–7: Flow of request processing when the integrity mode is disabled (default setting)



If integrity mode is disabled, a database lock is acquired and released when creating global session information in the database by extending HTTP session creation processing. However, a lock is not acquired with session acquisition processing after you commit once. The database lock acquisition processing and release processing are not executed with the subsequent update processing of global session information.

Figure 5–8: Flow of request processing when an integrity mode is enabled



If an integrity mode is enabled, a database lock is acquired and released when creating global session information in the database by extending the HTTP session creation processing. In addition, a lock is acquired again in session acquisition processing after you commit once. Thus, even if a failure occurs on a J2EE server or in a database during execution of a Web application after creating the HTTP session, inconsistency does not occur in the database processing. With the subsequent update processing of global session information, the processing is implemented for acquiring a database lock whenever updating the global session informaiton and unlocking after the updation is complete.

For details on the operations when locking global session information, see *6.4.5(1) Result of invoking lock acquisition processing when acquiring a lock*.

The functionality that you can use vary depending on whether the setting of an integrity mode is enabled or disabled.

5.6 Functionality that you can set when using the session failover functionality

This section describes the following functionality that you can set when using the session failover functionality. You can use the functionality as and when required.

- Inhibiting session failover
- Defining refer-only requests of an HTTP session#

  # You cannot use the functionality that defines refer-only requests of an HTTP session when you enable the integrity mode of the database session failover functionality.

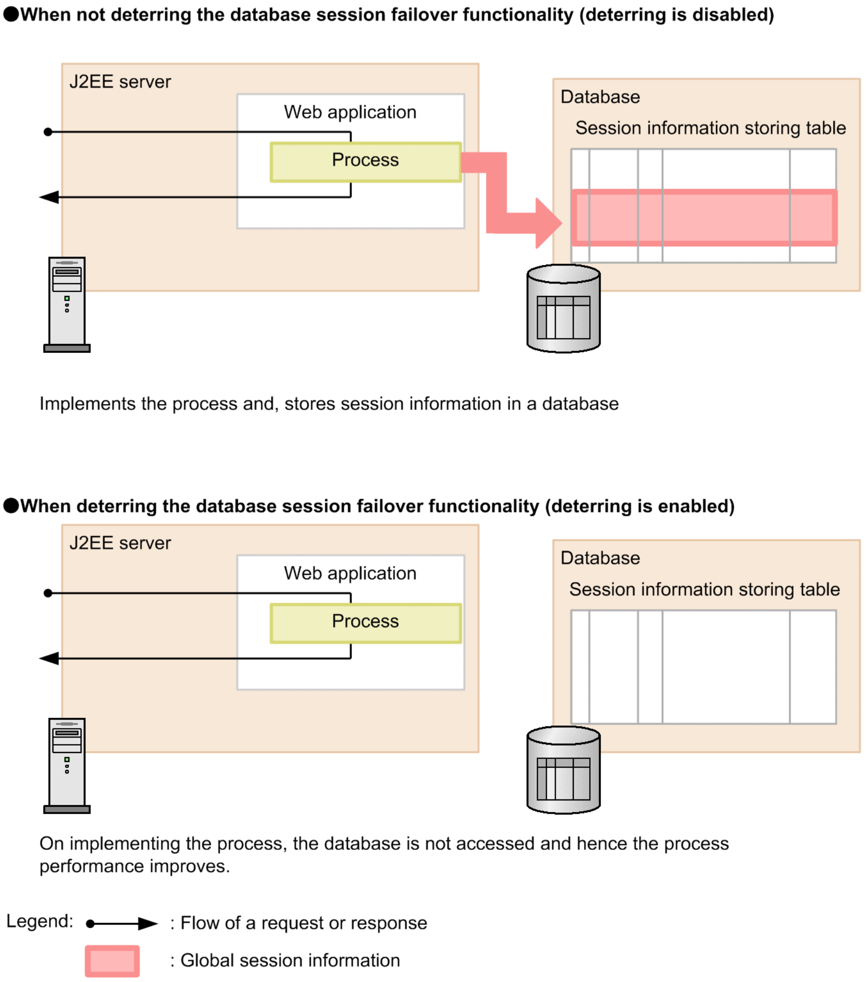
5.6.1 Inhibiting the session failover functionality

If you enable the session failover functionality and receive requests for which an HTTP session is acquired, processing such as accessing a database, and serializing the HTTP session are executed. If the same session ID as the request for which an HTTP session is acquired is sent even for the requests related to the static contents or contents that do not require an HTTP session, the session failover functionality operates and performs unnecessary processing such as accessing a database or an EADs server or serializing the HTTP sessions.

If you set a URl pattern that inhibits the session failover functionality in a URI or an extension, the processing of the session failover functionality for the requests of the set URL patterns is inhibited. Hence, unnecessary processing does not occur and processing performance improves. Thus, if you have set the session failover functionality, the functionality which inhibits the session failover only for the specific URL patterns is called *session failover inhibitionion functionality*.

The following figure shows the differences in the executed processing when a session failover inhibitionion functionality is enabled or disabled with an example of the database session failover functionality.

Figure 5–9: Differences in the processing when the session failover inhibition functionality is enabled or disabled (the database session failover functionality)



You can use the session failover inhibitionion functionality not only for improving performance but also for the following purposes:

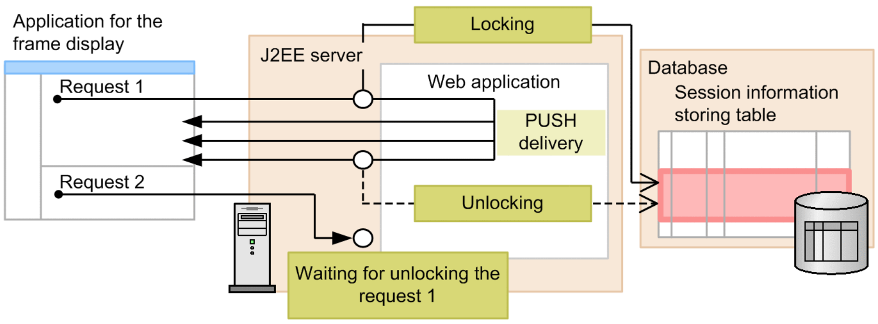
The database session failover functionality when you enable the integrity mode, executes the exclusive processing for requests of the same session ID. For example, if you invoke a servlet or JSP, for which the processing continues for a long time, such as the servlet or JSP that you must make resident for performing the PUSH delivery, from one of the HTML frames, all the requests sent from the same frame are not executed until processing of that servlet or JSP is complete. This happens because all the requests sent from one frame are the requests that send the same session ID.

For preventing such situations, you must inhibit the session failover functionality for particular requests that do not use the HTTP session.

The following figure shows the differences in processing executed when you enable or disable the session failover inhibitionion functionality when using the database session failover functionality by enabling the integrity mode. Note that the request 1 and request 2 in the figure send the same session ID.

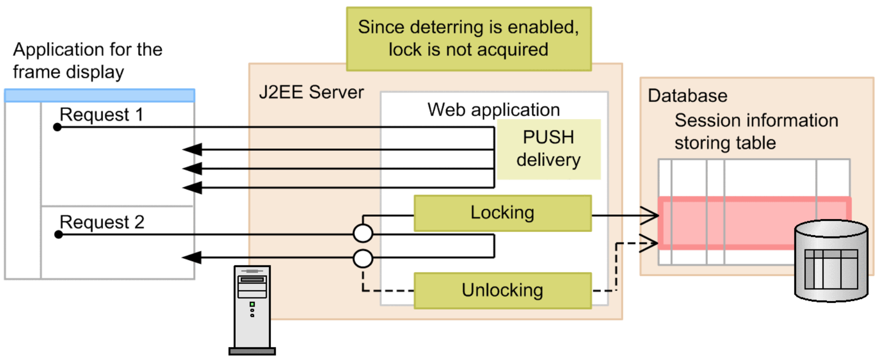
Figure 5–10: Differences in the processing when the session failover inhibitionion functionality is enabled or disabled (the database session failover functionality)

●**When not deterring the database session failover functionality (deterring is disabled)**



Request 1 does not use the HTTP session but, in order to send the acquired session ID, acquire a lock of the global session information before processing the request
For servlet/JSP in which the process does not end a the request 1 is PUSH delivery, as the processing of request 2 is exclusively performed, the processing of request 2 waits for the processing of request 1 to end.

●**When deterring the database session failover functionality (deterring is enabled)**



If you set the URI of request 1 that does not use an HTTP session as the deterring target for the database session failover functionality, you can process the request 2 executed later, without waiting for the locking of the global session information.

Legend: : Flow of requests or responses
: Flow of locking (exclusive acquisition of record)
: Flow of unlocking (exclusive release)
: Global session information

You can set enabling or disabling of the session failover inhibitionion functionality in the J2EE server unit or Web application unit.

- Notes

  This subsection describes the precautions to be taken when using the session failover inhibitionion functionality.

  - If you invoke the `getSession()` method or `getSession(boolean create)` method in the `javax.servlet.http.HttpServletRequest` interface during a request processing for which the session failover inhibitionion functionality has disabled the session failover functionality, the `com.hitachi.software.web.dbsfo.SessionOperationException` exception is thrown. As a result, you cannot apply the session failover inhibition functionality for the requests that invoke this method.

    For details on the `com.hitachi.software.web.dbsfo.SessionOperationException` exception, see *3.1 Exception classes* in the *uCosminexus Application Server API Reference Guide*.

- As the requests for which the session failover inhibitionion functionality has disabled the session failover functionality are not the requests that use an HTTP session, the access time of the HTTP session is not updated. Impacts of the functionality are:

  · The `getLastAccessedTime()` method in the `javax.servlet.http.HttpSession` interface returns the time of executing the request that used the previous HTTP session.

  · If the difference between the current time and the access time of an HTTP session exceeds the timeout time, the HTTP session times out. As a result, after creating an HTTP session, if you continue sending only the requests for which the session failover functionality is disabled, the HTTP session might time out.
- With JSP, an HTTP session is implicitly created by default. As a result, when applying the session failover inhibitionion functionality to JSP for which an HTTP session is not required, you must explicitly use session attributes of the page directive and set such that the HTTP session is not created.
- If you use the `FORM` authentication as a login authentication functionality provided by the Web container, an HTTP session is implicitly created. If you use the `FORM` authentication in the requests for which the session failover inhibitionion functionality has disabled the session failover functionality, you cannot create an HTTP session. As a result, `com.hitachi.software.web.dbsfo.SessionOperationException` exception occurs and authentication is not performed. However, if you have already created a session, exception does not occur and authentication is performed even for a request for which the session failover inhibitionion functionality has disabled the session failover functionality.

5.6.2 Defining refer-only requests of an HTTP session

The functionality for defining refer-only requests in HTTP session sets the URL patterns of the requests that are only to be referenced and not to be updated (*refer-only requests*), and thus deters the serialization of HTTP sessions or access to the database, for requests of those URL patterns.

This functionality can be used if the integrity mode of the database session failover functionality is disabled.

Note that you can use the session failover inhibition functionality in the case of the requests, which not only update but also not refer to the HTTP session. The requests corresponding to both the refer-only requests and the requests targeted for session failover inhibition functionality, processing are executed as the requests targeted for the session failover inhibitionion functionality. For details on the session failover inhibitionion functionality, see *5.6.1 Inhibiting the session failover functionality*.

You can set the functionality of defining refer-only requests of an HTTP session in the J2EE server unit or Web application unit.

For details on the settings in the J2EE server unit in the case of the database session failover functionality, see *6.6.1 J2EE server settings*.

- **Notes**

  This subsection describes the precautions to be taken when defining refer-only requests of an HTTP session.

  - You cannot disable an HTTP session with a refer-only request. If you invoke the `invalidate()` method in the `javax.servlet.http.HttpSession` interface that disables an HTTP session with a refer-only request, `com.hitachi.software.web.dbsfo.SessionOperationException` exception is thrown in a Web application.
  - Even in the case of refer-only requests, for the first request for which an HTTP session does not exist, an HTTP session is created, updated, and deleted. At that time, the global session information in the database is also updated.

At that time, the global session information in the database is also updated. As a result, when inheriting the global session, attribute information of the HTTP session returns to the state before update.

- If you change the expiration date of an HTTP session (invoking the `setMaxInactiveInterval()` method in the `javax.servlet.http.HttpSession` interface) in refer-only request processing, an expiration date of a global session does not change. As a result, when inheriting the global session, the expiration date of the session returns to the state before change.
- If you change the attribute information of an HTTP session in refer-only request processing, global session information does not change. As a result, when inheriting the global session, attribute information of the HTTP session returns to the state before change. Changes refer to the following changes in attribute information:

  · Registering a new attribute information in an HTTP session or replacing the registered session attributes by using the `setAttribute()` method or the `putValue()` method in the `javax.servlet.http.HttpSession` interface.

  · Deleting the attribute information registered in an HTTP session by using the `removeAttribute()` method or the `removeValue()` method in the `javax.servlet.http.HttpSession` interface.

  · Changing the contents of attribute information registered in an HTTP session.

  The following is an example of changing the contents of attribute information of a session:

```
java.util.Hashtable table = (java.util.Hashtable)session.getAttribute("
attr1");
table.put("key1", "value1");
```

  In this example, `session` is a variable that stores `HttpSession` objects. The `java.util.Hashtable` object is registered as attribute information of a session in `HttpSession` objects with the *attr1* name in another request.

5.7 Functionality executed when using a session failover functionality

This section describes the functionality that are automatically executed when using the session failover functionality. The functionality described here are applied when you disable the integrity mode of the database session failover functionality. This functionality are not applied when the integrity mode in the database session failover functionality is enabled.

5.7.1 Concurrent execution with the same session ID

The functionality of concurrent execution with the same session ID concurrently executes multiple requests when multiple requests of the same session ID are sent to multiple replicated J2EE servers or to one J2EE server. Global session information is not locked/released because multiple request processing are concurrently executed.

- **Notes**

  If you want to concurrently execute multiple requests processing by using concurrent execution of the same session ID, processing order of Servlet API issued by the web application will be uncertain.

  For the same HTTP session, if you concurrently send a request that registers attributes (the `setAttribute()` method in the `javax.servlet.http.HttpSession` interface) and a request that disables the session (the `invalidate()` method in the `javax.servlet.http.HttpSession` interface), or if you send duplicate requests that disable a session, the attributes for HTTP sessions, which are already disabled, might be registered or disabled depending on the processing order of Servlet API. In this case, Servlet API throws `java.lang.IllegalStateException` exception. As a result, implement web applications by considering that Servlet API throws `java.lang.IllegalStateException` exception.

5.7.2 Inheriting global session information when starting a web application

If you stop a web application or a J2EE server, or if a J2EE server becomes process down due to a failure, an HTTP session on the J2EE server is destroyed. The functionality that inherits global session information when restarting a web application is called *inhibiting global session information when starting a web application*.

The following is the procedure of inheriting global session information when restarting a web application after a J2EE server is down due to a failure.

1. Obtain the list of session IDs of the global session information to be inherited from a database.

   Obtain the list of session IDs of the global session information to be inherited from a database when starting a web application.

2. Execute the processing of inheriting the global session information.

   When you obtain the list of session IDs, output `KDJE34344-I` or `KDJE34429-I` message to the message log and start the processing of inheriting global session information.

   With processing of inheriting the global session information, inherit the global session information of the session IDs included in the list one by one from the database to the J2EE server.

   If you cannot inherit the global session information, a message corresponding to the cause of the failure is output.

3. End the processing of inheriting.

   When the processing of inheriting all global session information present in the list of session IDs completes, `KDJE34349-I` or `KDJE34430-I` message is output to the message log.

Note that global session information is not inherited in the case of the conditions described in the following table.

Table 5–6: Conditions and operations when global session information is not inherited

No.	Condition	Operation
1	If you cannot obtain the list of session IDs of the global session information to be inherited from a database due to a network failure.	`KDJE34345-W` or `KDJE34431-W` message is output to the message log and processing of inheriting the global session information ends.
2	If the global session information to be inherited already exists on the J2EE server (the information is already inherited on the J2EE server by receiving a request).	`KDJE34347-I` or `KDJE34432-I` message is output to the message log and processing of inheriting the global session information is skipped.
3	If the global session information to be inherited is already inherited on another replicated J2EE server by using the database session failover functionality.	`KDJE34348-I` message is output to the message log and processing of inheriting the global session information is skipped.
4	If you cannot obtain the global session information from a database due to a network failure.	`KDJE34346-W` message is output to the message log and processing of inheriting the global session information is skipped.
5	If you could not inherit because the number of HTTP sessions on the J2EE server reached the upper limit set by using the functionality of specifying the upper limit of number of `HttpSession` objects.	`KDJE34370-W` message is output to the message log and processing of inheriting the global session information is skipped.
6	If deserialization of the global session information fails.	`KDJE34328-E` or `KDJE34436-E` message is output to the message log and global session information is not inherited and deleted from the database.
7	If you are using the database session failover functionality and you change the server ID set by the functionality of adding server ID of `HttpSession`.	`KDJE34348-I` message is output to the message log and processing of inheriting the global session information is skipped.

5.7.3 Reducing an HTTP session

Reducing HTTP session is functionality that continues request processing by using HTTP session on the J2EE server without interrupting the processing, if a failure described in the following table occurs in a database.

Table 5–7: Contents of failures for which HTTP session reduction works

Failure occurrence location	Used functionality	Failure contents
Database	The database session failover functionality	• Blank records do not exist in the database when creating global session information • A database failure occurs while operating global session information

The following table describes operations when reducing an HTTP session due to a failure.

- **The database session failover functionality**

Table 5–8: Operations when reducing an HTTP session (in the case of the database session failover functionality)

Occurred failure	Reduction operation	Message output at the time of reduction#	Timing for releasing reduction	Inheriting a reduced HTTP session
Blank records do not exist in the database when creating global session information.	Creates only HTTP sessions on the J2EE server.	`KDJE34367-W`	If there are blank records in the database in subsequent HTTP session operations and if you could create global session information.	Not inherited because global session information does not exist in the database.
A database failure occurs while operating global session information.	Operates only HTTP sessions on the J2EE server.	`KDJE34368-W`	If you successfully access the database in subsequent HTTP session operations.	• If global session information does not exist in the database: Not inherited. • If global session information exists in the database: Old global session information might be inherited.

#

This message is output for each occurred failure when a session is reduced for the first time. Thereafter, the message is not output after all reduced sessions disappear until you perform session reduction once again. Note that `KDJE34369-I` message is output if all reduced HTTP sessions disappear.

5.8 Estimating memory

If you want to use the session failover functionality, estimate the following memory sizes as a preparation for environment setup.

- The database session failover functionality
  - Memory used in serialize processing
  - Size of HTTP session attribute information
  - Table capacity of database

This section describes how to estimate the size of memory.

5.8.1 Estimating memory used in serialize processing

With the session failover functionality, memory is temporarily allocated for serializing HTTP session attribute information when completing the request processing. You must consider the memory space required for this memory allocation when performing `JavaVM` tuning.

In tuning, estimate the increased amount of memory (*maximum increased amount*) considering the case if memory allocation processing duplicates in multiple threads. The formulas for calculating maximum increased amount of memory used for request processing, in web application unit and in J2EE server unit, are as follows:

```
Maximum increased amount of memory used in web application unit (bytes)=
Max Threads#1 x maximum size of HTTP session attribute information#2

Maximum increased amount of memory used in J2EE server unit (bytes)=
Total maximum increased amount of memory used in web application unit=
    Maximum increased amount of memory used in web application 1
  + Maximum increased amount of memory used in web application 2
      :
  + Maximum increased amount of memory used in web application n
```

#1

If you have set the number of concurrent execution threads in web application unit, indicates the value of Max Threads in web application unit. If you have not set the number of concurrent execution threads in web application unit, indicates the value of Max Threads in web container unit.

#2

Indicates the value estimated by using the functionality of estimating size of HTTP session attribute information.

Execute JavaVM tuning on the basis of the value obtained with the above formulas.

5.8.2 Estimating size of HTTP session attribute information

Maximum size of attribute information of an HTTP session is required when allocating disk space of the database used by the session failover functionality.

It is difficult to calculate and obtain the size of attribute information of an HTTP session from the contents of a web application. Therefore, the functionality for estimating size of HTTP session attribute information is provided with

Application Server. If you use the functionality for estimating the size of HTTP session attribute information, you can actually execute the application and output the size information of the attributes registered in the HTTP session, after serialization, as a message.

This subsection describes the functionality for estimating HTTP session attribute information size and calculation formulas used to obtain the size of attribute information of an HTTP session.

It also describes the memory that must be secured to suppress Full GC.

(1) Functionality for estimating HTTP session attribute information size

If you use the functionality for estimating HTTP session attribute information size, you can estimate the optimum maximum size of attribute information of an HTTP session by referring to the output size information.

This functionality is used for estimation. Because global information is not stored to a database, databaser are not connected.

Important note

Do not use the functionality for estimating HTTP session attribute information size in operating environment. If you use this functionality, the database session failover functionality become disabled and the global session information is not replicated to the database.

(a) Settings for enabling the functionality for estimating HTTP session attribute information size

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

- `webserver.dbsfo.check_size.mode` parameter

For details on Easy Setup definition file and parameters to be specified, see *4.3 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

If you enable the functionality for estimating HTTP session attribute information size, all other settings related to the database session failover functionality become disabled.

If you use the functionality for estimating HTTP session attribute information size, the functionality of specifying upper limit of the number of `HttpSession` objects and functionality of deleting HTTP Cookie that indicates session ID of HTTP session described in *5.4.2 Prerequisite settings* operate even if you disable it.

(b) Messages reporting the size of attribute information of an HTTP session

If you enable the functionality for estimating HTTP session attribute information size, the following messages reporting the attribute information of an HTTP session are output at Error level when the request processing of web application completes.

Table 5–9: Messages reporting the size of attribute information of an HTTP session

Message ID	Contents	Contents included in size information
`KDJE34330-I` or `KDJE34416-I`	Size of HTTP session attribution information created for each request	Total size of the result of serializing the attributes registered in the HTTP session (Total of the sizes output by `KDJE34331-I` or `KDJE34417-I`)#1

Message ID	Contents	Contents included in size information
KDJE34331-I or KDJE34417-I	Size of one attribute for which serialization is completed	• Size of the result of serializing attribute name (byte array) • Size of the result of serializing attribute value (byte array) • Magic number written by java.io.ObjectOutputStream class#2 • Size of version information data written by java.io.ObjectOutputStream class#2

#1

If attributes are not registered in an HTTP session, it is the size of a magic number written by `java.io.ObjectOutputStream` class and version information data written by `java.io.ObjectOutputStream` class.

#2

It is included only in the size of attribute, which is serialized first.

The following is an example of message output when you create HTTP session attribute information from an HTTP session in which the registered attributes are "Attribute 1" and "Attribute 2" (in the case of the database session failover functionality).

```
KDJE34331-I An attribute was serialized. (J2EE application = App01, context
root = /test, request URL = http://host01/test/TestServlet, attribute name
= Attribute1, class name = app.MyObject1, size(bytes) = 36, HTTP session ID
= 01234567aaaabbbbccccddddeeeeffff)
KDJE34331-I An attribute was serialized. (J2EE application = App01, context
root = /test, request URL = http://host01/test/TestServlet, attribute name
= Attribute2, class name = app.MyObject2, size(bytes) = 25, HTTP session ID
= 01234567aaaabbbbccccddddeeeeffff)
KDJE34330-I The attribute information was created. (J2EE application = App01
, context root = /test, request URL = http://host01/test/TestServlet, size(b
yte) = 61, HTTP session ID = 01234567aaaabbbbccccddddeeeeffff)
```

(2) Calculation formulas for determining the size of HTTP session attribute information

You can determine the maximum size of HTTP session attribute information by using the following formulas.

Here, the size is calculated by considering that serialization is performed for one HTTP session by using one `java.io.ObjectOutputStream` object.

```
Maximum size of HTTP session attribute information (bytes)=
    Total number of bytes of byte arrays that have serialized attribute name
s of all attributes registered in the HTTP session
  + Total number of bytes of byte arrays that have serialized attribute valu
es of all attributes registered in the HTTP session
```

If you register n objects as attributes in an HTTP session and name the registered attributes from attribute 1 to attribute n, you can determine the maximum size of HTTP session attribute information with the following formula:

```
Maximum size of HTTP session attribute information (bytes)=
    Number of bytes of byte array that has serialized attribute name of attr
ibute 1
  + Number of bytes of byte array that has serialized attribute value of att
ribute 1
  + Number of bytes of byte array that has serialized attribute name of attr
```

```
ibute 2
  + Number of bytes of byte array that has serialized attribute value of att
ribute 2
    : (omitted)
  + Number of bytes of byte array that has serialized attribute name of attr
ibute n
  + Number of bytes of byte array that has serialized attribute value of att
ribute n
```

You can determine the number of bytes of a byte array that has a serialized attribute name and the number of bytes of a byte array that has a serialized attribute value by the following formulas:

Number of bytes of a byte array that has serialized attribute name

```
Number of bytes of a byte array that has serialized attribute name=
  Number of characters in attribute name x 3 x 1.2
```

Number of bytes of a byte array that has serialized attribute value

```
Number of bytes of a byte array that has serialized attribute value=
  Total number of bytes of the values of all fields, possessed by the object
s of attribute value x 1.2
```

You can determine the number of bytes of field values by using the following formulas:

- In the case of String objects: Number of characters × 3
- In the case of other objects: Total number of bytes of the values of all fields, possessed by the object
- In the case of primitive type: Number of bytes required for storing each primitive type

Important note

The value, which you can calculate by using the calculation formula that determines the size of HTTP session attribute information, is a roughly estimated value. If you want to determine the conclusive maximum value of HTTP session attribute information, use the functionality for estimating HTTP session attribute information size.

(3) Memory that must be secured to suppress Full GC

Because the size of the HTTP session attribute information is the size after serialization, the size is different from the size of attribute objects, which are registered in the HTTP session, in the memory. Therefore, you must make a separate estimate of the memory in the external heap area required to suppress Full GC and set an appropriate value according to the estimate.

For details about suppression of Full GC, see *7. Suppression of Full GC by Using the Explicit Memory Management Functionality*.

5.8.3 Estimating disk space of a database

With the database session failover functionality, create three types of tables (application information table, session information table, and blank record information table). Estimate the size of disk space to be allocated by referring to the of each database on the basis of table and index information. Note that this information might change in version upgrade or modification patch of Component Container.

(1) Table information

This subsection describes the elements of column for each table and number of rows.

- Application information table

The following table describes the elements of a column.

Table 5–10: Elements of a column in application information table

No.	Column name	HiRDB type	ORACLE type	Index existence status
1	`APP_INFO_KEY`	CHAR(128) PRIMARY KEY	VARCHAR2(128) PRIMARY KEY	None
2	`APP_INFO_VALUE`	CHAR(512)	VARCHAR2(512)	None

The number of rows is as follows:

13 + Number of definitions of refer-only requests

- Session information storage table
  The following table describes the elements of a column.

Table 5–11: Elements of a column in session information storage table

No.	Column name	HiRDB type	ORACLE type	Index existence status
1	`RECORD_NO`	INTEGER PRIMARY KEY	NUMBER(10,0) PRIMARY KEY	None
2	`SESSIONID`	CHAR(112)	VARCHAR2(112)	Yes
3	`CREATION_TIME`	DECIMAL(23,0)	NUMBER(23,0)	None
4	`MAX_INACTIVE_INTERVAL`	INTEGER	NUMBER(10,0)	None
5	`THIS_ACCESSED_TIME`	DECIMAL(23,0)	NUMBER(23,0)	None
6	`ATTRIBUTES_DATA`	BINARY (maximum size of HTTP session attribute information)#1	BLOB#2	None
7	`STATUS`	CHAR(16)	VARCHAR2(16)	None
8	`OWNER_SERVER`	CHAR(512)	VARCHAR2(512)	None
9	`NEXT_FREE_RECORD_NO`	INTEGER	NUMBER(10,0)	None

#1

For details on estimating size of HTTP session attribute information, see *5.8.2 Estimating size of HTTP session attribute information*.

#2

Maximum size of the values stored in BLOB column is the maximum size of HTTP session attribute information. For details on estimating size of HTTP session attribute information, see *5.8.2 Estimating size of HTTP session attribute information*.

The number of rows is as follows:

- If you set to continue the start processing of web applications when negotiation processing fails
  Number of global session information stored in the database
- If you set to discontinue the start processing of web applications when negotiation processing fails
  Maximum value of the number of HttpSession objects

- Blank record information table

The following table describes the elements of a column.

Table 5–12: Elements of a column in blank record information table

No.	Column name	HiRDB type	ORACLE type	Index existence status
1	BLOCK_NO	INTEGER PRIMARY KEY	NUMBER(10,0) PRIMARY KEY	None
2	FREE_RECORD_NO	INTEGER	NUMBER(10,0)	None

The number of rows is fixed to 10.

(2) Index information

The following table describes the index of session information storage table.

No.	Index name	UNIQUE attribute	Column name
1	*Application-identifier*_SESSIONS_IDX	None	SESSIONID

Reference note

If you use HiRDB, performance might be improved if you satisfy the following conditions:

- The tables and indexes used in the database session failover functionality are placed in RD area#.
- Global buffer is set for each table and index placed in RD area.

For details on design of RD area and global buffer, see the *HiRDB Installation and Design Guide*.

#

If you place tables and indexes in RD area, you must edit the SQL file.

5.9 Precautions

This section describes precautions to be taken when using the session failover functionality and executing an application.

5.9.1 HTTP session that is implicitly created in JSP

Set not to implicitly create `HttpSession` objects with the processing that does not require session inheriting.

With the application for which you enabled the session failover functionality, global session information is created and processing of updating global session information occurs even when creating an HTTP session without registering attributes.

With JSP specifications, `HttpSession` objects are created by default. As a result, unnecessary processing might increase memory usage and generate load due to communication with a database.

Use `session` attribute of `page` directive for performing settings related to creating `HttpSession` objects.

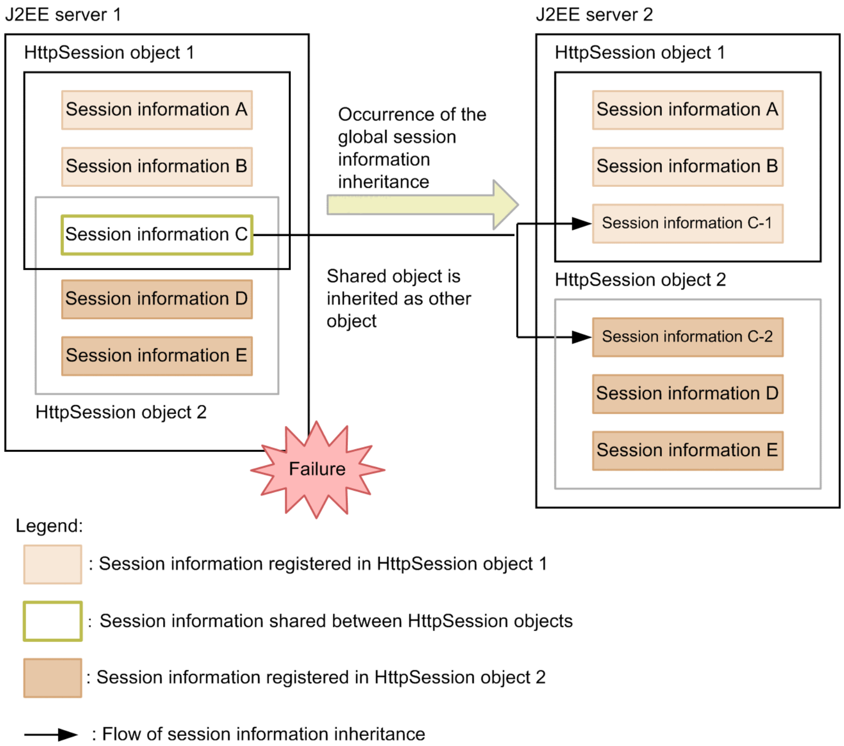
5.9.2 Processing considering that the same objects are registered in different HTTP sessions

Global session information is created in HTTP session unit.

If you have shared the same objects as session information in different `HttpSession` objects, the objects are not shared when inheriting the global session information. The objects are created as separate objects.

The following figure shows an example of inheriting when you have registered the same object in different `HttpSession` objects.

Figure 5–11: An example of inheriting when you have registered the same object in different HttpSession objects.



In this figure, session information C of the same object is shared in HttpSession object 1 and HttpSession object 2 on J2EE server 1. If a failure occurs on J2EE server 1 and the session information is inherited on J2EE server 2, the shared session information C is respectively created in HttpSession object 1 and HttpSession object 2 on J2EE server 2 as separate session information C-1 and session information C-2. Instances of session information C-1 and session information C-2 differ, but the contents are same.

5.9.3 Handling authentication information when inheriting session information

With Application Server, `Form` authentication, `Basic` authentication, and the `authenticate/login/logout` method of the `HttpServletRequest` are used as login authentication functionality. If you use this login authentication functionality in an application that uses the session failover functionality, the operations are as follows:

If you use `Form` authentication

If a failure occurs on the J2EE server and the session is to be inherited, you must once again perform authentication with `Form` authentication even if the session is successfully inherited.

If you use `Basic` authentication

You can continuously access without once again performing `Basic` authentication regardless of whether the session is to be inherited due to a failure on the J2EE server.

If you use the `authenticate/login/logout` method of the `HttpServletRequest`

If a failure occurs on the J2EE server and the session is to be inherited, you must once again perform authentication with the method even if the session is successfully inherited.

For details on `Basic` authentication and `Form` authentication, see *6.2 Web container-based authentication using DD settings* in the *uCosminexus Application Server Security Management Guide*.

5.9.4 Impact on servlet API

This subsection describes the following items as an impact on servlet API when using the session failover functionality.

- Operating servlet API related to `HttpSession` objects after inheriting a session
- Communicating with a database by invoking a servlet API

(1) Operating servlet API related to HttpSession objects after inheriting a session

The following table describes the notes on servlet API related to `HttpSession` objects after inheriting a session.

Table 5–13: Notes on servlet API related to HttpSession objects

No.	API name	Notes
1	`getCreationTime()`	If an `HttpSession` object is created by inheriting, the information of the `HttpSession` object before inheriting is inherited.
2	`getLastAccessedTime()`	
3	`getId()`	If an `HttpSession` object is created by inheriting, you can obtain the same ID as the `HttpSession` object before inheriting.
4	`isNew()`	Even if an `HttpSession` object is created by inheriting, return value *true* is not returned.

The servlet AIPs that are not described in this table are not impacted when the session failover functionality is used.

(2) Communicating with a database by invoking a servlet API

If you implement the servlet APIs described in the following table, communication with a database occurs as the extension of API invocation. As a result, performance is affected.

Table 5–14: Communication with a database

No.	Class	Method
1	`javax.servlet.http.HttpServletRequest`	`getSession()`#1
2	`javax.servlet.http.HttpServletRequest`	`getSession(boolean create)`#1
3	`javax.servlet.http.HttpSession`	`invalidate()`#2

#1

Performance is affected only if you create new `HttpSession` object.

#2

Performance is affected only if you invoke the `invalidate()` method in an enabled the `HttpSession` object.

6 Database session failover functionality

This chapter describes the database session failover functionality.

6.1 Organization of this chapter

This section describes the *database session failover functionality*.

If you use this functionality, information of a session that is running on the application is stored in the database. If a failure occurs on a Web server or a J2EE server, stored session information is passed to another J2EE server. As a result, even if requests are transferred to another J2EE server when a failure occurs, you can continue operations in the state before failure.

For details on the types, functionality differences, prerequisites, memory estimation, and notes related to the session failover functionality, see *5. Inheriting Session Information Between J2EE Servers*.

The following table describes the organization of this chapter.

Table 6–1: Organization of this chapter (the database session failover functionality)

Category	Title	Reference location
Description	Application procedure	*6.2.1*
	Selecting a mode in which performance is important (disabling integrity mode)	*6.3*
	Processing implemented in the database session failover functionality	*6.4*
Implementation	Definitions in `cosminexus.xml`	*6.5*
Setup	J2EE server settings	*6.6.1*
	Web application settings	*6.6.2*
	Database settings	*6.6.3*
	DB Connector settings	*6.6.4*
	Changing the settings related to the database session failover functionality	*6.7*
	Deleting database tables	*6.8*
Notes	Precautions to be taken when using the database session failover functionality	*6.9*

#

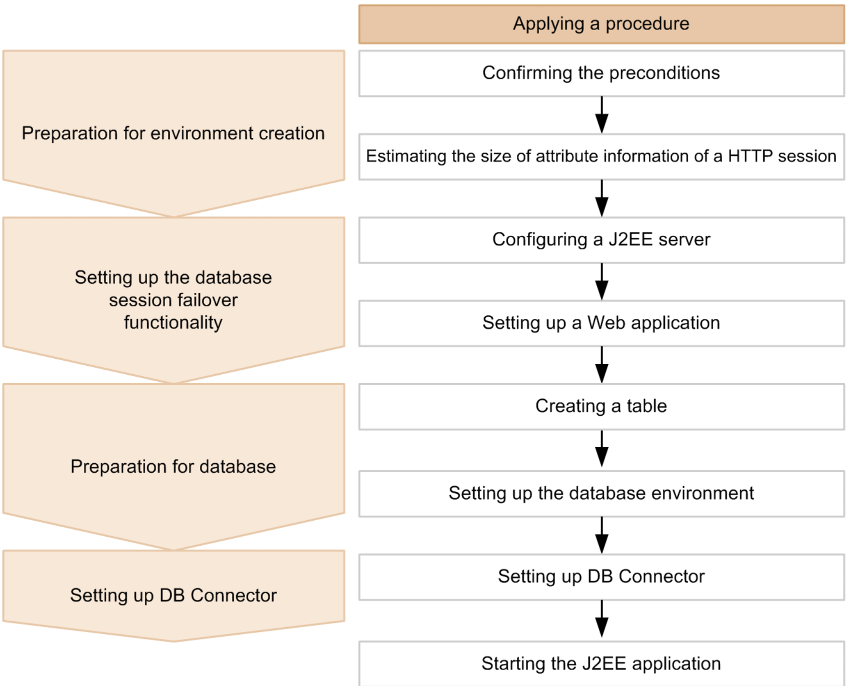
This functionality does not have any specific explanation in the *Operation*.

6.2 Preparation for using the database session failover functionality

6.2.1 Application procedures

This section describes preparations and various settings in environment setup required for using the database session failover functionality. The following figure shows the application procedures for the database session failover functionality.

Figure 6–1: Application procedures (the database session failover functionality)



Start a J2EE application after implementing preparations and various settings in environment setup in accordance with the application procedures shown in this figure.

(1) Preparing environment setup

The following table describes implementation contents and reference locations of the items to be implemented as preparations of environment setup when using the database session failover functionality.

Table 6–2: Implementation contents and reference locations of items to be implemented as preparations of environment setup when using the database session failover functionality

Implementation sequence	Implementation item	Implementation contents	Reference location
1	Checking prerequisites	Check configuration and settings that are prerequisites.	5.4
2	Estimating the size of HTTP session attribute information	Estimate the size of HTTP session attribute information. The estimated value is required for environment settings of a database.	5.8.2

(2) Settings of the database session failover functionality

The following table describes settings and reference locations of the settings of the database session failover functionality.

Table 6–3: Settings and reference locations of the database session failover functionality

Setting sequence	Setting item	Settings	Reference location
1	J2EE server settings	Perform the following: • Specifying the database session failover functionality (J2EE server unit) • Specifying an optional name of DB Connector • Specifying the integrity mode • Setting up the memory used in serialize processing • Specifying the session failover inhibition functionality (extension or URI unit) • Specifying the refer-only request • Specifying the server ID addition functionality of `HttpSession` • Specifying if pending queues are insufficient when using a functionality for controlling the number of concurrent execution threads • Setting up the Web application start processing when negotiation fails • Specifying the exceptions when executing the `getSession` method in the requests targeted for inhibiting the database session failover functionality	*6.6.1*
2	Web application settings#	Specify the following: • Specifying the database session failover functionality (Web application unit) • Specifying the upper limit of the number of `HttpSession` objects • Specifying the application identifier • Specifying the maximum size of HTTP session attribute information • Specifying the database session failover inhibition functionality by using an extension	*6.6.2*

Implement the settings of a Web application in a development environment. If you want to perform settings of a Web application in the execution environment by using server management commands, see *6.6.2 Web application settings*.

(3) Preparing a database

The following table describes implementation contents and reference locations of the items to be implemented as preparations of a database when using the database session failover functionality.

Table 6–4: Implementation contents and reference locations of items to be implemented as preparations of a database

Implementation sequence	Implementation item	Implementation contents	Reference location
1	Creating tables	• Allocating a disk space of a database • Creating an application information table • Creating a session information storage table • Creating a blank record information table	*5.8.3*, *6.6.3(2)*, *6.6.3(3)*, *6.6.3(4)*
2	Environment setup of database	Specify the following: • Specifying a timeout of a database	*6.6.3(5)*

(4) DB Connector settings

The following table describes settings and reference locations of DB Connector settings required for using the database session failover functionality.

Table 6–5: Settings and reference locations of DB Connector

Setting sequence	Setting item	Settings	Reference location
1	DB Connector settings	Soecify the following: • Specifying the transaction support level • Environment settings of DB Connector • Specifying an optional name of DB Connector	*6.6.4*

(5) Note

If you use HiRDB as a database, make sure that automatic commit is enabled. If automatic commit is disabled, an error occurs when an SQL statement is executed in the database session failover functionality.

6.3 Selecting a mode in which performance is important (disabling integrity mode)

This section describes the operations, functionality that you can use (deleting global session information), and precautions to be taken if you select a mode that emphasizes performance.

For selecting a mode that emphasizes performance, disable the setting of integrity mode (default). If you disable the integrity mode, you can concurrently execute the request processing of the same session ID (concurrent execution of the same session ID).

6.3.1 Operations performed when disabling integrity mode

For details on operations when you disable the integrity mode, see *5.7 Functionality executed when using a session failover functionality*.

6.3.2 Deleting global session information

Validity of global session information is monitored by monitoring HTTP sessions on a J2EE server. As a part of monitoring validity, the global session information on the database is deleted for HTTP sessions for which the validity of has expired. However, if a J2EE server stops due to a failure, the global session information used on that server is inherited on another J2EE server or the validity is not monitored until you restart the J2EE server. If the state of not monitoring the validity continues for a long time, the global session information, which is not deleted even if validity has elapsed, continues using records in the session information storage table.

Therefore, you must appropriately delete the global session information remaining in the database.

The following subsection describes how to delete global session information by using a command:

- **For deleting the global session information**

  Use the `cjclearsession` command for deleting the global session information. Execute the command before restarting a J2EE server or a Web application after the J2EE server or a Web application stops, and time exceeding the validity of the HTTP session has elapsed.

  In a Web application, if you have set validity for each HTTP session by using Sevlet API, execute the command in accordance with longest validity.

  The following are the procedures for deleting global session information:

  1. In the environment variable `CLASSPATH`, set the path of a JDBC driver to be used.

     When using the `cjclearsession` command for the first time, specify a path of the JDBC driver to be used in the environment variable `CLASSPATH`.

  2. Execute the `cjclearsession` command for deleting the global session information.

     Specify the application identifier, server ID, and information of a JDBC driver to be used, and information required for accessing the database with the command, and execute. All the global session information possessed by the J2EE server specified in the server ID of a Web application that is specified in application identifier, is deleted.

  3. Restart the J2EE server or a Web application if required.

  If you specify the `-count` option in the `cjclearsession` command and execute the command, you can view the number of global session information possessed by the J2EE server.

The timeout for connection tries to database and execution timeout of SQL that acquires or deletes the global session information of database is eight seconds.

If an error occurs during database access while executing the command, stop the command execution at the point at which the error occurred.

For details on the `cjclearsession` command, see *cjclearsession (Delete global session information (database session failover functionality))* in the *uCosminexus Application Server Command Reference Guide*.

- **Notes**

  Notes for deleting global session information:

  - Deleting information when the J2EE server, which owns the HTTP session to be deleted, is running

    If the J2EE server is running, request processing might be performed and the global session information might be newly created. As a result, if the J2EE server, which owns the HTTP session to be deleted, is running, the information might be deleted before the validity of the global session expires. When deleting global session information, stop the J2EE server, which owns the HTTP session to be deleted, and then execute the command.

  - Deleting before validity expires

    If you delete the global session information by executing the `cjclearsession` command before the validity of the global session expires, the operations are as follows.

Sr. No.	Integrity mode	Existence status of HTTP session on the J2EE server	Operation
1	Disables	None	You cannot inherit the global session.
2		Yes	Thereafter, the deleted global session information is not stored in the database and a Web application operates only with the HTTP session on the J2EE server.

  - Deleting when integrity mode is enabled

    Operation is not guaranteed if the integrity mode is enabled.

  - If using Oracle JDBC Thin Driver

    The `cjclearsession` command implements timeout when executing an SQL by using the `setQueryTimeout` method of the JDBC driver. For details on the points to be considered when connecting to Oracle by using Oracle JDBC Thin Driver, see *3.6.6 Prerequisites and notes when connecting to Oracle* in the *uCosminexus Application Server Common Container Functionality Guide.*

6.3.3 Notes

This subsection describes the points to be considered when you disable integrity mode.

- **Switching integrity mode**

  If you switch integrity mode from disabled to enabled, initialize the session information storage table and the application information table in the database as described in the following procedure:

  1. Stop all replicated J2EE servers.
  2. Destroy the HTTP session.

     For details on the procedures for destroying an HTTP session, see *6.7.3 Deleting global session information (destroying HTTP sessions)*.
  3. Initialize the preference information stored in the database.

     For details on the procedures for initializing the preference information stored in a database, see *6.7.2 Initializing a database table*.

- **Monitoring validity of global session information when stopping a J2EE server**

  If you stop a Web application or J2EE server, or if a process goes down due to a failure on a J2EE server, the validity of global session information is not monitored. Monitoring of validity starts when you start a Web application or when global session information is inherited on a J2EE server by receiving a request.

  If integrity mode is enabled and when a J2EE server stops, another J2EE server monitors the validity. For details on validity monitoring processing, see *6.4.3 Processing when validity of global session information expires*.

- **Operations performed when the amount of global session information reaches the upper limit**

  Reduce an HTTP session if the amount of global session information in a database reaches the upper limit when creating global session information. For details on reducing an HTTP session, see *5.7.3 Reducing an HTTP session*.

6.4 Processing implemented in the database session failover functionality

With the database session failover functionality, respective processing is executed at the following points of time:

- When starting an application
  Application negotiation processing is executed.
- When executing a request
  Global session information is stored, updated, and deleted.

This section describes the processing executed in the database session failover functionality.

This section also describes the processing when the validity of global session information expires, listeners that operate in association with events that occur in the database session failover functionality, the processing of locking global session information executed only in the case of integrity mode, and the operations performed when a failure occurs during global session information operations.

6.4.1 Processing when starting an application

This subsection describes the application negotiation processing executed when an application starts and the application identifiers used in application negotiation processing.

(1) Application negotiation processing

With a Web application that uses the database session failover functionality, *negotiation processing* is executed when you start an application.

With negotiation processing, the following contents are checked:

- Web applications are matching
- Settings of each Web application are matching
- J2EE server settings are matching
- Database settings are correct

The result of negotiation processing determines whether a Web application will start.

The following table describes the relation between the result of negotiation processing and the Web application states.

Table 6–6: Relation between the result of negotiation processing and the Web application states

Result of negotiation processing	Web application state	Cause of negotiation failure	Output message
Successful (no problem in checked contents)	Started	--	`KDJE34306-I`
Failed (there is a problem in checked contents)	Not started	Web application is not matching.	`KDJE34340-E`
	Not started#	Web application settings are not matching.	`KDJE34307-E`
	Started#		`KDJE34358-I`
	Not started	J2EE server settings are not matching.	`KDJE34307-E`

Result of negotiation processing	Web application state	Cause of negotiation failure	Output message
	Not started	Required tables do not exist in the database.	KDJE34308-W
	Not started	Contents of the existing table are not the contents of the table for the database session failover functionality.	KDJE34309-E
	Not started	Existing table is used by another application.	KDJE34340-E

Legend:

--: Not applicable

#

For the following confirmation items, if you have set different values in a Web application to be started and the same Web application on another J2EE server, you can select whether to continue or stop the processing of starting a Web application by specifying the `webserver.dbsfo.negotiation.high_level` parameter in the `<configuration>` tag of a logical J2EE server (j2ee-server) in the Easy Setup definition file.

- Upper limit of the number of `HttpSession` objects
- Validity of HTTP sessions defined in DD (`web.xml`)

If other confirmation contents are not matching, Web applications do not start.

If you set to stop the processing of starting a Web application, make sure to specify a valid value (1 or greater) as the upper limit of the number of `HttpSession` objects.

If an error occurs while accessing the database during negotiation processing, the `KDJE34312-W` message is output to the message log.

(a) Contents checked in negotiation processing (the database session failover functionality)

This subsection describes the contents checked in negotiation.

- **Web applications are matching**

  It is determined that Web applications are matching when all confirmation items match. The following table describes the confirmation items.

Table 6–7: Items used for confirming whether web applications are matching

Sr. No.	Confirmation item
1	Application identifier#
2	J2EE application name
3	Web application name (context root name)

For details on application identifier, see *(2) Application identifier*.

- **Settings of each Web application are matching**

  Whether the settings of each replicated Web application are matching is checked for the confirmation items described in the following table.

Table 6–8: Items used for confirming whether the settings of each Web application are matching

Sr. No.	Confirmation item
1	Upper limit of the number of `HttpSession` objects
2	Maximum size of HTTP session attribute information that you can include in global session information

Sr. No.	Confirmation item
3	Validity of HTTP sessions defined in DD (`web.xml`)
4	Extensions that inhibit the database session failover functionality

- **J2EE server settings are matching**
  Whether the settings of each replicated J2EE server are matching is checked for the confirmation items described in the following table.

Table 6–9: Items used for confirming whether the settings of each J2EE server are matching

Sr. No.	Confirmation item
1	Settings of integrity mode
2	Settings of refer-only requests
3	Settings if pending queues are insufficient when using the functionality for controlling the number of concurrent execution threads
4	Settings of exception when executing the `getSession` method in the requests targeted for inhibiting the database session failover functionality

- **Database settings are correct**
  Whether the conditions described in the following table are satisfied is checked.

Table 6–10: Conditions for checking that the database settings are correct

Sr. No.	Condition
1	Required tables exist in the database.
2	Contents of existing table are the contents of the table used for the database session failover functionality.
3	Existing tables are not used in other applications.

(b) Settings of a Web application that are checked in negotiation

First of all, settings of a Web application that succeeds in negotiation processing are stored in the application information table in the database. Stored settings are treated as correct preference information used in negotiation confirmation.

Therefore, if you want to change the settings of a Web application, you must delete the preference information already stored in the database that is related to a Web application targeted to make changes. For details on the procedures for changing the settings, see *6.7 Changing settings related to the database session failover functionality*.

(2) Application identifier

The *Application identifier* is a name used for recognizing clustered Web applications when using the database session failover functionality. By default, the system automatically generates application identifiers.

An application identifier is used in negotiation for checking whether Web applications are matching. Therefore, an application identifier must satisfy the following conditions:

- An application identifier matches the same Web application that operates on the replicated J2EE servers.
- An application identifier is a unique value in the system.

If an application identifier, which is automatically generated by the system, does not satisfy a condition, you need to define values that satisfy the conditions. For details on how to define an application identifier, see *6.5 Definitions in cosminexus.xml*.

The following subsections describe the rules for automatically generating an application identifier and examples of automatically generated application identifiers.

> **Important note**
>
> If the same application identifier is set to different Web applications, the second Web application fails in negotiation when starting and does not start.

(a) Rules for automatically generating an application identifier

By default, a string based on context root name is automatically set in application identifiers. If an application identifier is automatically generated, the applicable value is output to the message log with the `KDJE34302-I` message, when starting a Web application.

The following rules are applied when automatically generating an application identifier on the basis of context root name:

- Delete the forward slash (/) at the beginning.
- If the length of the string exceeds 16 characters, excluding the forward slash (/) at the beginning, use a string up to 16 characters.
- If characters, which cannot be used in an application identifier, are used in the context root name, replace the characters with underscores (_).
  You can use only alphanumeric characters (A - Z, a - z, and 0 - 9) and underscores (_) in an application identifier. Set values are case-sensitive.
- In root context, change to *ROOT* and not to a blank string.

If you apply the rules for automatic generation, an application identifier might not remain unique in the system. In that case, the second Web application, to which the same application identifier is set, fails in negotiation when starting and does not start. Therefore, it is essential to set an application identifier that unique in the system, for Web applications.

(b) Examples of automatically generated application identifiers

The following table shows examples of default application identifiers, which are automatically generated from context root name.

Table 6–11: Examples of automatically generated default application identifiers

Sr. No.	Context root name	Application identifier	Rules applied when creating
1	`/examples`	`examples`	Delete the forward slash (/) at the beginning
2	`/App01/test1`	`App01_test1`	• Delete the forward slash (/) at the beginning • Replace the forward slash (/) in between with an underscore (_)
3	`/WebApplication_001`	`WebApplication_0`	• Delete the forward slash (/) at the beginning • Delete 17th and later characters
4	`/examples/WebApplication`	`examples_WebAppl`	• Delete the forward slash (/) at the beginning • Replace the forward slash (/) in between with an underscore (_) • Delete 17^{th} and later characters
5	`/`	`ROOT`	Because it is root context, change to *ROOT*

6.4.2 Processing when executing a request

This subsection describes creating, updating, and deleting a global session when executing a request, and inheriting a global session.

If processing is executed in a Web application, processing for global session information occurs as an extension to the processing. The following table describes examples of processing executed in Web applications, processing executed for global session information at the time of executing requests corresponding to the example, and reference locations.

Table 6–12: Examples of processing in Web applications and mapping of processing executed for global session information

Sr. No.	Example of processing executed in Web applications	Processing executed for global session information	Reference location
1	Login	Creating global session information	*(1)*
2	Executing work (page transition/update)	Updating global session information	*(2)*
3	Logout	Deleting global session information	*(3)*
4	Logout due to timeout	Deleting global session information due to expiry of validity	*6.4.3*
5	Executing work after inheriting a global session on another J2EE server (when a failure occurs on a J2EE server)	Inheriting session information that uses global session information	*(4)*

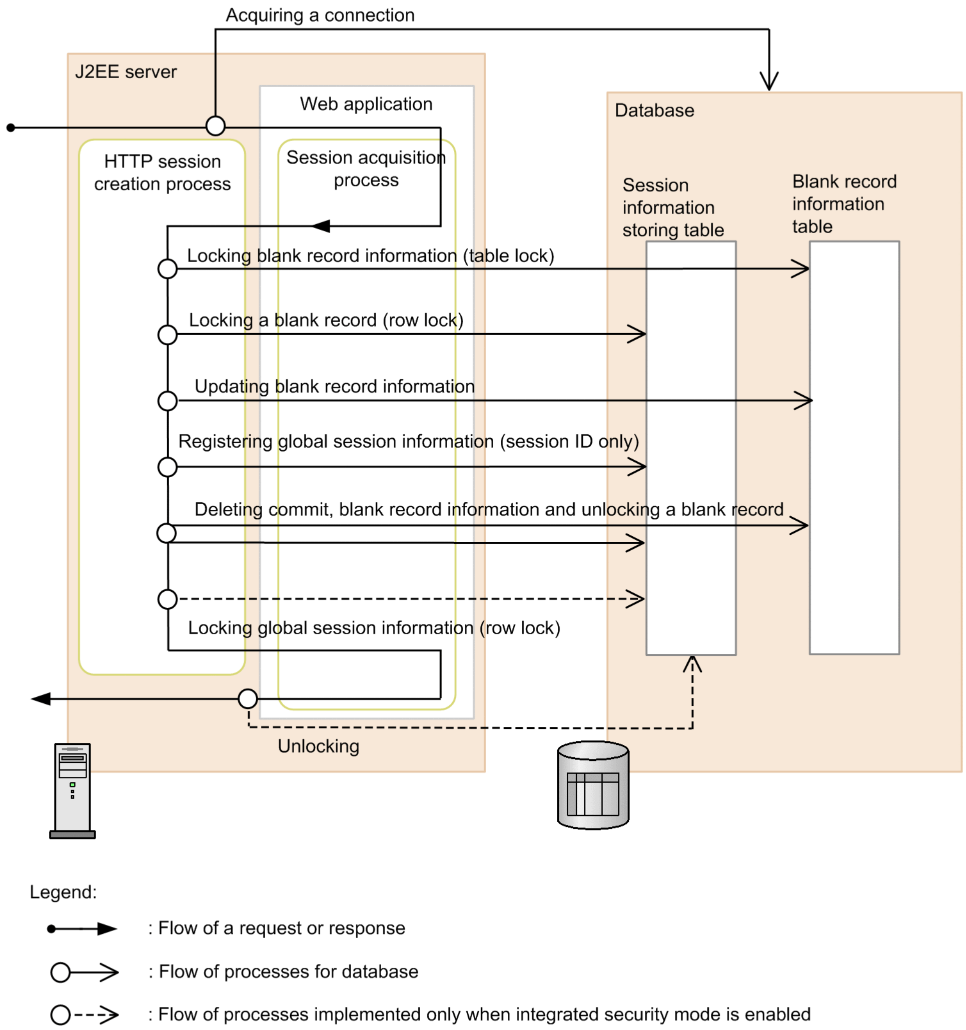
For processing results when a failure occurs during global session information operation, see *6.4.6 Operations performed when a failure occurs during global session information operation*.

(1) Creating global session information

When a new HTTP session is created on a J2EE server, global session information is created in the database.

The following figure shows the flow of processing executed when creating global session information.

Figure 6–2: Creating global session information (the database session failover functionality)



1. If an HTTP session receives the necessary request, a new HTTP session is created. An HTTP session is created when the `HttpSession` object is newly acquired by using the `getSession()` method or the `getSession(true)` method of the `javax.servlet.http.HttpServletRequest` interface in a Web application.

   Because `HttpSession` objects are also created in the following cases, a new HTTP session is created:

   - If you use the `Form` authentication
   - If you specify *true* in the `session` attribute of the `page` directive in JSP
   - If you omit specification of the `session` attribute of the `page` directive in JSP

2. The Global session information is created in the database as an extension to the HTTP session creation processing. Created global session information is stored in the session information storage table.

   The global session information is locked at the time of creation.

3. Blank record information is updated with the creation of global session information.

4. Created global session information is committed once.

   If integrity mode is enabled, a lock is acquired again. This is performed to avoid generation of inconsistency between the session information storage table and blank record information tables due to occurrence of a failure on a J2EE server or in a database, on which a Web application is running, after the HTTP session is created.

5. The HTTP session is updated when the processing in a Web application completes.

6. Global session information is updated as an extension to HTTP session update processing. If integrity mode is enabled, the lock is released after completing the update.

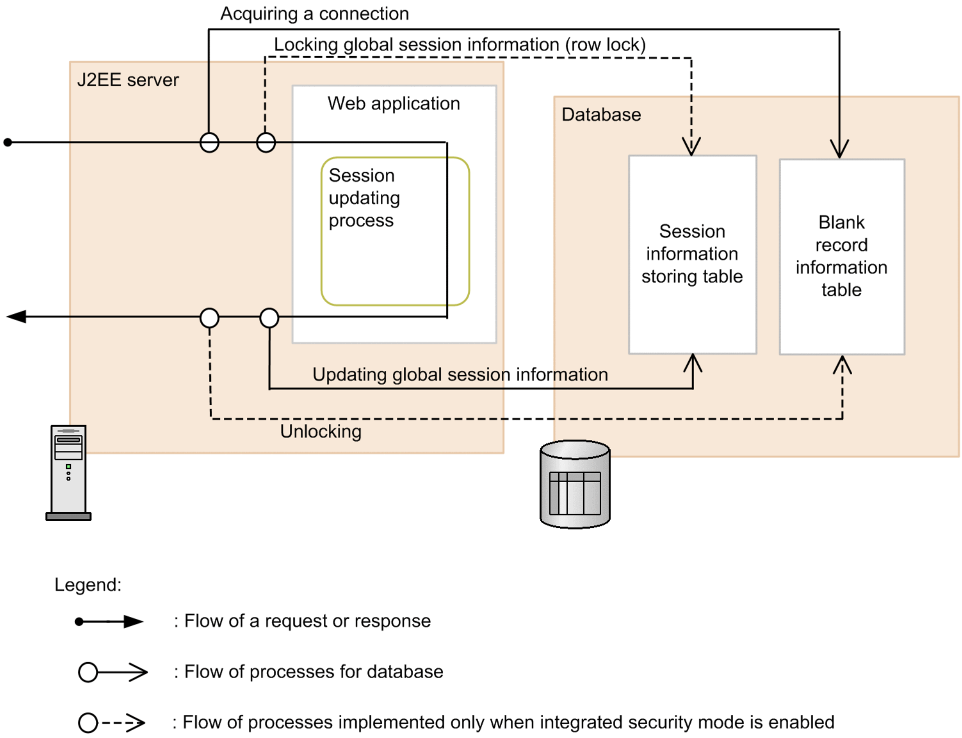
> **Important note**
>
> Operations performed when the amount of global session information reaches the upper limit
>
> Reduce the HTTP session if the amount of global session information in the database reaches the upper limit when creating global session information. For details on reducing an HTTP session, see *5.7.3 Reducing an HTTP session*.
>
> If the amount of global session information in the database reaches the upper limit when integrity mode is enabled, `java.lang.IllegalStateException` is thrown and acquiring the HTTP session fails.
>
> If you specify *true* in the `webserver.session.max.throwHttpSessionLimitExceededException` parameter in the `<configuration>` tag of a logical J2EE server (j2ee-server) in the Easy Setup definition file, `com.hitachi.software.web.session.HttpSessionLimitExceededException` is thrown instead of `java.lang.IllegalStateException`. For details on `HttpSessionLimitExceededException`, see *3.1 Exception classes* in the *uCosminexus Application Server API Reference Guide*.

(2) Updating global session information

When a session is updated during execution of a Web application, the HTTP session is updated on the J2EE server. At the same time, global session information in the database is also updated.

The following figure shows the flow of processing executed when updating global session information.

Figure 6–3: Updating global session information (the database session failover functionality)



1. Receive the request having an HTTP session.

If integrity mode is enabled, global session information in the database is locked before executing a Web application.

2. Along with updating of the session in a Web application, the HTTP session is updated.
3. The global session information in the database is updated to the latest information when the HTTP session is updated.
   If integrity mode is enabled, the lock is released.

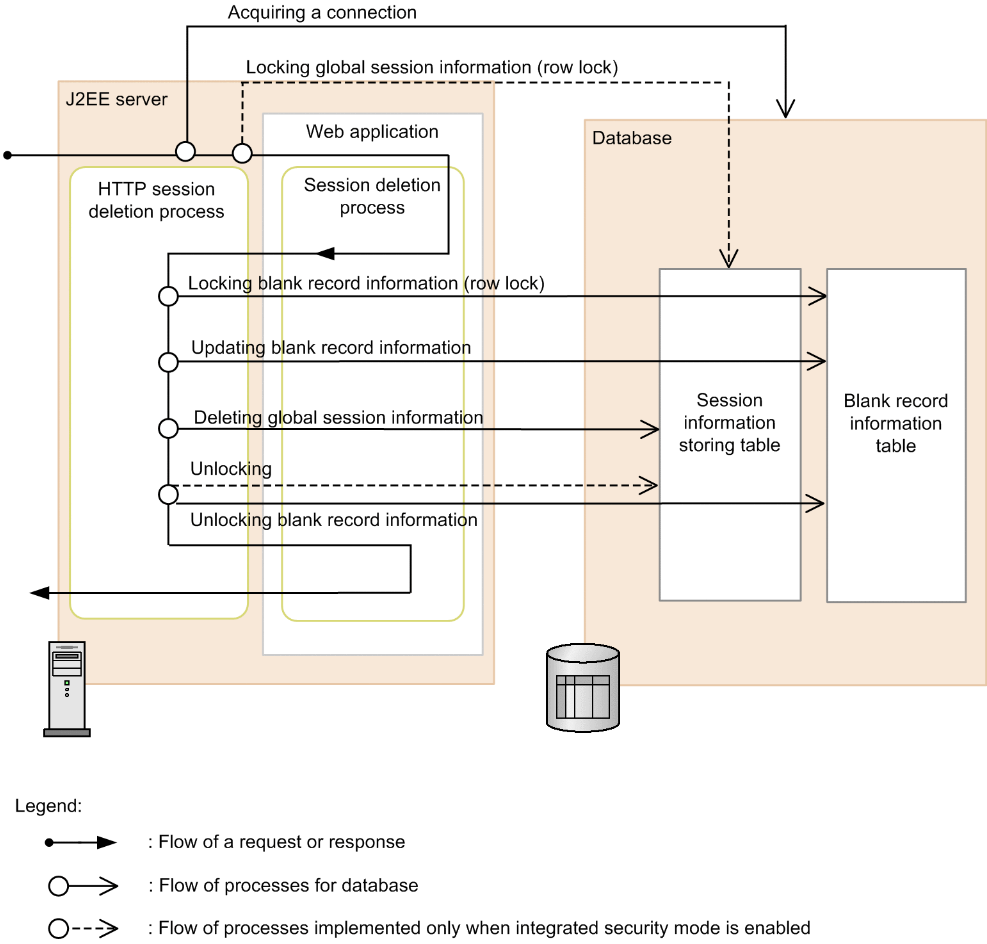
For details on the Operations performed when global session information is locked, see *6.4.5(1) Result of invoking lock acquisition processing when acquiring a lock*.

(3) Deleting global session information

If you implement the `invalidate()` method of the `javax.servlet.http.HttpSession` interface in session deletion processing in a Web application and explicitly delete an HTTP session, global session information in the database is deleted as an extension to that processing.

The following figure shows the flow of processing executed when deleting global session information.

Figure 6–4: Deleting global session information (the database session failover functionality)



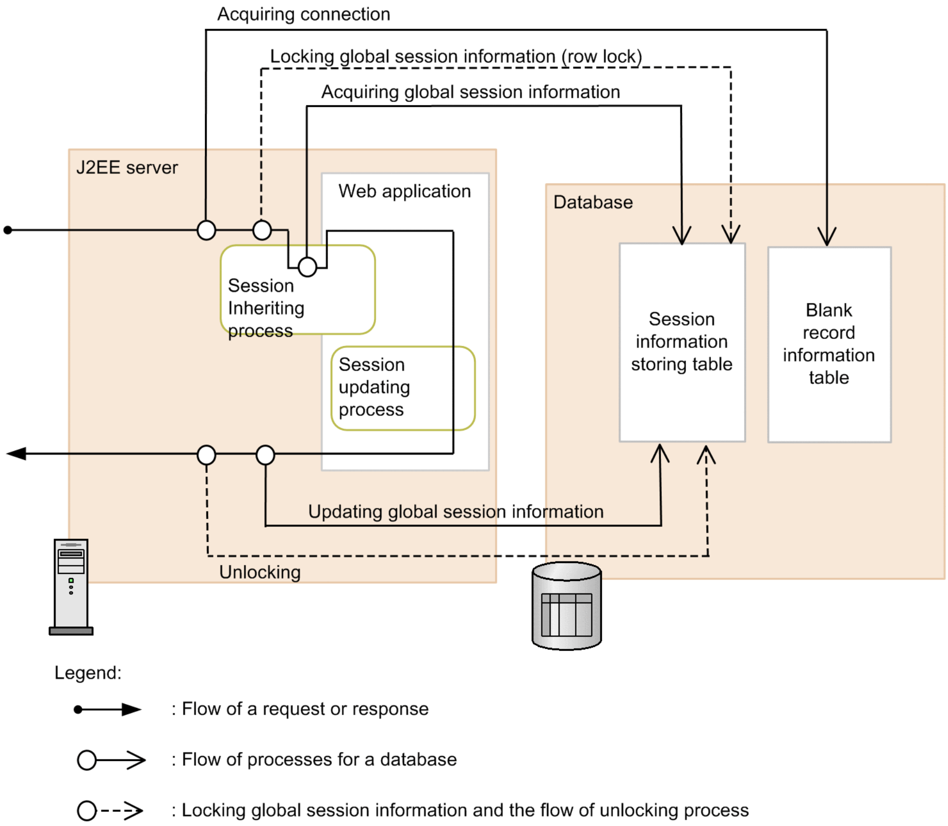
1. Receive a request indicating that an HTTP session needs to be deleted.
   If integrity mode is enabled, global session information in the database is locked before executing a Web application.
2. Along with deleting the session in a Web application, the HTTP session is deleted.
3. The global session information and the blank record information in the database are deleted when the HTTP session is deleted.
   If integrity mode is enabled, the lock is released.

(4) Inheriting session information that uses global session information

If an HTTP session, associated with the received request, does not exist on the J2EE server, an HTTP session is created again by using the global session information in the database. This inherits the session.

The following figure shows the flow of processing executed when inheriting session information that uses global session information.

Figure 6–5: Inheriting session information that uses global session information (the database session failover functionality)



1. If an HTTP session, associated with the received request, does not exist on the J2EE server, an HTTP session is created again on the J2EE server by invoking the global session information in the database.
   The re-created HTTP session inherits the session and executes session update processing in a Web application. The HTTP session is updated as an extension to session update processing.
2. Along with updating of the HTTP session, global session information is updated.

If inheriting of the global session information is successful, the `KDJE34321-I` message is output to the message log. If global session information could not be inherited because the global session information corresponding to the session ID, which is received from the client, does not exist in the database, the `KDJE34325-W` message is output to the message log.

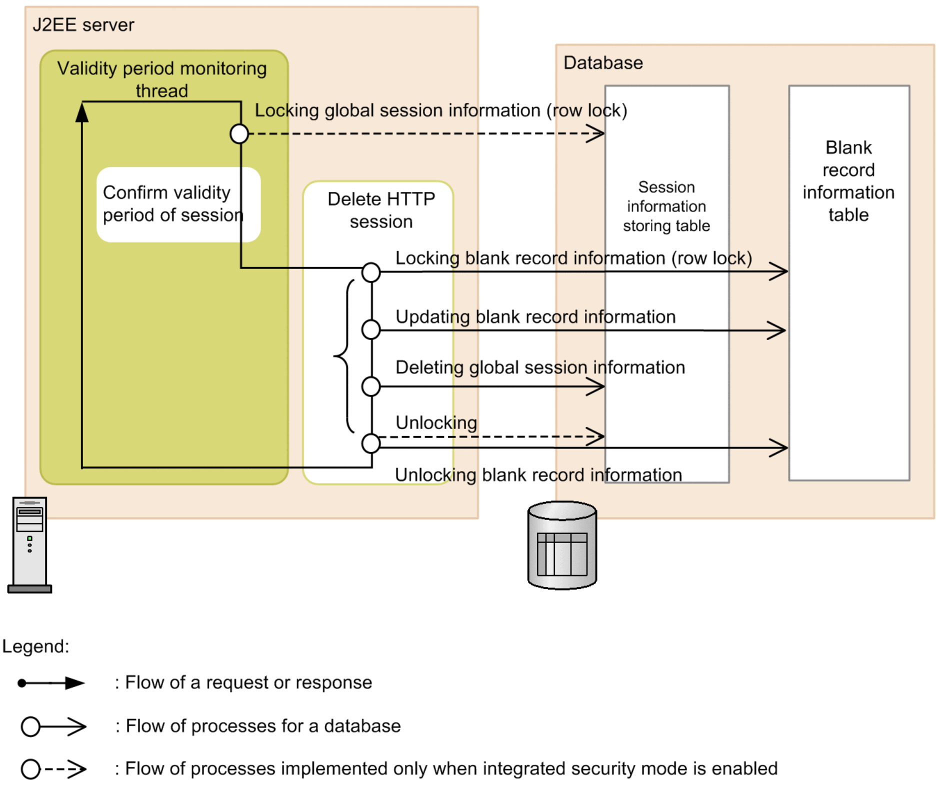
6.4.3 Processing when validity of global session information expires

Each HTTP session has a validity set to it. HTTP sessions that have exceeded validity, found as a result of checking validity on the basis of the information of the last access time, are deleted. When an HTTP session has deleted because it exceeded validity, corresponding global session information is also deleted as an extension of that processing.

Validity monitoring threads existing in web container periodically monitor the validity of HTTP sessions. Validity monitoring threads exist for every Web application.

The following figure shows the flow of processing executed when deleting global session information due to expiry of validity.

Figure 6–6: Processing when validity of global session information expires (the database session failover functionality)



1. If integrity mode is enabled, the global session information corresponding to the session, the validity of which is determined to have expired by using validity monitoring threads, is locked.
2. The HTTP session is deleted as an extension of session deletion processing. The global session information and the blank record information in the database are deleted when the HTTP session is deleted.
   If integrity mode is enabled, the lock is released.

6.4.4 Listeners that operate in association with events occurring in the database session failover functionality

When using the database session failover functionality, the `sessionDidActivate()` method in the `javax.servlet.http.HttpSessionActivationListener` interface is invoked when inheriting of a global session occurs. In that case, the `sessionCreated()` method in the `javax.servlet.http.HttpSessionListener` interface is not invoked.

With processing using an HTTP session, the listeners associated with the HTTP session corresponding to events stipulated in Java EE operate. Listeners associated with an HTTP session are the classes that implement the following interfaces:

- `javax.servlet.http.HttpSessionListener`

- `javax.servlet.http.HttpSessionActivationListener`
- `javax.servlet.http.HttpSessionAttributeListener`
- `javax.servlet.http.HttpSessionBindingListener`

When using the database session failover functionality, listeners associated with an HTTP session operate with events in the database session failover functionality as key factors.

The following table describes mapping among events stipulated in Java EE, events that occur in the database session failover functionality, and listeners that operate with events as key factors.

Table 6–13: Events that occur in the database session failover functionality and listeners to be operated

Sr. No.	Event stipulated in Java EE	Corresponding event (when using the database session failover functionality)	Listener that operates
1	Creating an HTTP session	Creating an HTTP session	The `sessionCreated()` method in the `javax.servlet.http.HttpSessionListener` interface
2	Disabling an HTTP session	• Disabling an HTTP session • Stopping a Web application	• The `sessionDestroyed()` method in the `javax.servlet.http.HttpSessionListener` interface • The `attributeRemoved()` method in the `javax.servlet.http.HttpSessionAttributeListener` interface# • The `valueUnbound()` method in the `javax.servlet.http.HttpSessionBindingListener` interface#
3	Adding HTTP session attributes	Adding HTTP session attributes	• The `attributeAdded()` method in the `javax.servlet.http.HttpSessionAttributeListener` interface • The `valueBound()` method in the `javax.servlet.http.HttpSessionBindingListener` interface
4	Changing HTTP session attributes	Changing HTTP session attributes	The `aattributeReplaced()` method in the `javax.servlet.http.HttpSessionAttributeListener` interface
5	Deleting HTTP session attributes	• Deleting HTTP session attributes • Stopping a Web application	• The `attributeRemoved()` method in the `javax.servlet.http.HttpSessionAttributeListener` interface • The `valueUnbound()` method in the `javax.servlet.http.HttpSessionBindingListener` interface
6	Activating a session	Inheriting global session	The `sessionDidActivate()` method in the `javax.servlet.http.HttpSessionActivationListener` interface
7	Deactivating a session	(no corresponding event)	(No listener operates)

Case when attributes were added when the event occurred.

Other listeners operate in the same way as cases in which the database session failover functionality is not used.

6.4.5 Locking global session information (when integrity mode is enabled)

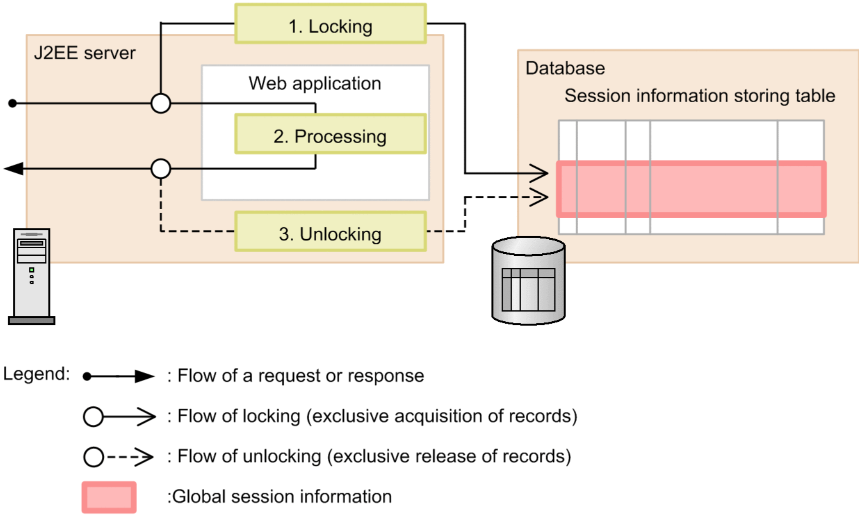
This subsection describes locking of global session information executed only when integrity mode is enabled. The operation is not executed when integrity mode is disabled.

With a system in which a J2EE server is replicated, requests having the same session ID might be concurrently sent to a different J2EE server. For example, if you access a page of frame structure or a page containing multiple images (`image` tag), browser functionality sends requests to the servers with multi-threads.

If information of the same global session is updated on different J2EE servers, the integrity of global session information is lost. As a result, with the database session failover functionality, acquire and control the exclusion of records, in which global session information under updating are stored, so that the records cannot be used on another server. This processing of acquiring exclusion is called *locking* of global session information. The processing of releasing the exclusion is called *releasing lock*.

The following figure shows locking of global session information by performing record exclusion.

Figure 6–7: Locking global session information by performing record exclusion



This subsection describes the processing executed when locking global session information by performing record exclusion. Sr. No. corresponds to the numbers in the figure.

1. On receiving a request from the client, global session information in the database is locked.
2. Web application processing is executed after locking.
3. Global session information is unlocked when the processing of a Web application is complete.

Thus, the global session information in the database is locked during the Web application operation and this assures that requests having the same session ID in the system do not process concurrently.

When a request is sent to a J2EE server, global session information is locked irrespective of whether the HTTP session is to be used in the Web application. However, global session information is not locked for the following requests:

- Requests for which an HTTP session is not created
- Requests having URL containing extensions or URIs that have inhibited the database session failover functionality
  By default, `txt`, `htm`, `html`, `jpg`, `gif`, and `js` are the extensions targeted for inhibiting. There are no URIs to be inhibited by default. For details on inhibiting the database session failover functionality, see *5.6.1 Inhibiting the session failover functionality*.

With the database session failover functionality, the lock of global session is enabled also between the threads sent to one J2EE server and not to different J2EE servers. If you send multiple requests having the same session ID to one J2EE server, requests are processed one by one in the order of receiving. Processing of a request received later starts after waiting for completion of processing of the request sent earlier.

> **Important note**
>
> A Web client might send multiple requests having the same session ID because contents that use frames and combine multiple dynamic pages that use HTTP sessions are updated. In this case, requests are processed one by one in the order of receiving. As a result, processing performance might degrade compared with the case of not using the database session failover functionality.

(1) Result of invoking lock acquisition processing when acquiring a lock

The result of invoking lock acquisition processing varies depending on the state of global session information in the database. The following table describes the relation between global session information state and the result of invoking lock acquisition processing.

Table 6–14: Relation between global session information state and the result of invoking lock acquisition processing

Sr. No.	State of global session information in the database	Result of invoking lock acquisition processing	Output message
1	Exists and not locked (in normal cases).	Global session information in the database is locked (ends successfully).	Not output
2	Does not exist.	It is determined that a session disabled due to timeout or a session having invalid session ID is targeted. Hence, HTTP session in the J2EE server is deleted. As a result, a Web application is executed in the state of not having an HTTP session.	`KDJE34315-W`
3	Session exists, but it is updated on another J2EE server and is newer than the information of the HTTP session on the J2EE server.	It is determined that it is global session information used on another J2EE server. Hence, the contents of global session information in the database are inherited (inheriting a session[#1] occurs).	`KDJE34322-I`[#2]
4	Exists and locked because it is being used.	Waiting for lock [#3] occurs. A lock is acquired after processing of the request that uses an HTTP session ends and the Web application starts.	Not output

#1

For details on inheriting a session, see *6.4.2 (4) Inheriting session information that uses global session information*.

#2

Output at Warning level.

#3

For details on waiting for lock, see *(2) Waiting for lock*.

(2) Waiting for lock

If you receive a request that uses an HTTP session in a global session that is targeted for locking, you must wait for acquiring the lock. The state of waiting for acquiring a lock is called *waiting for lock* in global session information. The timeout that results because of waiting for a lock is called a *lock timeout*.

The following table describes the relation between global session information states after waiting for a lock occurs and as a result of invoking lock acquisition processing.

Table 6–15: Relation between global session information states after waiting for lock occurs and as a result of invoking lock acquisition processing

Sr. No.	State of global session information after waiting for lock occurs	Result of invoking lock acquisition processing after waiting for lock occurs	Output message
1	Request processing, which was using the session earlier, ends and the lock is released.	Global session information in the database is locked (ends successfully).	Not output
2	Timeout time elapsed, but the lock is not released (lock timeout occurred)#1.	When session acquisition processing is executed in a Web application, `com.hitachi.software.web.dbsfo.DatabaseAccessException`#2 is thrown.	`KDJE34312-W`
3	A failure occurred in the database while waiting for lock and lock is not released. (lock timeout occurred).	When session acquisition processing is executed in a Web application, `com.hitachi.software.web.dbsfo.DatabaseAccessException`#2 is thrown.	`KDJE34312-W`

#1

This state includes the case when an SQL statement used for locking is sent to the database and a timeout occurs due to a failure in the communication path.

#2

The `DatabaseAccessException` class inherits the `java.lang.IllegalStateException` class. For details on the `DatabaseAccessException` class, see *3.1 Exception classes* in the *uCosminexus Application Server API Reference Guide.*

(3) Locking global session information when a failure occurs on a J2EE server

If an OS hangs or a network failure occurs on a J2EE server, on which a Web application is running, the global session information, which is locked in the database, might be temporarily locked.

For recovering the session information from the locked state, you need to take one of the following measures:

- Perform settings in the database for monitoring connections from a client and recovering by detecting disconnection.
  If you perform these settings, the database functionality detects disconnection from a J2EE server and automatically releases the lock after a certain period of time. Also, the state is returned to the state before acquiring the lock when a disconnection is detected. If you use HiRDB, set the functionality for monitoring UAP processing time. For details on the functionality for monitoring UAP processing time, see the *HiRDB UAP Development Guide*.
- Perform regular maintenance of the database.

For details on setup and operation of a database, see the manual of the database used.

6.4.6 Operations performed when a failure occurs during global session information operation

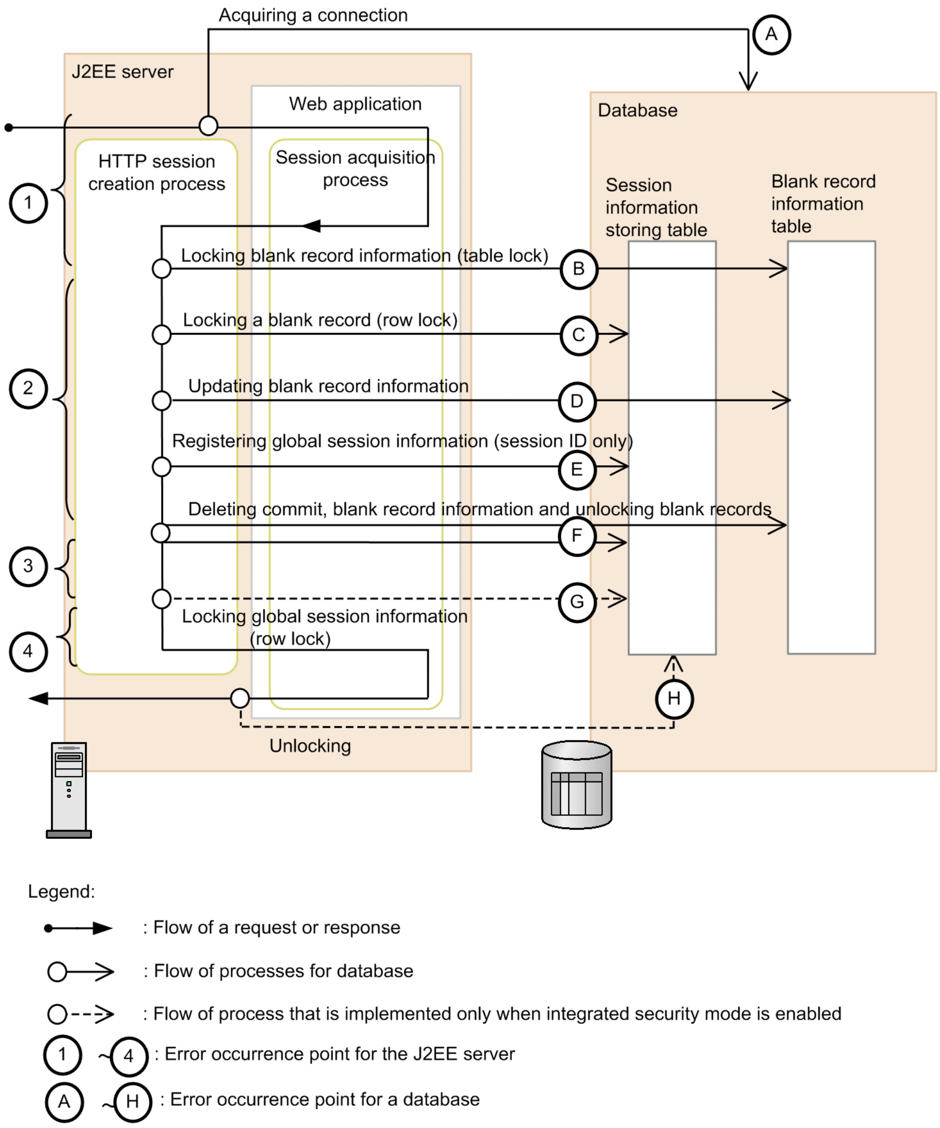
This section describes the operations performed when a failure occurs during global session information operations. This section describes the points of failure, state of session, impact on other requests, and output messages for every operation of global session information.

(1) Operations performed when a failure occurs while creating global session information

This subsection describes the operations performed when a J2EE server failure or database failure occurs while creating global session information.

The following figure shows the flow of processing for creating global session information and points of failure.

Figure 6–8: Flow of processing of creating global session information and points of failure



In the description below, numbers (failure points of the J2EE server) and alphabets (failure points of the database) in the figure are mapped with numbers or alphabets of failure points in the table.

(a) Operations performed when a failure occurs on a J2EE server (process down)

The following table describes the operations performed when a J2EE server failure occurs and process goes down while creating global session information.

Table 6–16: Operations performed when a failure occurs on a J2EE server (process down)

Failure point	State of session		Impact on other requests
	HTTP session on J2EE server	Global session information	
1	Not created	Not created	None
2	Not created	Not created (rolled back)#1	You cannot newly create all HTTP sessions until the database detects client connection
3	Not created	Created#2	None
4	Disappears due to process down	Created#2	None

#1

`SQLException` occurs and rolls back to the state before receiving the request.

#2

You cannot inherit global session information in this state. If validity expires, validity monitoring deletes the session information.

(b) Operations performed when a failure occurs in a database (if SQLException occurs)

The following table describes the operations performed when a database failure occurs and `SQLException` occurs while creating global session information. Operations vary when integrity mode is enabled and disabled.

Table 6–17: Operations performed when a database fails and SQLException occurs (when integrity mode is disabled)

Failure point	State of session		Impact on other requests	Web application operation	Message
	HTTP session on J2EE server	Global session information			
A	Reduced and created#1	Not created	None	Ends successfully	`KDJE34368-W`
B - F	Reduced and created#1	Not created (rolled back)#2	None	Ends successfully	`KDJE34368-W`
G	--	--	--	--	--
H	--	--	--	--	--

Legend:

--: Not applicable

#1

The reduced HTTP session is updated in the database in the processing of updating global session information at the time of receiving a request for the next time.

#2

`SQLException` occurs and rolls back to the state before receiving the request.

Table 6–18: Operations performed when a database failure and SQLException occurs (when integrity mode is enabled)

Failure point	State of session		Impact on other requests	Web application operation	Message
	HTTP session on J2EE server	Global session information			
A	Not created	Not created	None	An exception occurs while acquiring HTTP session#1	`KDJE34314-W`
B - F	Not created	Not created (rolled back)#2	None	An exception occurs while acquiring HTTP session#1	`KDJE34312-W`
G	Not created	Created#3	None	An exception occurs while acquiring HTTP session#1	`KDJE34312-W`
H	Not created (deleted)	Created#3	None	--	`KDJE34312-W`

Legend:

--: Not applicable

#1

`com.hitachi.software.web.dbsfo.DatabaseAccessException` occurs when invoking the `getSession` method in the `javax.servlet.http.HttpServletRequest` interface in case of Servlet and before executing user code in the case of JSP.

#2

`SQLException` occurs and rolls back to the state before receiving the request.

#3

You cannot inherit global session information in this state. If validity expires, validity monitoring deletes the session information.

(c) Operations performed when a failure occurs in a database (when a database is not responding or slows down)

The following table describes the operations performed when a database failure occurs, and the database is not responding or slows down while creating global session information. Operations vary when integrity mode is enabled and disabled.

Table 6–19: Operations performed when a database failure occurs and the database is not responding or slows down (when integrity mode is disabled)

Failure point	State of session		Impact on other requests	Web application operation	Message
	HTTP session on J2EE server	Global session information			
A	Reduced and created#1	Not created	None	Ends successfully	`KDJE34368-W`
B - F	Reduced and created#1	Not created (rolled back)#2	You cannot newly create all HTTP sessions until a timeout occurs by waiting for lock release.	Ends successfully	`KDJE34368-W`
G	--	--	--	--	--
H	--	--	--	--	--

Legend:

--: Not applicable

#1

The reduced HTTP session is updated in the database in the processing of updating global session information at the time of receiving a request for the next time.

#2

A timeout occurs because of waiting for database lock release and the state rolls back to the state before receiving the request.

Table 6–20: Operations performed when a database failure occurs and the database is not responding or slows down (when integrity mode is enabled)

Failure point	State of session		Impact on other requests	Web application operation	Message
	HTTP session on J2EE server	Global session information			
A	Not created	Not created	None	An exception occurs while acquiring HTTP session#1	`KDJE34314-W`
B - F	Not created	Not created (rolled back)#2	You cannot newly create all HTTP sessions until a timeout occurs by waiting for lock release.	An exception occurs while acquiring HTTP session#1	`KDJE34312-W`
G	Not created	Created#3	None	An exception occurs while acquiring HTTP session#1	`KDJE34312-W`
H	Not created (deleted)	Created#3	None	--	`KDJE34312-W`

Legend:

--: Not applicable

#1

`com.hitachi.software.web.dbsfo.DatabaseAccessException` occurs when invoking the `getSession` method in the `javax.servlet.http.HttpServletRequest` interface in the case of Servlet and before executing user code in the case of JSP.

#2

A timeout occurs because of waiting for database unlocking and rolls back to the state before receiving the request.

#3

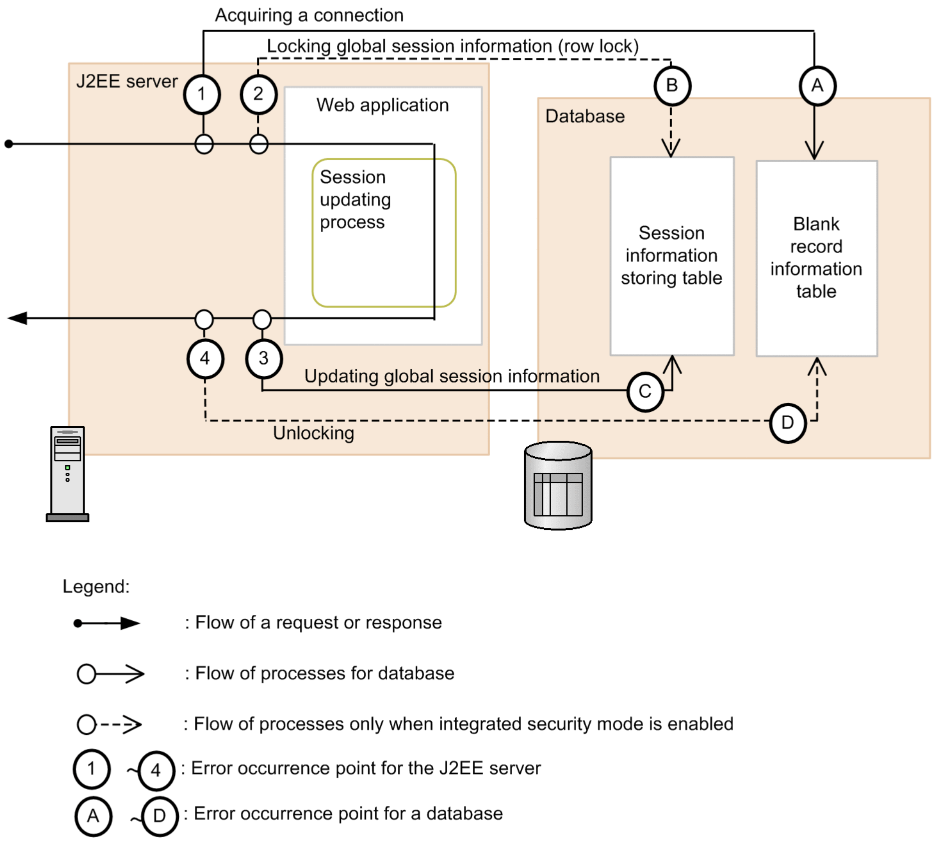
You cannot inherit global session information in this state. If validity expires, validity monitoring deletes the session information.

(2) Operations performed when a failure occurs while updating global session information

This subsection describes the operations performed when a J2EE server failure or database failure occurs while updating global session information.

The following figure shows the flow of processing of updating global session information and points of failure.

Figure 6–9: Flow of processing of updating global session information and points of failure



(a) Operations performed when a failure occurs on a J2EE server (process down)

The following table describes the operations performed when a J2EE server failure occurs and a process goes down while updating global session information.

Table 6–21: Operations performed when a failure occurs on a J2EE server (process down)

Failure point	State of session		Impact on other requests
	HTTP session on J2EE server	Global session information	
1	Disappears due to process down	Not updated	None
2	Disappears due to process down	Not updated (rolled back)#	You cannot perform operations of the relevant HTTP sessions until the database detects client connection
3	Disappears due to process down	Not updated (rolled back)#	You cannot perform operations of the relevant HTTP sessions until the database detects client connection
4	Disappears due to process down	Not updated (rolled back)#	You cannot perform operations of the relevant HTTP sessions until the database detects client connection

#

`SQLException` occurs and rolls back to the state before receiving the request.

(b) Operations performed when a failure occurs in a database (if SQLException occurs)

The following table describes the operations performed when a database failure occurs and `SQLException` occurs while updating global session information. Operations vary when integrity mode is enabled and disabled.

Table 6–22: Operations performed when a database failure and SQLException occurs (when integrity mode is disabled)

Failure point	State of session		Impact on other requests	Web application operation	Message
	HTTP session on J2EE server	Global session information			
A	Reduced and updated#	Not updated	None	Ends successfully	`KDJE34368-W`
B	--	--	--	--	--
C	Reduced and updated#	Not updated	None	--	`KDJE34368-W`
D	--	--	--	--	--

Legend:

--: Not applicable

#

The reduced HTTP session is updated in the database in the process of updating global session information at the time of receiving a request for the next time.

Table 6–23: Operations performed when a database failure and SQLException occurs (when integrity mode is enabled)

Failure point	State of session		Impact on other requests	Web application operation	Message
	HTTP session on J2EE server	Global session information			
A	Not updated	Not updated	None	An exception occurs while acquiring HTTP session#1	`KDJE34314-W`
B	Not updated (deleted)	Not updated (rolled back)#2	None	An exception occurs while acquiring HTTP session#1	`KDJE34312-W`
C	Not updated (deleted)	Not updated (rolled back)#2	None	--	`KDJE34312-W`
D	Not updated (deleted)	Not updated (rolled back)#2	None	--	`KDJE34312-W`

Legend:

--: Not applicable

#1

`com.hitachi.software.web.dbsfo.DatabaseAccessException` occurs when invoking the `getSession` method in the `javax.servlet.http.HttpServletRequest` interface in the case of Servlet and before executing user code in the case of JSP.

#2

`SQLException` occurs and rolls back to the state before receiving the request.

(c) Operations performed when a failure occurs in a database (when database is not responding or slows down)

The following table describes the operations performed when a database failure occurs, and the database is not responding or slows down while updating global session information. Operations vary when integrity mode is enabled and disabled.

Table 6–24: Operations performed when a database failure occurs and the database not responding or slows down (when integrity mode is disabled)

Failure point	State of session		Impact on other requests	Web application operation	Message
	HTTP session on J2EE server	Global session information			
A	Reduced and updated#1	Not updated	None	Ends successfully	KDJE34368-W
B	--	--	--	--	--
C	Reduced and updated#1	Not updated (rolled back)#2	You cannot perform operations of the relevant HTTP sessions until a timeout occurs by waiting for lock release	--	KDJE34368-W
D	--	--	--	--	--

Legend:

--: Not applicable

#1

Reduced HTTP session is updated in the database in the processing of updating global session information at the time of receiving a request for the next time.

#2

A timeout occurs because of waiting for database unlocking and rolls back to the state before receiving the request.

Table 6–25: Operations performed when a database failure occurs and the database is not responding or slows down (when integrity mode is enabled)

Failure point	State of session		Impact on other requests	Web application operation	Message
	HTTP session on J2EE server	Global session information			
A	Not updated	Not updated	None	An exception occurs while acquiring HTTP session#1	KDJE34314-W
B	Not updated (deleted)	Not updated (rolled back)#2	You cannot perform operations of the relevant HTTP sessions until a timeout occurs by waiting for lock release	An exception occurs while acquiring HTTP session#1	KDJE34312-W
C	Not updated (deleted)	Not updated (rolled back)#2	You cannot perform operations of the relevant HTTP sessions until a timeout occurs by waiting for lock release	--	KDJE34312-W
D	Not updated (deleted)	Not updated (rolled back)#2	You cannot perform operations of the relevant HTTP sessions until a	--	KDJE34312-W

Failure point	State of session		Impact on other requests	Web application operation	Message
	HTTP session on J2EE server	Global session information			
			timeout occurs by waiting for lock release		

Legend:

--: Not applicable

#1

`com.hitachi.software.web.dbsfo.DatabaseAccessException` occurs when invoking the `getSession` method in the `javax.servlet.http.HttpServletRequest` interface in the case of Servlet and before executing user code in the case of JSP.

#2

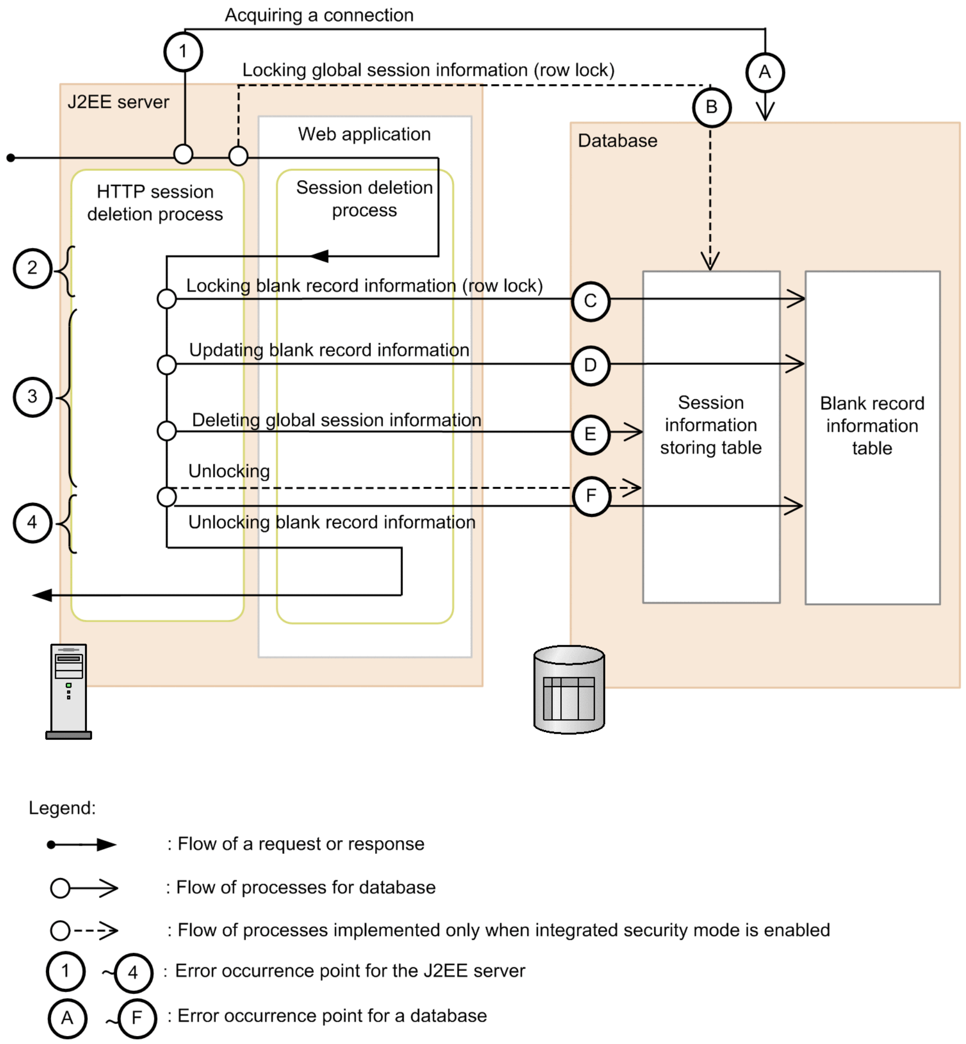
A timeout occurs because of waiting for database lock release and rolls back to the state before receiving the request.

(3) Operations performed when a failure occurs while deleting global session information

This subsection describes the operations performed when a J2EE server failure or database failure occurs while deleting global session information.

The following figure shows the flow of processing of deleting global session information and points of failure.

Figure 6–10: Flow of processing of deleting global session information and points of failure



(a) Operations performed when a failure occurs on a J2EE server (process down)

The following table describes the operations performed when a J2EE server failure occurs and process goes down while deleting global session information.

Table 6–26: Operations performed when a failure occurs on a J2EE server (process down)

Failure point	State of session		Impact on other requests
	HTTP session on J2EE server	Global session information	
1	Disappears due to process down	Not deleted	None
2	Disappears due to process down	Not deleted (rolled back)#	You cannot perform operations of the relevant HTTP sessions until the database detects client connection
3	Disappears due to process down	Not deleted (rolled back)#	You cannot perform operations of the relevant HTTP sessions until the database detects client connection
4	Disappears due to process down	Deleted	None

#

`SQLException` occurs and rolls back to the state before receiving the request.

(b) Operations performed when a failure occurs in a database (if SQLException occurs)

The following table describes the operations performed when a database failure occurs and `SQLException` occurs while deleting global session information. Operations vary when integrity mode is enabled and disabled.

Table 6–27: Operations performed when a database failure and SQLException occurs (when integrity mode is disabled)

Failure point	State of session		Impact on other requests	Web application operation	Message
	HTTP session on J2EE server	Global session information			
A	Deleted	Not deleted	None	An exception occurs while acquiring HTTP session#1	`KDJE34377-E`#2
B	--	--	--	--	--
C - F	Deleted	Not deleted (rolled back)#3	None	An exception occurs when disabling HTTP session#4	`KDJE34377-E`#2

Legend:

--: Not applicable

#1

`com.hitachi.software.web.dbsfo.DatabaseAccessException` occurs when invoking the `getSession` method in the `javax.servlet.http.HttpServletRequest` interface in the case of Servlet and before executing user code in the case of JSP.

#2:

A message is output only when the first failure occurs. Thereafter, messages are not output for the same failure until you restart the Web application.

#3

`SQLException` occurs and rolls back to the state before receiving the request.

#4

`com.hitachi.software.web.dbsfo.DatabaseAccessException` occurs when invoking the `invalidate` method in the `javax.servlet.http.HttpServletRequest` interface in the case of Servlet and when invoking the `invalidate` method of implicit object session in the case of JSP.

Table 6–28: Operations performed when a database failure and SQLException occurs (when integrity mode is enabled)

Failure point	State of session		Impact on other requests	Web application operation	Message
	HTTP session on J2EE server	Global session information			
A	Not deleted	Not deleted	None	An exception occurs while acquiring HTTP session#1	`KDJE34314-W`
B	Deleted	Not deleted (rolled back)#2	None	An exception occurs while acquiring HTTP session#1	`KDJE34312-W`
C - F	Deleted	Not deleted (rolled back)#2	None	An exception occurs when disabling HTTP session#3	`KDJE34312-W`

#1

`com.hitachi.software.web.dbsfo.DatabaseAccessException` occurs when invoking the `getSession` method in the `javax.servlet.http.HttpServletRequest` interface in the case of Servlet and before executing user code in the case of JSP.

#2

`SQLException` occurs and rolls back to the state before receiving the request.

#3

`com.hitachi.software.web.dbsfo.DatabaseAccessException` occurs when invoking the `invalidate` method in the `javax.servlet.http.HttpServletRequest` interface in case of Servlet and when invoking the `invalidate` method of implicit object session in case of JSP.

(c) Operations performed when a failure occurs in a database (when database is not responding or slows down)

The following table describes the operations performed when a database failure occurs, and database is not responding or slows down while deleting global session information. Operations vary when integrity mode is enabled and disabled.

Table 6–29: Operations performed when a database failure occurs, and database is not responding or slows down (when integrity mode is disabled)

Failure point	State of session		Impact on other requests	Web application operation	Message
	HTTP session on J2EE server	Global session information			
A	Deleted	Not deleted	None	An exception occurs while acquiring HTTP session#1	`KDJE34377-E`#2
B	--	--	--	--	--
C - F	Deleted	Not deleted (rolled back)#3	You cannot perform operations of the relevant HTTP sessions until a timeout occurs by waiting for lock release	An exception occurs when disabling HTTP session#4	`KDJE34377-E`#2

Legend:

--: Not applicable

#1

`com.hitachi.software.web.dbsfo.DatabaseAccessException` occurs when invoking the `getSession` method in the `javax.servlet.http.HttpServletRequest` interface in the case of Servlet and before executing user code in the case of JSP.

#2:

A message is output only when the first failure occurs. Thereafter, messages are not output for the same failure until you restart the Web application.

#3

A timeout occurs because of waiting for database unlocking and rolls back to the state before receiving the request.

#4

`com.hitachi.software.web.dbsfo.DatabaseAccessException` occurs when invoking the `invalidate` method in the `javax.servlet.http.HttpServletRequest` interface in the case of Servlet and when invoking the `invalidate` method of implicit object session in the case of JSP.

Table 6–30: Operations performed when a database failure occurs and the database is not responding or slows down (when integrity mode is enabled)

Failure point	State of session		Impact on other requests	Web application operation	Message
	HTTP session on J2EE server	Global session information			
A	Not deleted	Not deleted	None	An exception occurs while acquiring HTTP session#1	KDJE34314-W
B	Deleted	Not deleted (rolled back)#2	You cannot perform operations of the relevant HTTP sessions until a timeout occurs by waiting for lock release	An exception occurs while acquiring HTTP session#1	KDJE34312-W
C - F	Deleted	Not deleted (rolled back)#2	You cannot perform operations of the relevant HTTP sessions until a timeout occurs by waiting for lock release	An exception occurs when disabling HTTP session#3	KDJE34312-W

#1

`com.hitachi.software.web.dbsfo.DatabaseAccessException` occurs when invoking the `getSession` method in the `javax.servlet.http.HttpServletRequest` interface in the case of Servlet and before executing user code in the case of JSP.

#2

A timeout occurs because of waiting for database lock release and the state rolls back to the state before receiving the request.

#3

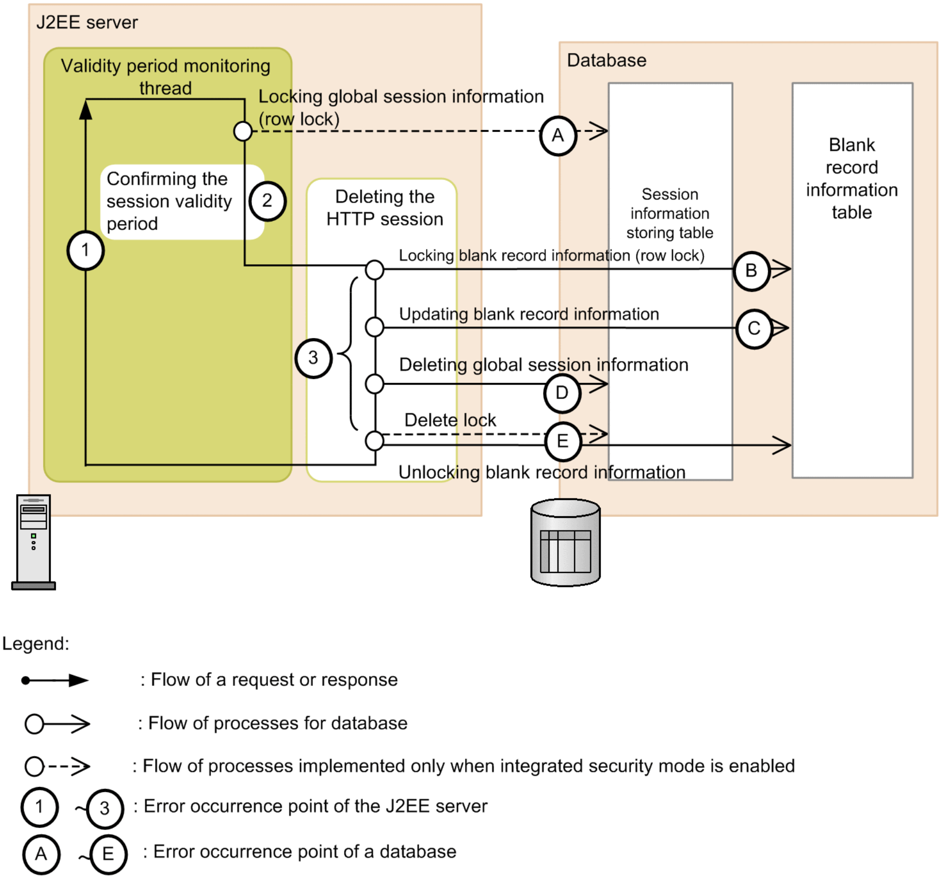
`com.hitachi.software.web.dbsfo.DatabaseAccessException` occurs when invoking the `invalidate` method in the `javax.servlet.http.HttpServletRequest` interface in the case of Servlet and when invoking the `invalidate` method of implicit object session in the case of JSP.

(4) Operations performed when a failure occurs while deleting global session information due to expiry of validity

This subsection describes the operations performed when a J2EE server failure or database failure occurs while deleting global session information due to the expiration of validity.

The following figure shows the flow of processing of deleting global session information due to expiry of validity and points of failure.

Figure 6–11: Flow of processing of deleting global session information due to expiry of validity and points of failure



(a) Operations performed when a failure occurs on a J2EE server (process down)

The following table describes the operations performed when a J2EE server failure occurs and process goes down while deleting global session information due to expiry of validity.

Table 6–31: Operations performed when a failure occurs on a J2EE server (process down)

Failure point	State of session		Impact on other requests
	HTTP session on J2EE server	Global session information	
1	Disappears due to process down	Not deleted	None
2 and 3	Disappears due to process down	Not deleted (rolled back)	You cannot perform operations of the relevant HTTP sessions until the database detects client disconnection

(b) Operations performed when a failure occurs in a database (if SQLException occurs)

The following table describes the operations performed when a database failure occurs and `SQLException` occurs while deleting global session information due to expiry of validity. Operations vary when integrity mode is enabled and disabled.

Table 6–32: Operations performed when a database failure and SQLException occurs (when integrity mode is disabled)

Failure point	State of session		Impact on other requests	Web application operation	Message
	HTTP session on J2EE server	Global session information			
A	--	--	--	--	--
B - E	Deleted	Not deleted (rolled back)	None	--	KDJE34377-E#

Legend:

--: Not applicable

#

A message is output only when the first failure occurs. Thereafter, messages are not output for the same failure until you restart the Web application.

Table 6–33: Operations performed when a database failure and SQLException occurs (when integrity mode is enabled)

Failure point	State of session		Impact on other requests	Web application operation	Message
	HTTP session on J2EE server	Global session information			
A	Deleted	Not deleted (rolled back)	None	--	KDJE34336-W
B - E	Deleted	Not deleted (rolled back)	None	--	KDJE34312-W

Legend:

--: Not applicable

(c) Operations performed when a failure occurs in a database (when database is not responding or slows down)

The following table describes the operations performed when a database failure occurs and the database is not responding or slows down while deleting global session information due to expiry of validity. Operations vary when integrity mode is enabled and disabled.

Table 6–34: Operations performed when a database failure occurs and the database is not responding or slows down (when integrity mode is disabled)

Failure point	State of session		Impact on other requests	Web application operation	Message
	HTTP session on J2EE server	Global session information			
A	--	--	--	--	--
B - E	Deleted	Not deleted (rolled back)	You cannot perform operations of the relevant HTTP sessions until a timeout occurs by waiting for lock release	--	KDJE34377-E#

Legend:

--: Not applicable

#

A message is output only when the first failure occurs. Thereafter, messages are not output for the same failure until you restart the Web application.

Table 6–35: Operations performed when a database failure occurs and the database is not responding or slows down (when integrity mode is enabled)

Failure point	State of session		Impact on other requests	Web application operation	Message
	HTTP session on J2EE server	Global session information			
A	Deleted	Not deleted (rolled back)	You cannot perform operations of the relevant HTTP sessions until a timeout occurs by waiting for lock release	--	KDJE34336-W
B - E	Deleted	Not deleted (rolled back)	You cannot perform operations of the relevant HTTP sessions until a timeout occurs by waiting for lock release	--	KDJE34312-W

Legend:

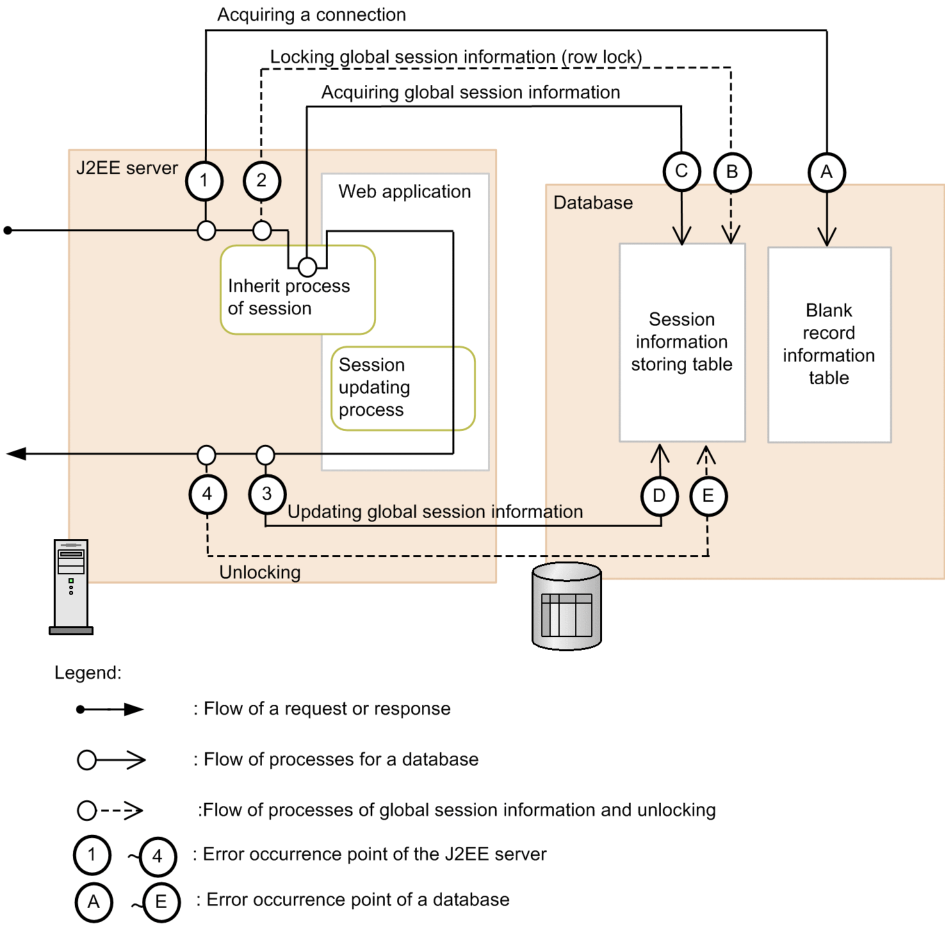
--: Not applicable

(5) Operations performed when a failure occurs while inheriting global session by using global session information

This subsection describes the operations performed when a J2EE server failure or database failure occurs while inheriting global session by using global session information.

The following figure shows the flow of processing of inheriting global session by using global session information and points of failure.

Figure 6–12: Flow of processing of inheriting global session by using global session information and points of failure



(a) Operations performed when a failure occurs on a J2EE server (process down)

The operations performed when a J2EE server failure occurs and process goes down while inheriting global session by using global session information are the same as when a J2EE server failure occurs while updating global session information.

For details on operations performed when a J2EE failure occurs while updating a global session information, see the operations performed when a J2EE failure occurs described in *(2) Operations performed when a failure occurs while updating global session information*.

(b) Operations performed when a failure occurs in a database (if SQLException occurs)

The following table describes the operations performed when a database failure occurs and `SQLException` occurs during C processing in the figure while inheriting a global session by using global session information. The operations performed when a failure occurs during A, B, D, and E processing in the figure are the same as the operations performed when `SQLException` occurs in a database while updating global session information.

For details on the operations performed when a database failure and `SQLException` occurs while updating global session information, see the operations performed when a database failure occurs (if `SQLException` occurs) described in *(2) Operations performed when a failure occurs while updating global session information*.

Table 6–36: Operations performed when a database failure and SQLException occurs

Failure point	State of session		Impact on other requests	Web application operation	Message
	HTTP session on J2EE server	Global session information			
C	Not inherited	Not inherited	None	An exception occurs while acquiring HTTP session#	`KDJE34314-W`

#

`com.hitachi.software.web.dbsfo.DatabaseAccessException` occurs when invoking the `getSession` method in the `javax.servlet.http.HttpServletRequest` interface in the case of Servlet and before executing user code in the case of JSP.

(c) Operations performed when a failure occurs in a database (when database is not responding or slows down)

The following table describes the operations performed when a database failure occurs and the database is not responding or slows down during C processing in the figure while inheriting global session by using global session information. The operations performed when a failure occurs during A, B, D, and E processing in the figure are the same as the operations performed when a database is not responding or slows down while updating global session information.

For details on the operations performed when a database failure occurs and the database is not responding or slows down while updating global session information, see the operations performed when a database failure occurs (when database is not responding or slows down) described in *(2) Operations performed when a failure occurs while updating global session information*.

Table 6–37: Operations performed when a database failure occurs and the database is not responding or slows down

Failure point	State of session		Impact on other requests	Web application operation	Message
	HTTP session on J2EE server	Global session information			
C	Not inherited	Not inherited	You cannot perform operations of the relevant HTTP sessions until a timeout occurs by waiting for lock release	An exception occurs while acquiring HTTP session#	`KDJE34314-W`

#

`com.hitachi.software.web.dbsfo.DatabaseAccessException` occurs when invoking the `getSession` method in the `javax.servlet.http.HttpServletRequest` interface in the case of Servlet and before executing user code in the case of JSP.

6.5 Definitions in cosminexus.xml

Specify definitions for using the database session failover functionality in the `<war>` tag in `cosminexus.xml`.

The following table describes the definitions of the database session failover functionality in `cosminexus.xml`.

Table 6–38: Definitions of the database session failover functionality in cosminexus.xml

Item	Tag to be specified	Settings
Setting of the database session failover functionality	`http-session-dbsfo-enabled`	In the unit of a Web application, set whether to enable the database session failover functionality.
Upper limit of the number of `HttpSession` objects	`http-session-http-session-max-number`	Set upper limit of the number of `HttpSession` objects.
Application identifier	`http-session-dbsfo-application-id`	Set application identifier.
Maximum size of HTTP session attribute information	`http-session-dbsfo-attribute-data-size-max`	Set maximum size of HTTP session attribute information included in global session information.
Inhibiting the database session failover functionality by using extension	`http-session-dbsfo-exclude-extensions`	If you enable the database session failover functionality in the unit of a Web application, set extensions that inhibit the database session failover functionality.

For details on the tags to be specified, see *2.2.6 Details of the WAR property* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

6.6 Settings in the execution environment

6.6.1 J2EE server settings

Implement the settings of J2EE server in Easy Setup Definition file. Specify the definitions of the database session failover functionality in the `<configuration>` tag of a logical J2EE server (j2ee-server) in the Easy Setup definition file.

The following table describes the definitions of the database session failover functionality in Easy Setup definition file.

Table 6–39: Definitions of the database session failover functionality in Easy Setup definition file

Item	Parameter to be specified	Settings	Reference location
Setting of the database session failover functionality	`webserver.dbsfo.enabled`	Set in the unit of J2EE server whether to use the database session failover functionality.	--
Specifying optional name of DB Connector	`webserver.dbsfo.connector.name`	Specify optional name of DB Connector to be set in DB Connector. For details on setting optional name of DB Connector, see *6.6.4(2) Specifying optional name of DB Connector*.	--
Setting maximum size of HTTP session attribute information that you can include in global session information	`webserver.dbsfo.attribute_data_size.max`	Set maximum size of HTTP session attribute information that you can include in global session information.	--
Setting of functionality for estimating the size of HTTP session attribute information	`webserver.dbsfo.check_size.mode`	Set whether to use functionality for estimating the size of HTTP session attribute information.	--
Setting integrity mode	`webserver.dbsfo.integrity_mode.enabled`#1	Set whether to enable integrity mode of the database session failover functionality.	--
Setting memory used in serialize processing	--	Consider the memory used in serialize processing and perform tuning of `JavaVM`#2.	--
Setting of inhibiting the database session failover functionality by using extension	`webserver.dbsfo.exclude.extensions`	If you want to use the database session failover functionality in the unit of J2EE server, set the extensions that inhibit database session failover functionality.	--
Setting of Inhibiting the database session failover functionality in the unit of URI	`webserver.dbsfo.exclude.uris`	If you want to use the database session failover functionality in the unit of J2EE server, set the URIs that inhibit the database session failover functionality.	(1)
Setting refer-only requests	`webserver.dbsfo.session_read_only.uris`	Set URI to be treated as refer-only requests.	(2)
Setting server ID addition functionality of `HttpSession`	• `webserver.session.server_id.enabled` • `webserver.session.server_id.value`	Set server ID addition functionality of `HttpSession`. Set different values as server IDs for each replicated J2EE server. #3	--
Setting if pending queues are insufficient when using the functionality for controlling	`webserver.dbsfo.thread_control_queue.enabled`	When the functionality that controls the number of concurrent execution threads in the unit of a Web	--

Item	Parameter to be specified	Settings	Reference location
the number of concurrent execution threads		application is enabled, set operations to be performed when free space in pending queues is insufficient.	
Setting of processing of starting a Web application when negotiation fails	`webserver.dbsfo.negotiation.high_level`	Set whether to continue or cancel the processing of starting a Web application when negotiation fails.	--
Setting of exceptions thrown when using HTTP session in the requests targeted for inhibiting the database session failover functionality	`webserver.dbsfo.exception_type_backcompat`	Set exceptions to be thrown when using HTTP session in the requests targeted for inhibiting the database session failover functionality.	--

Legend:

--: Not applicable

#1

If you specify *false* in the `webserver.dbsfo.integrity_mode.enabled` parameter in the setting of integrity mode, you must specify the `webserver.session.server_id.enabled` parameter and `webserver.session.server_id.value` parameter in the setting of server ID addition functionality of HttpSession.

#2

For details on how to estimate the size of memory used in serialize processing, see *5.8.1 Estimating memory used in serialize processing*. For details on `JavaVM` tuning, see *7. JavaVM Memory Tuning* in the *uCosminexus Application Server System Design Guide.*

#3

In a node switching system consisting of the executing node and standby node, set the same value on both nodes.

For details on the Easy Setup definition file and parameters to be specified, see *4.3 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide.*

(1) Settings for inhibiting th database session failover functionality

This subsection describes how to specify a URI when inhibiting the database session failover functionality in the unit of URI.

(a) For specifying URI

Specify a URI, which includes the context path and starts with the forward slash (/). Do not include path parameter, query, or fragment. Note that you cannot use a semicolon (;) in URI of set values.

If you want to specify multiple URIs, separate them with semicolons (;).

(b) Exact match specification and prefix match specification

You can specify by using one of the following methods:

Exact match specification

The specified URI is targeted for inhibiting the database session failover functionality only if it exactly matches with the request URI.

Specification example (in the case of Easy Setup definition file)

```
:
configuration
  logical-server-typej2ee-server/logical-server-type
    param
      param-namewebserver.dbsfo.exclude.uris</param-name>
```

```
      param-value/examples/test/TestServlet;/examples/test2/TestServlet2/p
aram-value
    /param
configuration
:
```

In this example, the following request URIs are targeted for inhibiting the database session failover functionality:

- `http://host/examples/test/TestServlet`
- `http://host/examples/test/TestServlet?name=value`
- `http://host/examples/test/TestServlet;gsessionid=XXXXXXXXXX`

Prefix match specification

If request URI and prefix match, it is targeted for inhibiting the database session failover functionality.

Specification example (in the case of Easy Setup definition file)

```
:
configuration
  logical-server-typej2ee-server/logical-server-type
    param
      param-namewebserver.dbsfo.exclude.uris/param-name
      param-value/examples/*/param-value
    /param
configuration
:
```

In this example, the following request URIs are targeted for inhibiting the database session failover functionality:

- `http://host/examples/test/TestServlet`
- `http://host/examples/dbsfo/DbsfoServlet?name=value`

Note that URI specification must end with the forward slash and an asterisk (`/*`) in the case of prefix match specification. For example, if you specify the following URI, it is considered for exact match specification and not for prefix match specification.

- `/examples/test*`

(c) Normalizing URI

URIs to be targeted for inhibiting the database session failover functionality must be specified after normalizing. If you specify a URI that is not normalized, the `KDJE34341-W` message is output and the corresponding URI is not targeted for inhibiting.

An example of normalized URI is:

- `/examples/test/servlet/TestServlet`

Examples of URIs that are not normalized, are as follows. These URIs are not targeted for inhibiting.

- `/examples/test/jsp/../servlet/TestServlet`
- `/examples/test/./servlet/TestServlet`

(d) Mapping with URL encode

If you specify a URL encoded URI as a target for inhibiting, requests of URL encoded URL that match with the specified URI, are targeted for inhibiting the database session failover functionality. Similarly, if you specify a URI for which URL encoding is not performed, requests of URL, for which URL encoding is not performed, are targeted for inhibiting the database session failover functionality.

However, if you use URI decode functionality, whether the target URL is to be targeted for inhibiting the database session failover functionality according to URI is determined after decoding is performed. Hence, if a URL encoded URL matches with the URI specified as a target of inhibiting, the URL is targeted for inhibiting the database session failover functionality in the unit of URI.

The following table describes URLs that are targeted for inhibiting depending on enabled/disabled status of URI decode functionality.

Table 6–40: URL that are targeted for inhibiting depending on enabled or disabled state of the URI decode functionality

Property set value	Request URL			
	URI decode function enabled		URI decode function disabled	
	Encoded	Not encoded	Encoded	Not encoded
Encoded	Does not inhibit	Does not inhibit	Inhibits	Does not inhibit
Not encoded	Inhibits	Inhibits	Does not inhibit	Inhibits

Legend:

Inhibits: Inhibits the database session failover functionality (the database session failover functionality is disabled).

Does not inhibit: Does not inhibit the database session failover functionality (the database session failover functionality is enabled).

Encoded: URI that includes URL encoded string.

(Example: /examples/%61/Servlet)

Not encoded: URI that does not include URL encoded string.

(Example: /examples/a/Servlet)

(e) Notes on URI specification

URIs that are set for inhibiting the database session failover functionality by URI, are not checked in negotiation. Therefore, check that URIs set on each J2EE server are the same.

(2) Specifying refer-only requests

This subsection describes how to specify a URI when setting refer-only requests.

(a) Specifying URI

Specify a URI that includes the context path and starts with the forward slash (/). Do not include path parameter, query, or fragment. You can specify up to 512 characters. Note that you cannot use a semicolon (;) in URI of set values.

If you want to specify multiple URIs, separate them with semicolons (;).

(b) Exact match specification and prefix match specification

You can specify by using one of the following methods:

Exact match specification

Only if the specified URI exactly matches with the request URI, it becomes a refer-only request.

Specification example (in the case of Easy Setup definition file)

```
:
configuration
```

```
  logical-server-typej2ee-server/logical-server-type
    param
      param-namewebserver.dbsfo.session_read_only.uris/param-name
      param-value/examples/test/TestServlet;/examples/test2/TestServlet2/p
aram-value
    /param
configuration
:
```

In this example, the following request URIs become refer-only requests:

- `http://host/examples/test/TestServlet`
- `http://host/examples/test/TestServlet?name=value`
- `http://host/examples/test/TestServlet;gsessionid=XXXXXXXXXX`

Prefix match specification

If the prefix matches with the request URI, it becomes a refer-only request.

Specification example (in the case of Easy Setup definition file)

```
:
<configuration>
  logical-server-typej2ee-server/logical-server-type
    param
      param-namewebserver.dbsfo.session_read_only.uris/param-name
      param-value/examples/*/param-value
    /param
configuration
:
```

In this example, the following request URIs become refer-only requests:

- `http://host/examples/test/TestServlet`
- `http://host/examples/dbsfo/DbsfoServlet?name=value`

Note that URI specification must end with the forward slash and an asterisk (/*) in the case of prefix match specification. For example, if you specify the following URI, it is considered for exact match specification and not for prefix match specification.

- `/examples/test*`

(c) Normalizing URI

A URI that you want to make a refer-only request, must be normalized and specified. If you specify a URI that is not normalized, the `KDJE34357-W` message is output and the corresponding URI does not become a refer-only request.

An example of normalized URI is:

- `/examples/test/servlet/TestServlet`

Examples of URIs that are not normalized, are shown below. These URIs do not become refer-only requests.

- `/examples/test/jsp/../servlet/TestServlet`
- `/examples/test/./servlet/TestServlet`

(d) Mapping with URL encode

If you specify a URL encoded URI as a refer-only request, the request of URL encoded URL tjat matches with the specified URI, becomes a refer-only request. Similarly, if you specify a URI that is not to be URL encoded, the request of URL that is not URL encoded becomes a refer-only request.

However, if you use URI decode functionality, whether the target URL is a refer-only request according to URI is determined after decoding is performed. As a result, if a URL encoded URL matches with the URI specified as a refer-only request, the URL becomes a refer-only request in URI unit.

The following table describes URLs that become refer-only requests depending on enabled/disabled status of URI decode functionality.

Table 6–41: URLs that become refer-only requests depending on enabled/disabled status of URI decode functionality

Property set value	Request URL			
	URI decode function enabled		URI decode function disabled	
	Encoded	Not encoded	Encoded	Not encoded
Encoded	Does not become a refer-only request	Does not become a refer-only request	Becomes a refer-only request	Does not become a refer-only request
Not encoded	Becomes a refer-only request	Becomes a refer-only request	Does not become a refer-only request	Becomes a refer-only request

Legend:

Becomes a refer-only request: The request URL becomes a refer-only request.

Does not become a refer-only request: The request URL does not become a refer-only request.

Encoded: URI that includes URL encoded string.

(Example: /examples/%61/Servlet)

Not encoded: URI that does not include URL encoded string.

(Example: /examples/a/Servlet)

6.6.2 Web application settings

Perform the Web application settings in execution environment by using server management commands and property file. For definition of the database session failover functionality, use WAR property files.

The tags to be specified in WAR property file map with `cosminexus.xml`. For details on definitions in `cosminexus.xml`, see *6.5 Definitions in cosminexus.xml*.

6.6.3 Database settings

This section describes table creation and environment setup required for using the database session failover functionality.

Important note

When creating a table, if you make any changes in the template file that are not described here, the operations of the database session failover functionality are not guaranteed.

(1) Permissions required for connecting to a database

You must have permissions to operate database tables. Also, the conditions must be met. This subsection describes permissions and conditions required for operating tables of each database. Here, a user connected to a database is called a *user connected to database*.

- **In HiRDB**

  Here, it is assumed that the user connected to a database performs all operations related to tables used in the database session failover functionality. The user connected to database must have the following permissions and satisfy the following conditions:

  - User must own the schema.
  - User must have CONNECT permissions.
  - In the schema of the user connected to a database, create tables, indexes, and stored procedures used in the database session failover functionality.

  For details on creating database tables, see *6.6.3(2) Creating database tables*. For details on deleting database tables, see *6.8 Deleting database tables*.

- **In Oracle**

  Here, it is assumed that the database administrator creates or deletes database tables used in the database session failover functionality, and that the user connected to a database of the database session failover functionality performs other usual database operations. The user connected to a database must have the following permissions and satisfy the following conditions:

  - User must have CREATE SESSION system permissions.
  - In the schema of the user connected to a database, create tables, indexes, and stored procedures used in the database session failover functionality.

  For details on creating database tables, see *6.6.3(2) Creating database tables*. For details on deleting database tables, see *6.8 Deleting database tables*.

(2) Creating database tables

With the database session failover functionality, you must create three types of tables in a database. The following table describes tables to be created and reference location of creation procedure.

Table 6–42: Tables to be created and reference locations of creation procedure

Table name	Physical name in the database	Reference location of creation procedure
Application information table	SFO_*APPLICATION_ID*_APP_INFO	*6.6.3(3)*
Session information storage table	SFO_*APPLICATION_ID*_SESSIONS	*6.6.3(4)*
Blank record information table	SFO_*APPLICATION_ID*_REC_INFO	

Template files for creating database tables, which are used in the database session failover functionality, are stored in the following locations:

In Windows:

Application-Server-installation-directory\CC\sfo\sql\

In UNIX:

/opt/Cosminexus/CC/sfo/sql/

There are two types of template files for table creation for each database used. The following table describes mapping of databases to be used, files, and types of tables to be created.

Table 6–43: Template files for table creation and tables to be created

Database to be used	Template file	Types of table to be created		
		Application information table	Session information storage table	Blank record information table
HiRDB	`hirdb_create_apptbl.sql`	Y	--	
	`hirdb_create_sessiontbl.sql`	--	Y	
Oracle	`oracle_create_apptbl.sql`	Y	--	
	`oracle_create_sessiontbl.sql`	--	Y	

Legend:

Y: Can be created

--: Cannot be created

The following subsections describe details on template files for each used database.

Register the creator of the table in the user set in DB Connector.

(3) Creating Application information table

Application information table stores settings related to the database session failover functionality set in a Web application.

The procedures for creating the application information table are as follows:

1. Copy the template file to any location.

   A template file is provided as an SQL file used for creating a table. The storage locations of template files for each database used are:

   - Storage location of template files when using HiRDB

     In Windows: *Application-Server-installation-directory*`\CC\sfo\sql\hirdb_create_apptbl.sql`

     In UNIX: `/opt/Cosminexus/CC/sfo/sql/hirdb_create_apptbl.sql`

   - Storage location of template files when using Oracle

     In Windows: *Application-Server-installation-directory*`\CC\sfo\sql\oracle_create_apptbl.sql`

     In UNIX: `/opt/Cosminexus/CC/sfo/sql/oracle_create_apptbl.sql`

2. Edit the template file.

   Edit the template file in accordance with the preference information of a Web application and create an SQL file used for creating a table.

   The following table describes change locations and change contents in a template file.

   Table 6–44: Change locations and change contents in a template file

Change location		Change target	Change contents
HiRDB	Oracle		
• First line • Fifth line	• First line • Fifth line	*APPLICATION_ID*	Change the application identifier of the application to be used.

Change location		Change target	Change contents
HiRDB	Oracle		
None	• First line • Fifth line	*SCHEMA_NAME*	Change the schema name of user connected to database.
Sixth line	Sixth line	*HTTP_SESSION_NO*	Change the number of global session information stored in the database.

3. Execute the created SQL file for table creation.

   For executing the SQL file, use SQL Executer when using HiRDB and SQL*Plus when using Oracle.

(4) Creating session information storage tables and blank record information tables

Session information storage table stores global session information. The blank record information table manages the unused records in the session information storage table. The session information storage table and blank record information table are concurrently created by executing one SQL file for table creation.

The procedures for creating the session information storage table and blank record information table is as follows:

1. Copy the template file to any location.

   A template file is provided as an SQL file used for creating a table. The storage locations for template files for each database used are:

   - Storage location of template files when using HiRDB:

     In Windows: *Application-Server-installation-directory*`\CC\sfo\sql\hirdb_create_sessiontbl.sql`

     In UNIX: `/opt/Cosminexus/CC/sfo/sql/hirdb_create_sessiontbl.sql`

   - Storage location of template files when using Oracle

     In Windows: *Application-Server-installation-directory*`\CC\sfo\sql\oracle_create_sessiontbl.sql`

     In UNIX: `/opt/Cosminexus/CC/sfo/sql/oracle_create_sessiontbl.sql`

2. Edit the template file.

   Edit the template file in accordance with the preference information of a Web application and create an SQL file used for creating a table.

   The following table describes change locations and change contents for each database used in a template file.

   Table 6–45: Change locations and change contents in a template file

Change location		Change target	Change contents
HiRDB	Oracle		
• First line • 13th line • 18th line • 19th line • 23rd line • 48th line • 50th line • 57th line • 60th line	• First line • 13th line • 18th line • 19th line • 23rd line • 49th line • 51st line • 58th line • 61st line	*APPLICATION_ID*	Change the application identifier of the application to be used.

Change location		Change target	Change contents
HiRDB	Oracle		
• 74th line	• 74th line		
None	• First line • 13th line • 18th line • 19th line • 23rd line • 49th line • 51st line • 58th line • 60th line • 74th line	*SCHEMA_NAME*	Change the schema name of user connected to database.
Seventh line	None	*ATTRIBUTE_DATA_SIZE_MAX*	Change the maximum size (units: bytes) of HTTP session attribute information.
74th line	74th line	*HTTP_SESSION_NO*	Change the amount of global session information stored in the database.

3. Execute the created SQL file for table creation.
   For executing the SQL file, use SQL Executer when using HiRDB and SQL*Plus when using Oracle.

(5) Environment settings of database

If you want to use the database session failover functionality, set a timeout in database (for HiRDB, UAP processing time monitoring functionality).

If the database session failover functionality is enabled, table records of the database that are targeted for operation during functionality processing, are exclusively controlled. As a result, when problems such as a failure on a J2EE server host occur, records targeted for operation might remain in exclusion state. In this case, creation of new HTTP sessions or connection between J2EE servers and database might fail.

If you set a timeout, such situations are detected, the transaction is rolled back, and you return to the state before records are exclusively controlled when a timeout occurs. Therefore, there is no impact on the system.

For preventing malfunctioning, set a timeout value for the database that is greater than the timeout value set in DB Connector. For details on database settings and procedure, see the *HiRDB UAP Development Guide* when using HiRDB and Oracle manuals when using Oracle.

The following table describes the processing in which records are exclusively controlled, tables targeted for operation in the processing, impact on the system when a failure occurs on a J2EE server during a processing, and the messages are output.

Table 6–46: Processing in which records are exclusively controlled and impact on the system when a failure occurs on a J2EE server during a processing

Sr. No.	Processing that exclusively controls records	Table targeted for operation	Impact on the system when a failure occurs	Output message
1	Negotiation processing when starting a Web application	Application information table	Because application negotiation fails, the Web application that uses the database session failover functionality fails to start.	Not output

Sr. No.	Processing that exclusively controls records	Table targeted for operation	Impact on the system when a failure occurs	Output message
2	Processing of creating global session information	Blank record information table	The system can only create a total of 90% of the HTTP sessions. After this, HTTP session creation or deletion processing might fail.	• When integrity mode is disabled: KDJE34368-W • When integrity mode is enabled: KDJE34312-W
3	Processing of deleting global session information	Blank record information table	The system can only create a total of 90% of the HTTP sessions. After this, HTTP session creation or deletion processing might fail.	• When integrity mode is disabled: KDJE34377-E • When integrity mode is enabled: KDJE34312-W
4	Processing of updating global session information	Session information storage table	The number of HTTP sessions that you can create in the system reduces by one. After this, if you receive a request that operates the reduced HTTP session, HTTP session acquisition fails.	• When integrity mode is disabled: KDJE34368-W • When integrity mode is enabled: KDJE34312-W
5	Processing of monitoring validity of global session information	Application information table	Validity of global session information in the database is not monitored.	• When integrity mode is disabled: Do not execute exclusion processing • When integrity mode is enabled: KDJE34336-W

6.6.4 DB Connector settings

When using the database session failover functionality, create a new DB Connector apart from the one used in an application. You need to create one DB Connector for each J2EE server. Use the same DB Connector for all applications that use the database session failover functionality.

For details on the procedures from importing to starting a DB Connector, see *4.2 Settings for connecting to the database* in the *uCosminexus Application Server Application Setup Guide*.

This section describes the following settings required for DB Connector to be used in the database session failover functionality:

- Specifying the transaction support level
- Specifying an optional name of DB Connector
- Environment settings of DB Connector

(1) Setting transaction support level

With the database session failover functionality, you must set transaction support levels. Specify `NoTransaction` in the `<transaction-support>` tag under the `<hitachi-connector-property>-<resourceadapter>-<outbound-resourceadapter>` tag in the Connector property file.

(2) Specifying optional name of DB Connector

With the database session failover functionality, you must specify an optional name for DB Connector. By default `COSMINEXUS_SFO_DBCONNECTOR` is specified as the optional name for DB Connector.

If you want to change the name to be specified from the default name, specify any name in the `<optional-name>` tag under the `<hitachi-connector-property>`-`<resourceadapter>`-`<outbound-resourceadapter>`-`<connection-definition>`-`<connector-runtime>`-`<resource-external-property>` tag in Connector property file. For details on specifying optional name of DB Connector, see *2.6 Assigning optional name to Enterprise Bean or J2EE server (user-specified name space functionality)* in the *uCosminexus Application Server Common Container Functionality Guide*.

You must also change the optional name of DB Connector to be defined in a J2EE server to the same value. For details on specifying the optional name of DB Connector for J2EE servers, see *6.6.1 J2EE server settings*.

(3) Environment settings of DB Connector

The database session failover functionality is used for implementing 24-hour continuous operation. In order to ensure that there is no impact on the system even if a failure occurs in a database, you must perform settings such as connection failure detection for implementing continuous operation. Consider the time required for recovery and set the value.

For details on connection failure detection, see *3.15.1 Detecting a connection failure* in the *uCosminexus Application Server Common Container Functionality Guide.*

For details on properties to be set in DB Connector and setting methods, see *4.2.2 Defining the DB Connector properties* in the *uCosminexus Application Server Application Setup Guide.*

This subsection describes the properties required to be set in a DB Connector.

(a) Properties to be set in <config-property> tag

This subsection describes the properties set in the `<config-property>` tag for each database used. Settings of the database session failover functionality are not required for the properties that are not described in this section.

> **Important note**
>
> If you want to use the database session failover functionality, you must set statement pooling. Statement pooling greatly affects memory usage of J2EE servers. Therefore, consider the connection pooling settings also and determine the value to be specified in the `PreparedStatementPoolSize` property.
>
> For details on statement pooling, see *3.14.4 Statement pooling* in the *uCosminexus Application Server Common Container Functionality Guide*. For details on connection pooling, see *3.14.1 Connection pooling* in the *uCosminexus Application Server Common Container Functionality Guide*. For details on the values that you can specify in *PreparedStatementPoolSize*, see *4.1.10 Properties that you can specify in the <config-property> tag set up for DB Connector* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*. For details on memory size to be used per statement, see JDBC related documents.

•Properties to be set when using HiRDB

The following table describes properties to be set when using HiRDB.

Table 6–47: Properties to be set when using HiRDB

Value to be specified in the <config-property-name> tag	Value to be specified in the <config-property-type> tag	Contents or value to be specified in the <config-property-value> tag	Mandatory/ Optional
`description`	`java.lang.String`	Specify connection addition information required for connecting to a database.	Mandatory
`DBHostName`	`java.lang.String`	Specify host name of HiRDB to be connected.	Mandatory
`loginTimeout`	`java.lang.Integer`	Specify maximum waiting time (seconds) for establishing physical connection with HiRDB server when acquiring Connection objects by using the `getConnection` method.	Optional
`LONGVARBINARY_Access`	`java.lang.String`	Specify *LOCATOR*.	Mandatory
`PreparedStatementPoolSize`	`java.lang.Integer`	Specify the numeric value determined by 30 × *number-of-Web-applications -on-a-J2EE-server-that-use -the-database-session-failover -functionality*.	Mandatory
`CancelStatement`	`java.lang.Boolean`	Specify `true`.	Mandatory
`logLevel`	`java.lang.String`	Specify any level for the levels of log trace output by DB Connector.	Optional

Legend:

Mandatory: Must be specified

Optional: Specify if required

Important note

To use the database session failover functionality, specify the following values in the client environment definition:

Environment variable name	Value
`PDISLLVL`	`2`
`PDFORUPDATEEXLOCK`	`NO`
`PDDBLOG`	`ALL`

- **Properties to be set when using Oracle**

The following table describes properties to be set when using Oracle.

Table 6–48: Properties to be set when using Oracle

Value to be specified in the <config-property-name> tag	Value to be specified in the <config-property-type> tag	Contents or value to be specified in the <config-property-value> tag	Mandatory/ Optional
`databaseName`	`java.lang.String`	Specify a specific database name (SID) on Oracle server.	Mandatory#
`serverName`	`java.lang.String`	Specify host name or IP address of Oracle server.	Mandatory#

Value to be specified in the <config-property-name> tag	Value to be specified in the <config-property-type> tag	Contents or value to be specified in the <config-property-value> tag	Mandatory/ Optional
`portNumber`	`java.lang.Integer`	Specify a port number that listens to requests from Oracle server.	Mandatory#
`url`	`java.lang.String`	Specify JDBC URL required by Oracle JDBC Thin Driver for connecting to a database.	Mandatory#
`loginTimeout`	`java.lang.Integer`	Specify a timeout (units seconds) for connection trial to database.	Optional
`PreparedStatementPoolSize`	`java.lang.Integer`	Specify the numeric value determined by 30 × <number of Web applications on a J2EE server that uses the database session failover functionality>.	Mandatory
`logLevel`	`java.lang.String`	Specify any level for the levels of log trace output by DB Connector.	Optional

Legend:

Mandatory: Must be specified

Optional: Specify if required

#

Specify all values of `databaseName`, `serverName`, and `portNumber`, or specify the value of url.

(b) Properties to be specified in <property> tag

The following table describes the properties to be set in the `<property>` tag. Settings of the database session failover functionality are not required for the properties that are not described in this section.

Table 6–49: Properties to be specified in the <property> tag

Value to be specified in the <property-name> tag	Value to be specified in the <property-type> tag	Value of the <property-default-value> tag	Contents or value to be specified in the <property-value> tag	Mandatory /Optional
`MaxPoolSize`	`int`	10	Specify maximum value of connection pool#1.	Mandatory
`MinPoolSize`	`int`	10	Specify minimum value of connection pool#1.	Mandatory
`LogEnabled`	`boolean`	true	Specify *true*.	Mandatory
`User`#2	`String`	--	Specify a user name.	Mandatory
`Password`	`String`	--	Specify a password.	Mandatory
`ValidationType`	`int`	1	Specify *1*.	Mandatory
`RetryCount`	`int`	0	Specify connection acquisition retry count. Specify a value, in accordance with the database settings and network environment that enables recovery of a database when a failure occurs.	Optional
`RetryInterval`	`int`	10	Specify connection acquisition retry interval.	Optional

Value to be specified in the <property-name> tag	Value to be specified in the <property-type> tag	Value of the <property-default-value> tag	Contents or value to be specified in the <property-value> tag	Mandatory /Optional
			Specify a value, in accordance with the database settings and network environment that enables recovery of a database when a failure occurs.	
`RequestQueueEnable`	`boolean`	true	Specify *true*.	Mandatory#3
`RequestQueueTimeout`	`int`	30	Specify maximum value that can stop the queue waiting for connection acquisition when connections exhaust.#4	Mandatory#3
`WatchEnabled`	`boolean`	true	Specify whether you want to enable connection pool monitoring.	Optional
`WatchInterval`	`int`	30	Specify connection pool monitoring interval.	Optional
`WatchThreshold`	`int`	80	Specify threshold value that monitors connection pool use status.	Optional
`WatchWriteFileEnabled`	`boolean`	true	Specify *true*.	Optional

Legend:

Mandatory: Must be specified

Optional: Specify if required

--: None

#1

Calculate the value of the connection pool by using the formulas given below. Set the same value as the maximum and minimum value of the connection pool.

If you set number of concurrent execution threads in Web application unit or URL unit:

Sum of number of concurrent execution threads in a Web application that uses the database session failover functionality on a J2EE server + 2

If you set number of concurrent execution threads in J2EE server unit,

Number of concurrent execution threads in J2EE server + 2

If a J2EE server receives requests exceeding the maximum value of connection pools, those requests go to the state of waiting in a queue for acquiring connection, when connections exhaust.

#2

Register the creator of the table in the user set in DB Connector.

#3

These settings are not required if the functionality that controls the number of concurrent execution threads in the Web application unit, is disabled.

#4

Specify a value in the following range.

If Web server integration is used:

1 < `RequestQueueTimeout` < Timeout value for data reception from the web container that is set in the reverse proxy

In the preceding expression, *Timeout value for data reception from the web container that is set in the reverse proxy* is the value specified for the `timeout` key in the `ProxyPass` directive or the value specified in the `Timeout` directive.

If requests are directly exchanged with the J2EE server and not via a web server:

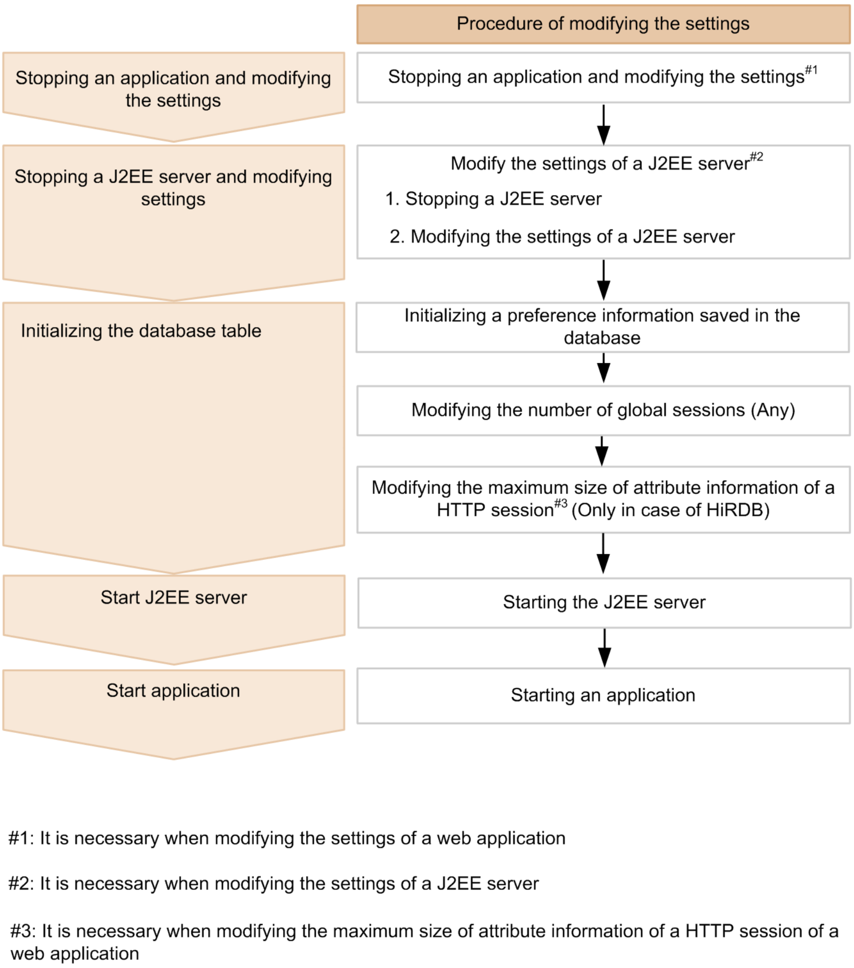
1 < `RequestQueueTimeout`

6.7 Changing settings related to the database session failover functionality

This section describes the changes in settings related to the database session failover functionality. With the database session failover functionality, store preference information such as application information and global session information in database tables. When changing the settings of a once started Web application to check that there is no error in settings by using negotiation processing executed when starting a Web application, you must initialize the preference information of a Web application stored in a database. For details on negotiation processing, see *6.4.1 Processing when starting an application*.

The following figure shows the flow of changing settings related to the database session failover functionality.

Figure 6–13: Flow of changing settings related to the database session failover functionality



Points to be considered when changing settings related to the database session failover functionality are:

- **Notes related to scope of stopping when changing settings**
  If you want to change the following settings, stop all other replicated J2EE servers or applications.
  - Setting integrity mode
  - Setting the amount of global session information in the database
- **Notes related to changing the maximum size of HTTP session attribute information**
  If you change the maximum size of HTTP session attribute information to a smaller value, when you inherit the global session information created before the change, the maximum size after the change might be exceed. If the

maximum size exceeds, the `KDJE34320-E` message is output while serializing attribute information and global session information is not stored in the database. Therefore, if you change the maximum size of HTTP session attribute information to a smaller value, destroy the HTTP session. For details on destroying an HTTP session, see *6.7.3 Deleting global session information (destroying HTTP sessions)*.

This section describes the changes in settings of a J2EE server and an application, and initialization of a database.

Reference note

For starting and stopping an application, use server management commands or the management portal. For details on starting an application, see *cjstartapp (start J2EE application)* in the *uCosminexus Application Server Command Reference Guide*. For details on stopping an application, see *cjstopapp (stop J2EE application)* in the *uCosminexus Application Server Command Reference Guide*. For details on operating the management portal, see *12.3 Managing a J2EE application* in the *uCosminexus Application Server Management Portal User Guide*.

6.7.1 Changing settings of a J2EE server and application

This subsection describes the procedure for changing settings of a J2EE server and a Web application. If you change the settings, you must initialize the information stored in the database. For details on initialization of the information stored in a database, see *6.7.2 Initializing a database table*.

(1) Stopping an application and changing settings

For changing settings of an application, stop a J2EE application and change the settings of a Web application.

After completing changes in the settings of Web applications on one J2EE server, change the settings of Web applications on other replicated J2EE servers. By changing the settings of the same Web applications one by one for replicated J2EE servers, you can change the settings of the entire system without stopping the entire system.

For details on setting items, see *6.5 Definitions in cosminexus.xml* and for details on changing the settings of a Web application, see *6.6.2 Web application settings*.

(2) Stopping a J2EE server and changing settings

For changing the J2EE server settings, execute the following procedures:

1. Stop J2EE applications.
   Stop all J2EE applications on the J2EE server.

2. Stop the J2EE server.
   Stop the J2EE server.

3. Change the settings of J2EE server in Easy Setup definition file.
   Change the settings in the Easy Setup definition file. For details on setting items on the J2EE server, see *6.6.1 J2EE server settings*.

4. Change settings of other replicated J2EE servers.
   Serially execute steps 1 to 3 for other replicated J2EE servers and specify the same changes to the settings of all replicated J2EE servers.

6.7.2 Initializing a database table

If you change the information used in a Web application or the information related to a Web application, you must initialize the preference information of the Web application stored in the database. This subsection describes the procedures for initializing preference information stored in the database. This subsection also describes the procedures for changing the amount of global session information in a database and changing the maximum size of HTTP session attribute information in a database.

(1) Initializing preference information stored in a database

If you change the settings of a once started Web application, you must initialize the preference information stored in the database.

This subsection describes the procedure for initializing preference information stored in a database.

1. Copy the template file to any location.
   A template file is provided as an SQL file used for initializing the preference information stored in a database. The storage locations of template files for each database used are:
   - Storage location of template files when using HiRDB
     In Windows: *Application-Server-installation-directory*`\CC\sfo\sql\hirdb_reset_apptbl.sql`
     In UNIX: `/opt/Cosminexus/CC/sfo/sql/hirdb_reset_apptbl.sql`
   - Storage location of template files when using Oracle
     In Windows: *Application-Server-installation-directory*`\CC\sfo\sql\oracle_reset_apptbl.sql`
     In UNIX: `/opt/Cosminexus/CC/sfo/sql/oracle_reset_apptbl.sql`
2. Edit the template file.
   Edit the template file in accordance with the preference information of the Web application and create an SQL file for initializing the preference information stored in the Web application. The following table describes change locations and change contents in template files.

   Table 6–50: Change locations and change contents in template files

Change location	Change target	Change contents
First line	*APPLICATION_ID*	Change application identifier of the application to be used.

3. Execute the SQL file for initializing the preference information stored in the created database.
   For executing the SQL file, use SQL Executer when using HiRDB and SQL*Plus when using Oracle.

(2) Changing the amount of global session information in a database

Change the amount of global session information in the database in accordance with the upper limit of the number of HTTPsession objects. However, if negotiation processing executed when starting an application fails, the amount of global session information in the database and the setting of upper limit of the number of HTTPsession objects can be different if you have set to continue the processing of starting a Web application.

This subsection describes the procedure for changing the amount of global session information in a database. If you change the amount of global session information in a database, all session information in the database is deleted.

1. Stop the J2EE applications and J2EE servers.
   Stop all applications in the J2EE server and all replicated J2EE servers.
2. Copy the template file to any location.

A template file is provided as an SQL file for changing the amount of global session information in a database. The storage locations of template files for each used database are:

- Storage location of template files when using HiRDB
  In Windows: *Application-Server-installation-directory*`\CC\sfo\sql\hirdb_change_session_num.sql`
  In UNIX: `/opt/Cosminexus/CC/sfo/sql/hirdb_change_session_num.sql`
- Storage location of template files when using Oracle
  In Windows: *Application-Server-installation-directory*`\CC\sfo\sql\oracle_change_session_num.sql`
  In UNIX: `/opt/Cosminexus/CC/sfo/sql/oracle_change_session_num.sql`

3. Edit the template file.
   Edit the template file in accordance with the preference information of the Web application and create an SQL file for changing the amount of global session information in the database. The following table describes change locations and change contents in template files.

Table 6–51: Change locations and change contents in template files

Change location		Change target	Change contents
HiRDB	Oracle		
• First line • Second line • Third line • Sixth line • Seventh line	• First line • Second line • Third line • Sixth line	*APPLICATION_ID*	Change the application identifier of the application to be used.
• Fourth line • Seventh line	• Fourth line • Sixth line	*HTTP_SESSION_NO*	Change the amount of global session information stored in the database.

4. Execute the SQL file created for changing the amount of global session information in the database.
   For executing the SQL file, use SQL Executer when using HiRDB and SQL*Plus when using Oracle.

(3) Changing the maximum size of HTTP session attribute information in a database (only in the case of HiRDB)

If you change the maximum size of HTTP session attribute information that you can include in global session information that is set in a Web application after creating a session information storage table, you must change the maximum size of the HTTP session attribute information in the database. Set a value for the maximum size of HTTP session attribute information in the database that is greater than the maximum size of HTTP session attribute information that you can include in the global session information set in a Web application. For details on setting the maximum size of HTTP session attribute information that you can include in global session information set in a Web application, see *6.5 Definitions in cosminexus.xml*.

This subsection describes the procedures for changing the maximum size of HTTP session attribute information in a database. These procedures are required only when using HiRDB.

1. Stop the J2EE applications and J2EE servers.
   Stop all applications in the J2EE server and all replicated J2EE servers.

2. Copy the template file to any location.

A template file is provided as an SQL file for changing maximum size of HTTP session attribute information in a database. The storage locations of template file are:

- In Windows: *Application-Server-installation-directory*`\CC\sfo\sql\hirdb_change_attributes_size.sql`
- In UNIX: `/opt/Cosminexus/CC/sfo/sql/hirdb_change_attributes_size.sql`

3. Edit the template file.

   Edit the template file in accordance with preference information of a Web application and create an SQL file used for changing the maximum size of HTTP session attribute information in a database. The following table describes the change locations and change contents in template files.

Table 6–52: Change locations and change contents in template files

Change location	Change target	Change contents
First line	*APPLICATION_ID*	Change the application identifier of the application to be used.
Second line	*ATTRIBUTE_DATA_SIZE_MAX*	Change size (units: bytes) of the column that stores HTTP session attribute information.

4. Execute the SQL file created for changing the amount of global session information in a database.

   For executing the SQL file, use SQL Executer.

6.7.3 Deleting global session information (destroying HTTP sessions)

You might have to destroy HTTP sessions existing in a system when upgrading the version of an application during system operation.

With the database session failover functionality, because global session information is stored in a database, you cannot destroy an HTTP session with stopping the J2EE application or J2EE sever. Destroy an HTTP session by deleting global session information from the database.

Execute the following procedures for deleting global session information:

1. Stop the J2EE applications.

   Stop all J2EE applications on the J2EE server.

2. Delete global session information in the database.

   Delete global session information in the procedures for changing global session information in a database. At this time, do not change the amount of global session information and execute only the change procedure. For details on the change procedure, see *6.7.2(2) Changing the amount of global session information in a database*.

3. Start the J2EE application.

6.8 Deleting database tables

When changing the settings of an application that uses the database session failover functionality, deleting database tables might be required. This section describes deleting database tables.

The following table describes tables to be deleted and reference locations for the procedure of deleting.

Table 6–53: Tables to be deleted and reference location of the procedure of deleting

Table name	Physical name in a database	Reference location for the procedure of deleting
Application information table	`SFO_`*APPLICATION_ID*`_APP_INFO`	*6.8.1*
Session information storage table	`SFO_`*APPLICATION_ID*`_SESSIONS`	*6.8.2*
Blank record information table	`SFO_`*APPLICATION_ID*`_REC_INFO`	

Template files for deleting database tables that are used in the database session failover functionality, are stored in the following location:

In Windows:

Application-Server-installation-directory`\CC\sfo\sql\`

In UNIX:

`/opt/Cosminexus/CC/sfo/sql/`

There are two types of template files for table deletion for each database used. The following table describes mapping of databases to be used, files, and types of tables to be deleted.

Table 6–54: Template files for table deletion and tables to be deleted

Database to be used	Template file	Types of table to be deleted		
		Application information table	Session information storage table	Blank record information table
HiRDB	`hirdb_delete_apptbl.sql`	Y	--	--
	`hirdb_delete_sessiontbl.sql`	--	Y	Y
Oracle	`oracle_delete_apptbl.sql`	Y	--	--
	`oracle_delete_sessiontbl.sql`	--	Y	Y

Legend:
Y: Can be deleted
--: Cannot be deleted

The following subsection describes details on template files for each database used.

6.8.1 Deleting application information tables

Application information tables store settings related to the database session failover functionality set in a Web application.

The procedures for deleting the application information table are as follows:

1. Copy the template file to any location.
   A template file is provided as an SQL file used for deleting a table. The storage locations of the template files for each used database are:
   - Storage location of template files when using HiRDB
     In Windows: *Application-Server-installation-directory*`\CC\sfo\sql\hirdb_delete_apptbl.sql`
     In UNIX: `/opt/Cosminexus/CC/sfo/sql/hirdb_delete_apptbl.sql`
   - Storage location of template files when using Oracle
     In Windows: *Application-Server-installation-directory*`\CC\sfo\sql\oracle_delete_apptbl.sql`
     In UNIX: `/opt/Cosminexus/CC/sfo/sql/oracle_delete_apptbl.sql`
2. Edit the template file.
   Edit the template file in accordance with the preference information for a Web application and create an SQL file used for deleting a table. The following table describes change locations and change contents in a template file.

Table 6–55: Change locations and change contents in a template file

Change location		Change target	Change contents
HiRDB	Oracle		
First line	First line	*APPLICATION_ID*	Change the application identifier of the application to be used.
None	First line	*SCHEMA_NAME*	Change the schema name of user connected to database.

3. Execute the SQL file created for table deletion.
   For executing the SQL file, use SQL Executer when using HiRDB and SQL*Plus when using Oracle.

6.8.2 Deleting session information storage table and blank record information table

The session information storage table stores global session information. The blank record information table manages the unused records in the session information storage table.

This subsection describes the procedures for deleting the session information storage table and blank record information table.

1. Copy the template file to any location.
   A template file is provided as an SQL file used for deleting a table. The storage locations of template file for each used database are:
   - Storage location of template files when using HiRDB
     In Windows: *Application-Server-installation-directory*`\CC\sfo\sql\hirdb_delete_sessiontbl.sql`
     In UNIX: `/opt/Cosminexus/CC/sfo/sql/hirdb_delete_sessiontbl.sql`
   - Storage location of template files when using Oracle
     In Windows: *Application-Server-installation-directory*`\CC\sfo\sql\oracle_delete_sessiontbl.sql`

In UNIX: `/opt/Cosminexus/CC/sfo/sql/oracle_delete_sessiontbl.sql`

2. Edit the template file.

   Edit the template file in accordance with preference information of a Web application and create an SQL file used for deleting a table. The following table describes change locations and change contents in template files.

Table 6–56: Change locations and change contents in template files

Change location		Change target	Change contents
HiRDB	Oracle		
• First line • Second line • Third line • Fourth line	• First line • Second line • Third line • Fourth line	*APPLICATION_ID*	Change the application identifier of the application to be used.
None	• First line • Second line • Third line • Fourth line	*SCHEMA_NAME*	Change the schema name of user connected to the database.

3. Execute the SQL file created for table deletion.

   For executing the SQL file, use SQL Executer when using HiRDB and SQL*Plus when using Oracle.

6.9 Precautions to be taken when using database session failover functionality

This section describes the points to be considered when using the database session failover functionality.

- If you operate the contents of tables used in the database session failover functionality, you cannot correctly keep the system information and hence you cannot continue the normal operation.

  You can execute only the following operations as the operations that accompany changes made in the tables in a database, which are to be used in the database session failover functionality. Even when executing the following operations, follow the procedures described in reference locations.

  - Changing settings of the database session failover functionality (see *6.7*)
  - Initializing a table (see *6.7.2*)
  - Destroying an HTTP session (see *6.7.3*)

  Do not execute any other operations.

- If the size of HTTP session attribute information exceeds maximum size, the information of the HTTP session is not replicated in the database.

7 Suppression of Full GC by Using the Explicit Memory Management Functionality

With Application Server, you can use a memory space other than the Java heap as the Java object placement destination when executing a Java application. The function is called the Explicit Memory Management functionality. You can suppress occurrence of Full GC by using this functionality efficiently.

This chapter describes how you can use the Explicit Memory Management functionality to suppress occurrence of Full GC.

Note that this functionality cannot be used if G1 GC is used. If the `-XX:+HitachiUseExplicitMemory` option is specified when G1 GC is used, the following message is sent to the standard output and the Java VM terminates:

```
Using -XX:+UseG1GC and -XX:+HitachiUseExplicitMemory at the same time is
not supported.
```

Also note that on the application server, the Explicit Memory Management functionality is enabled by default. Therefore, if you use G1 GC, specify the `-XX:-HitachiUseExplicitMemory` option.

7.1 Organization of this chapter

The Explicit Memory Management functionality suppresses occurrence of Full GC by moving Java objects to the Explicit heap, an area outside the Java heap.

The following table describes the organization of this chapter.

Table 7–1: Organization of this chapter (Suppression of Full GC by using the Explicit Memory Management functionality)

Category	Title	Reference location
Explanation	Overview of the Explicit Memory Management functionality	*7.2*
	Overview of the memory space used in the Explicit Memory Management functionality	*7.3*
	Objects placed in the Explicit heap when using a J2EE server	*7.4*
	Objects that you can optionally place in the Explicit heap in the application	*7.5*
	The life cycle of the Explicit memory block and executed processes	*7.6*
	Releasing the Explicit memory block when the automatic release functionality is enabled	*7.7*
	Releasing the Explicit memory block when the automatic release functionality is disabled	*7.8*
	Releasing the Explicit memory block by using the `javagc` command.	*7.9*
	Reducing the time required for automatic release processing of the Explicit memory blocks	*7.10*
	Reducing the memory usage of the Explicit heap that is used in an HTTP session	*7.11*
Implementation	Implementing a Java program that uses the Explicit Memory Management functionality API	*7.12*
Settings	Settings in the execution environment	*7.13*
Notes	Points to be considered when using the Explicit Memory Management functionality	*7.14*

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

For details about the mechanism of GC, see *7. JavaVM Memory Tuning* in the *uCosminexus Application Server System Design Guide*.

7.2 Overview of the Explicit Memory Management functionality

This section describes the overview of the Explicit Memory Management functionality.

7.2.1 Objectives of using the Explicit Memory Management functionality

The Explicit Memory Management functionality is a function that suppresses occurrence of Full GC. By using this function, you can reduce the frequency of system halts and achieve a stable throughput.

The size of a Java heap handle on Application Server is increased by increasing the logical address space handled in the system or by expanding the system scale. It is a problem that the time required for GC increases as the Java heap size increases. While GC is being performed, system operation stops. Especially, the time required for Full GC increases according to the size of the used Java heap area. This also means that increasing the size of the available Java heap can increase the time required for Full GC accordingly.

Reference note

Relationship between the GC algorithm and system downtime

The Java VM supports Copy as the algorithm for copy GC and Mark Sweep Compact as the algorithm for Full GC. Both of these algorithms are of Stop-The-World type. With the algorithm of this type, GC requires the same time as the downtime of the system that uses the Java VM.

7.2.2 Mechanism of suppressing Full GC by using the Explicit Memory Management functionality

The Explicit Memory Management functionality uses an independent area called the *Explicit heap* as the Java object placement destination. The Explicit heap is an area outside the Java heap that is not subject to GC. Java objects in the Java heap can trigger GC. If the Explicit Memory Management functionality is used, these Java objects are moved to the Explicit heap. As a result, Full GC for those Java objects is suppressed.

This section describes the mechanism whereby Full GC is suppressed by using the Explicit Memory Management functionality. This section also describes the role of the functionality.

(1) Mechanism of suppressing Full GC

If the Eden area runs out of free space while a Java application is running, GC occurs. At this time, Java VM performs Full GC if the following expression is satisfied:

```
Size of memory that is used in New area > Size of free space in Tenured area
```

Note

Because the Eden area is out of free space, the size of the memory used in the New area is almost the same as the maximum size of the `New` area.

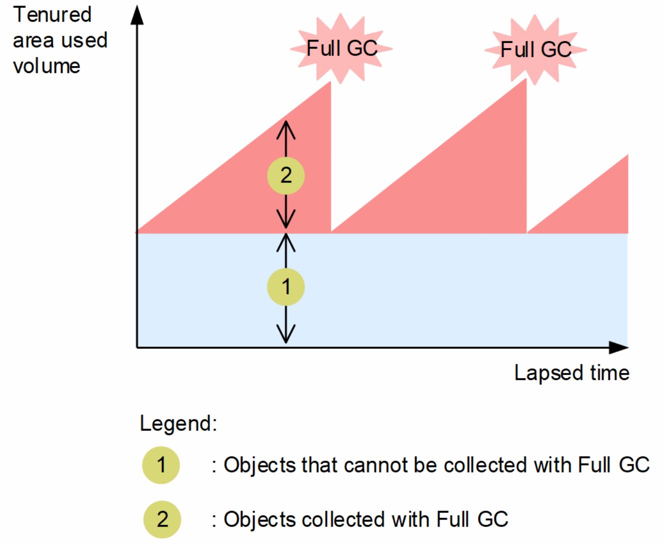
As the preceding expression shows, Full GC occurs when the free space in the Tenured area becomes low. The free space in the Tenured area is consumed by Java objects that are moved (promoted) from the Survivor area when copy GC occurs. Therefore, occurrence of Full GC can be suppressed by decreasing the number of Java objects that are promoted. Note that objects that survive copy GC several times without being deleted and get promoted to the Tenured area are called *long-life objects*.

The long-life objects are broadly categorized into two types. Long-life objects of one type are those that are not reclaimed by Full GC. For example, the objects that should essentially be stored in the Tenured area, and which continue to exist constantly during the application execution. Because these objects do not increase continuously, they are not an essential cause of Full GC occurrence. If you want to eliminate the impact of such long-life objects, increase the size of the Tenured area.

Long-life objects of the other type are those that are reclaimed by Full GC. These objects have life that is long enough to the extent that they are promoted to the Tenured area but become unnecessary in a certain period of time. Because such long-life objects continue to increase until Full GC occurs, they can be a cause of Full GC occurrence.

The following figure shows objects that are reclaimed and are not reclaimed by Full GC.

Figure 7–1: Objects that are reclaimed and are not reclaimed by Full GC

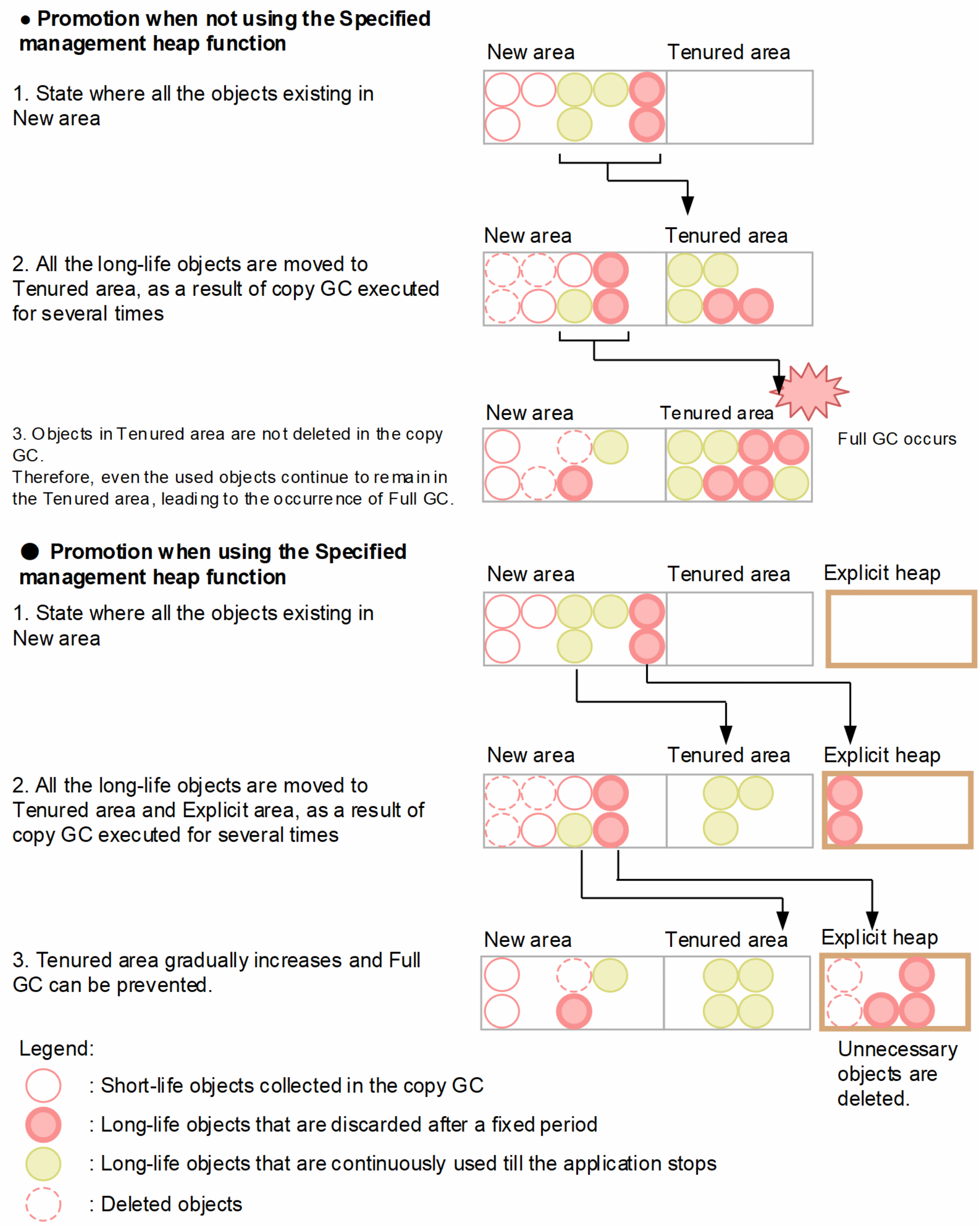


You cannot prevent the increase of objects, which will become unnecessary over a period of time only by increasing the size of the Tenured area. For example, if you double the size of the Tenured area, the Full GC occurrence interval will only be doubled. The effect gained will be less than expected.

Therefore, to suppress occurrence of Full GC, it is important to reduce the number of objects that are promoted to the Tenured area but become unnecessary in a certain period of time.

On the application server, settings are specified so that some Java objects are promoted to the Explicit heap when copy GC occurs. The following figure shows the difference in promotion when you are not using the Explicit Memory Management functionality and when you are using the Explicit Memory Management functionality.

Figure 7–2: Difference in promotion when you are not using Explicit Memory Management functionality and when you are using Explicit Memory Management functionality



In both the cases, status is same at step 1. In step 2, when the objects are promoted, all long-life objects are moved to the Tenured area if you are not using the Explicit Memory Management functionality. On the other hand, if you are using the Explicit Memory Management functionality, the objects from among the long-life objects that will definitely be destroyed after a certain period of time, are moved to the Explicit heap. Thus, only the long-life objects, which are not planned to be destroyed, are moved to the Tenured area and the used size of the Tenured area increases slowly. As shown in step 3, if you are using the Explicit Memory Management functionality, objects in the Explicit heap are deleted when they become unnecessary.

For details on the target Java objects, see *7.4 When using J2EE server objects placed in Explicit heap*. For details about GC algorithms, see *7. JavaVM Memory Tuning* in the *uCosminexus Application Server System Design Guide*.

If you use the Explicit Memory Management functionality in an application developed that you have developed, generate long-life objects, which will be destroyed over a certain period of time, directly in the Explicit heap. That will prevent an increase in the memory size of the Tenured area. For details on the Java objects that can be generated in the Explicit heap, see *7.5 Objects that you can optionally place in the Explicit heap in the application*.

(2) Positioning of the Explicit Memory Management functionality

The *Explicit Memory Management functionality* is a functionality of JavaVM. You can use the Explicit Memory Management functionality in the following two ways:

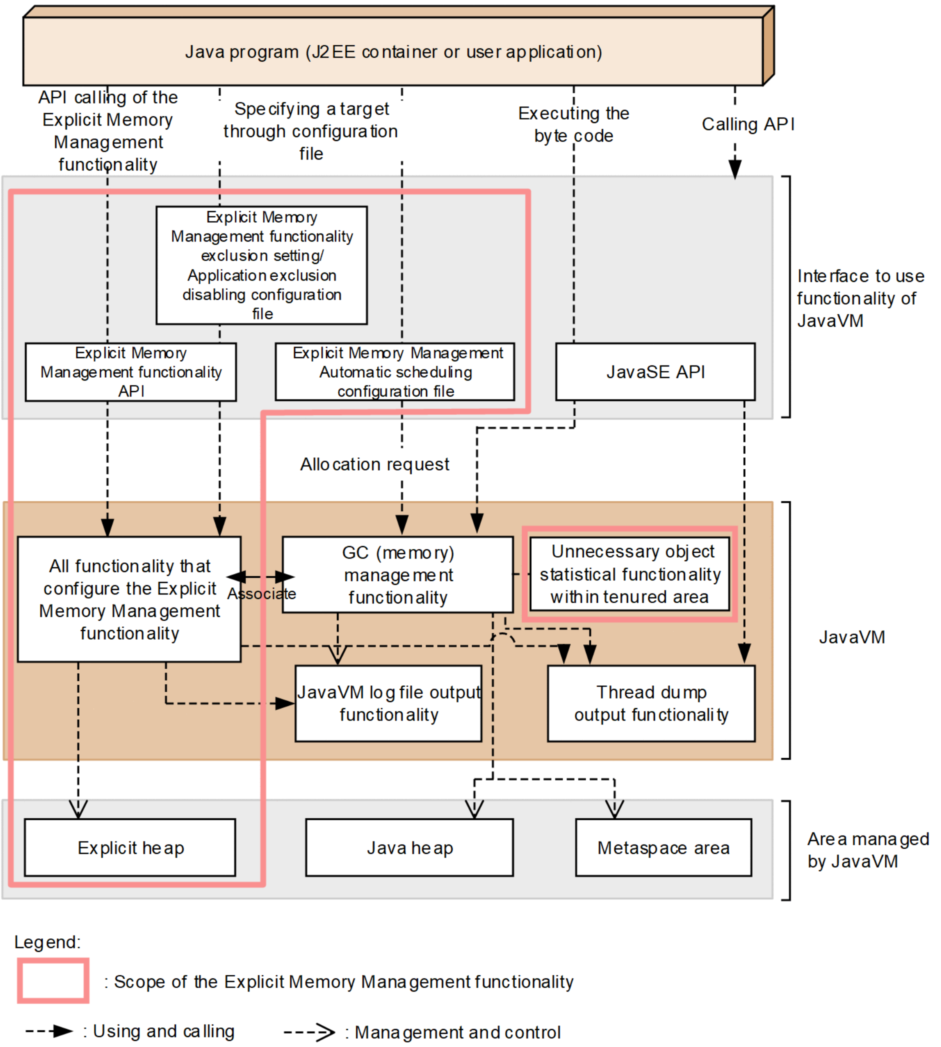
- By using the configuration file of the Explicit Memory Management functionality.

  The following are the configuration files of the Explicit Memory Management functionality. By using these files, you can set target objects that use the Explicit Memory Management functionality.

  - A configuration file for the Explicit Memory Management functionality application exclusion or disabling the application exclusion
  - Auto allocated configuration file
- By using the Explicit Memory Management functionality API

The following figure shows the positioning of the Explicit Memory Management functionality. Note that the JavaVM log file output functionality in the figure refers to JavaVM log file output functionality.

Figure 7–3: Positioning of the Explicit Memory Management functionality



This subsection describes the Explicit Memory Management functionality API, automatic placement configuration file, configuration file for the Explicit Memory Management functionality application exclusion or disabling application exclusion, functionality for configuring the Explicit Memory Management functionality, functionality of statistics of unnecessary objects in the Tenured area, and the Explicit heap.

Explicit Memory Management functionality API

If you want to use the Explicit Memory Management functionality from a Java program, use the Explicit Memory Management functionality API. With this API, you can execute the operations related to the Explicit heap. You can also collect the use status of the Explicit heap as statistics.

Automatic placement configuration file

Use the automatic placement configuration file to use the Explicit Memory Management functionality without making changes to the Java program. Specify the objects that you want to place in the explicit management heap, in the file.

Configuration file for Explicit Memory Management functionality application exclusion or disabling application exclusion

The objects referenced by objects that have been automatically moved (by using the automatic placement functionality) to the Explicit management heap are also automatically moved to the Explicit management heap when GC occurs based on a reference relation. If you want to exclude the objects to be moved on the basis of this reference relation, from an application of the Explicit Memory Management functionality in the unit of class, use the configuration file for the Explicit Memory Management functionality application exclusion and configuration file for disabling application exclusion of the Explicit Memory Management functionality.

When you want to exclude an object from an application target of the Explicit Memory Management functionality, use the configuration file for the Explicit Memory Management functionality application exclusion. Specify classes of the objects, which are not to be moved to the Explicit management heap, in this file.

In cases such as when all classes in the same package are excluded from an application target of the Explicit Memory Management functionality in the configuration file for the Explicit Memory Management functionality application exclusion, if you want to target some classes for application of the Explicit Memory Management functionality, use the configuration file for disabling application exclusion of the Explicit Memory Management functionality. Specify the classes, for which the setting of application exclusion of the Explicit Memory Management functionality is to be disabled, in this file.

Functionality for configuring the Explicit Memory Management functionality

Any functionality that configures the Explicit Memory Management functionality is included in JavaVM. Such functionality is called by API. You can execute the following processes:

- Management and control of the Explicit heap and memory blocks in the heap
- Moving of objects to the Explicit heap by changing the allocation processing linked with GC
  The allocation process is executed by the extension of a new keyword.
- Control on movement of objects to Explicit heap memory blocks
- Output of an Explicit heap event log and the status to JavaVM log file and the thread dump

Functionality of statistics of unnecessary objects in the Tenured area

This functionality checks the unnecessary objects that are the cause of memory increase in the Tenured area. For details on the functionality of statistics of unnecessary objects in Tenured area, see *9.8 Unused objects statistical functionality in the Tenured area* in the *uCosminexus Application Server Maintenance and Migration Guide*.

Explicit heap

Java objects not subject to GC are moved to this area. This area is managed by the Explicit Memory Management functionality. The Explicit heap is configured from multiple memory blocks (Explicit memory blocks).

(3) Required memory size when using the Explicit Memory Management functionality

The Explicit heap managed by the Explicit Memory Management functionality is an area outside the Java heap. When using an Explicit heap, the memory usage increases as compared to memory usage when not using the Explicit heap.

When using the Explicit Memory Management functionality, you need to estimate and appropriately set the maximum size of the Explicit heap as the required memory size. For details on the flow of using the Explicit Memory Management functionality, objects stored in the Explicit heap (objects that are the cause of memory size increase in the Tenured area), and the estimation of the Explicit heap size, see *7.11 Explicit heap tuning* in the *uCosminexus Application Server System Design Guide*.

7.2.3 Prerequisites for using the Explicit Memory Management functionality

This subsection describes the prerequisites for using the Explicit Memory Management functionality. The availability of using the Explicit Memory Management functionality varies from server to server and command to command.

The following table describes whether the Explicit Memory Management functionality is supported. For the default settings, see *7.13.1 Common settings for using the Explicit Memory Management functionality (setting JavaVM options)*.

Table 7–2: Support for the Explicit Memory Management functionality

Server or command type	Supported
J2EE server	Y
Batch server	Y
cjclstartap command	Y

Legend:

Y: The functionality is supported.

7.3 Overview of memory space used in the Explicit Memory Management functionality

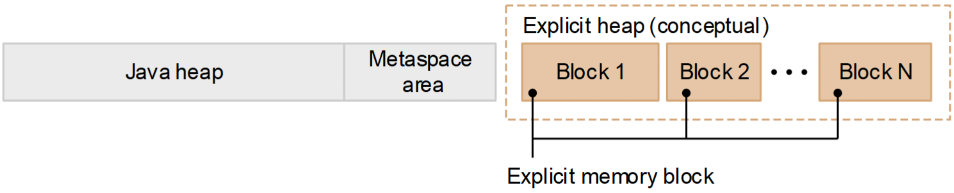
This subsection describes the structure of an Explicit heap, which is the memory space used by the Explicit Memory Management functionality. For the configuration of memory space used by the Java VM, see also *7.2.6 Configuration of memory space used by JavaVM when using serial GC and JavaVM options* in the *uCosminexus Application Server System Design Guide*.

The Explicit heap is a memory space in which no GC occurs. It is configured from multiple memory blocks. Memory blocks that configure an Explicit heap are called *Explicit memory blocks*. The Explicit heap is a concept that represents all Explicit memory blocks.

Execute operations such as initialization and release for each Explicit memory block unit.

The following figure shows the concept of an Explicit heap.

Figure 7–4: Concept of Explicit heap



Set the maximum size of an Explicit heap in the JavaVM startup option `-XX:HitachiExplicitHeapMaxSize`. For details on the `-XX:HitachiExplicitHeapMaxSize` option, see *-XX:HitachiExplicitHeapMaxSize (Option for specifying the maximum size of the Explicit memory block)* in the *uCosminexus Application Server Definition Reference Guide*. You can generate (initialize) maximum 1,048,575 Explicit memory blocks. You cannot generate Explicit memory blocks more than the maximum number.

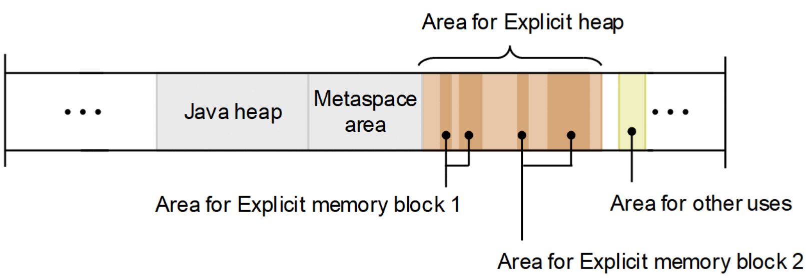
About the time when the Explicit heap area is secured:

The actual memory area of the maximum size of the Explicit heap, that you have specified in the `-XX:HitachiExplicitHeapMaxSize` option, is secured when JavaVM is started. The area is secured as a continued area from the Java heap and Metaspace area.

If the memory required to place Java objects in the Explicit memory blocks is insufficient, the memory area for Explicit memory blocks is secured from the area of the Explicit heap that was secured when starting JavaVM. For this reason, the memory area in the Explicit memory blocks is divided into multiple areas.

The following figure shows the image of using the virtual memory space.

Figure 7–5: Image of using the virtual memory space



Although the area for an Explicit heap is a continued area, the area used in a single Explicit memory block is non-continuous.

7.4 When using J2EE server objects placed in Explicit heap

This section describes the objects placed in an Explicit heap when using the J2EE server.

On the J2EE server, the following objects are placed in the Explicit heap to suppress occurrence of Full GC:

- Objects related to an HTTP session

The Web container secures the Explicit memory block area, and releases and reserves Explicit memory blocks. This section describes the processes executed by the Web container for the object.

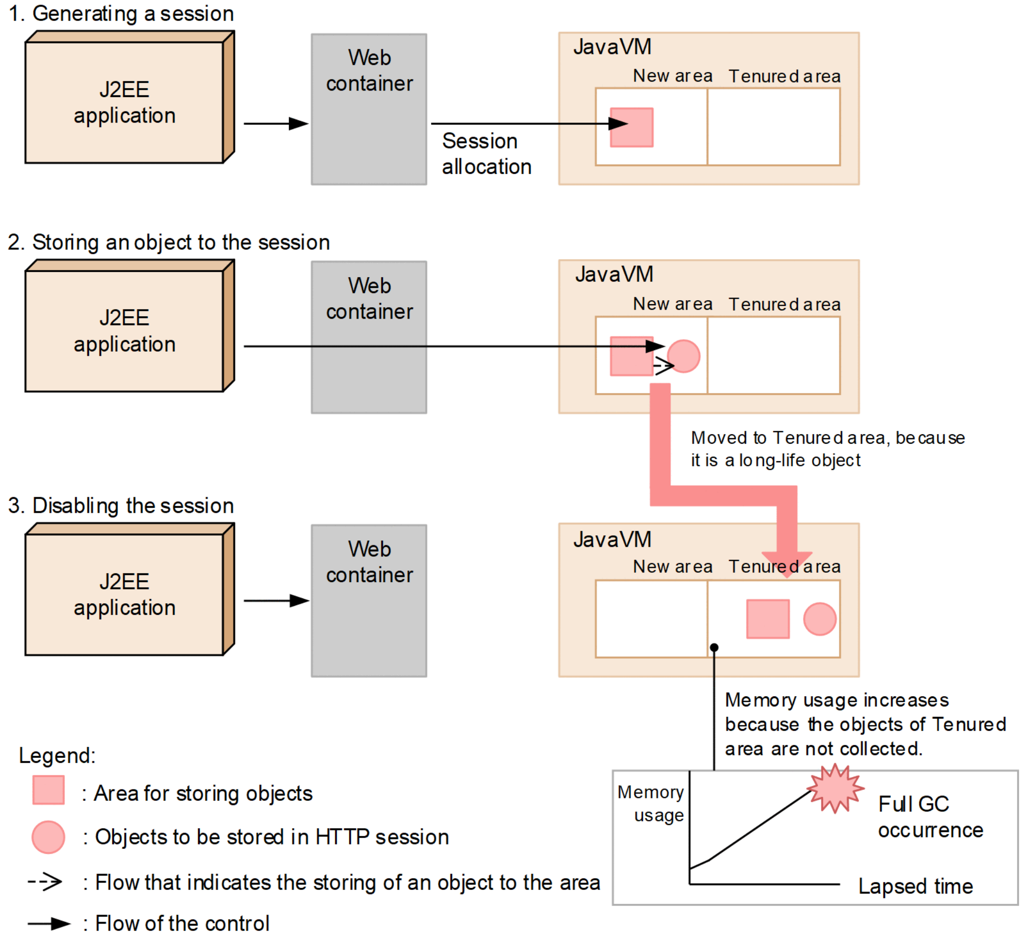
7.4.1 Objects related to HTTP session

The objects stored in an HTTP session are the objects that are retained when the session is active. The objects exist from the generation of the session until the session is destroyed.

These objects are likely to be used continuously while copy GC is performed several times if the Explicit Memory Management functionality is not used. Therefore, these objects are easy to get promoted to the Tenured area as long-life objects. Because objects that have been promoted to the Tenured area are not reclaimed by copy GC, they remain even after the session is destroyed. For this reason, the amount of memory used for the Tenured area continues to increase, thus causing Full GC to occur.

The following figure shows an example of not using the Explicit Memory Management functionality

Figure 7–6: An example of not using the Explicit Memory Management functionality

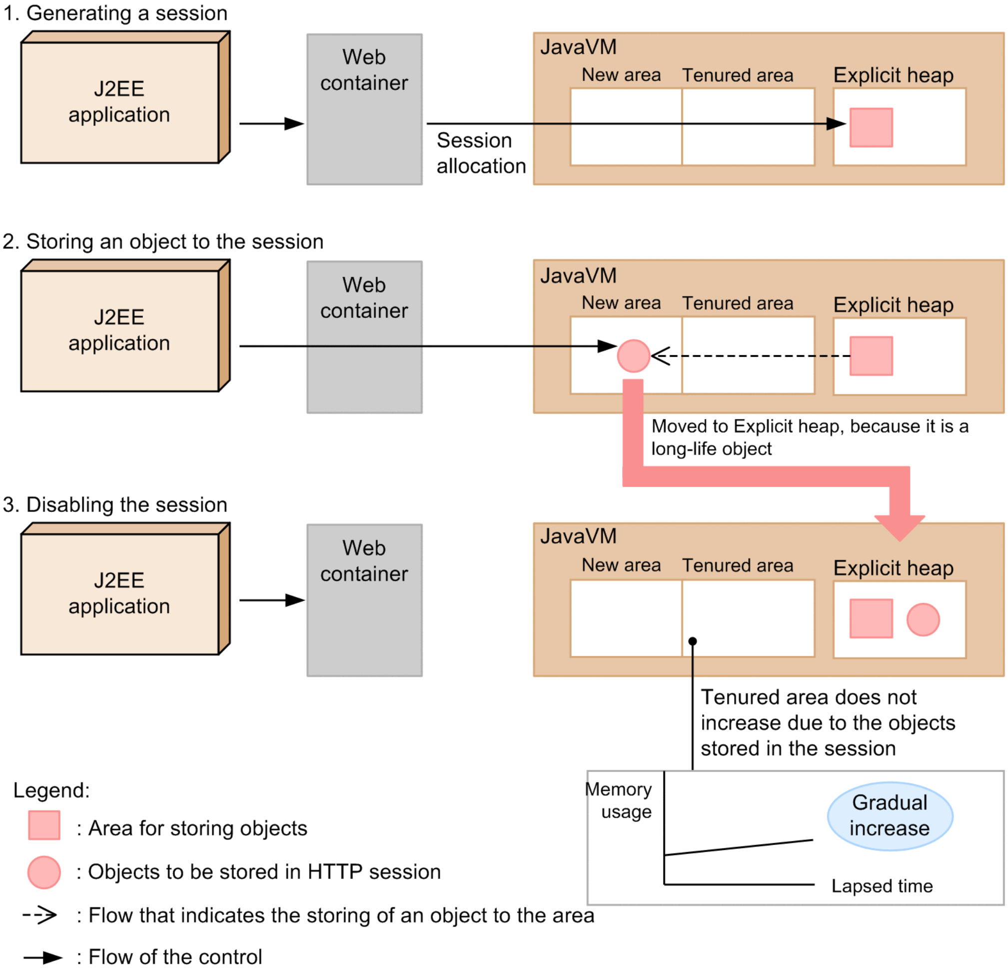


When you generate a session in step 1, the area for storing objects is secured in the New area. In step 2, objects are stored in a session. After copy GC occurs several times, the objects and their storage area are moved to the Tenured area. Although the session deactivates in step 3, objects in the Tenured area are not reclaimed and the memory usage goes on increasing.

In this situation, you can suppress occurrence of Full GC by changing where objects related to an HTTP session are moved from the Tenured area to the Explicit heap.

The following figure shows an example of using the Explicit Memory Management functionality

Figure 7–7: An example of using the Explicit Memory Management functionality



If you use the Explicit Memory Management functionality, the Tenured area does not increase due to the objects related to an HTTP session. You can therefore suppress occurrence of Full GC. The J2EE server explicitly releases the Java objects placed in the Explicit heap after the session is destroyed.

This subsection describes the timing of securing and releasing the Explicit memory block area in which the HTTP session is to be placed.

When you create an HTTP session

When you create a new HTTP session, an Explicit memory block is created in the Explicit heap area. One Explicit memory block is allocated to one session. The HTTP session is secured inside the Explicit memory block.

However, objects are placed in the Java heap, immediately after a session is created. When Java objects that trigger GC are promoted after copy GC occurs several times, the objects are moved to the Explicit heap.

When an object is stored in the HTTP session (when setAttribute method is executed)

The `javax.servlet.http.HttpSession.setAttribute` method places the objects stored in the HTTP session, in Explicit memory blocks allocated to each session.

However, when the `setAttribute` method is executed, the objects are placed in the Java heap. When Java objects that trigger GC are promoted after copy GC occurs several times, the objects are moved to the Explicit heap. At that time, all the objects that were being referenced from the object to be moved are moved to the Explicit heap. However, the objects might not move to an Explicit heap depending on the specification of the option of controlling object movement to the Explicit memory block.

When the object is deleted from the HTTP session (when removeAttribute method is executed)

The Explicit heap is protected from GC. Therefore, even if you delete an object from the HTTP session by executing the `javax.servlet.http.HttpSession.removeAttribute` method, the area that was using the object is not released.

This also applies to attributes that are changed by using the `setAttribute` method, because GC is not performed for attributes that were set before they are changed.

The memory is released in Explicit memory blocks. Note that if your Web application repeatedly and frequently executes the `setAttribute` method, the area inside the Explicit memory block might get unnecessarily consumed even if you execute the `removeAttribute` method.

When the HTTP session is destroyed

When the HTTP session is destroyed, the Web container reserves the release of Explicit memory blocks created at the time of the HTTP session creation.

The Explicit memory blocks for which a reservation for release is made are actually released when copy GC or Full GC is performed later. At that time, all the areas reserved for release are released.

If references to objects stored in the session are retained after the Explicit memory blocks were released, see the following description.

- *7.7 Releasing Explicit memory blocks when the automatic release functionality is enabled*
- *7.8 Releasing Explicit memory blocks when the automatic release functionality is disabled*

The following table describes the mapping of an operation or an action executed by a Web application and a JavaVM action.

Table 7–3: Mapping of operations executed by Web application (API) and JavaVM action

Operations (API) or actions executed by Web application	Web container action	JavaVM action
• `javax.servlet.http.HttpServletRequest.getSession()` • `javax.servlet.http.HttpServletRequest.getSession(boolean)`	Generating a session	Securing Explicit memory blocks
• `javax.servlet.http.HttpSession.setAttribute(String, Object)`	Storing objects in the session	Placing objects in the Explicit memory block
• `Session timeout` • `javax.servlet.http.HttpSession.invalidate()`	Destroying the session	Releasing Explicit memory blocks

Apart from the HTTP session, the Web application + 2# Explicit memory blocks are used inside the Web container for objects for the HTTP session management.

#: Because two objects for management are kept internally in the Web container, add that count as well.

Note that you can reduce the memory size of the Explicit heap that is used in an HTTP session, by using the memory saving functionality of the Explicit heap. For details, see *7.11 Reducing memory usage of the Explicit heap that is used in an HTTP session*.

If you implement an application by referring to *Appendix A Efficient Usage of the Explicit Heap Used in an HTTP Session* in the *uCosminexus Application Server System Design Guide*, you can efficiently apply the Explicit Memory Management functionality to a HTTP session.

7.5 Objects that you can optionally place in the Explicit heap in the application

This section describes the objects that you can optionally place in the Explicit heap.

If you want to place Java objects other than the objects that are set in the J2EE server, in the Explicit heap, specify the objects that you want to place by using the automatic placement configuration file. For details on the automatic placement configuration file, see *7.13.2 Using the Explicit Memory Management functionality by using the automatic placement configuration file*.

For details on the automatic release functionality of the Explicit Management Heap, see *7.7 Releasing Explicit memory blocks when the automatic release functionality is enabled*.

If you are using the Explicit Memory Management functionality API, see *7.12 Implementing the Java program that uses the Explicit Memory Management functionality API*.

> **Tip**
>
> Even after you have tuned the Java and Explicit heap sizes, if the memory used for the Tenured area increases and Java objects that can trigger Full GC exist, consider placing the objects in the Explicit heap.

7.5.1 Conditions for objects that you can place in the Explicit heap

This subsection describes the Prerequisites for objects that you can place in an Explicit heap and the objects that are effective when placed.

(1) Prerequisites for objects that you can place

The objects that you want to place in the Explicit heap (Explicit memory block) must satisfy the following prerequisites:

- **The object must be a long-life object, which is the cause of increase in the Tenured area memory size**

  A certain amount of overhead is required for placing and releasing objects for Explicit memory blocks. Therefore, reduce the placing and releasing of objects in Explicit memory blocks as much as possible.

  If the Explicit Memory Management functionality is not used, placing short-life objects subject to reclaim by copy GC in Explicit memory blocks does not contribute to Full GC suppression and, what is more, increases overhead. Make sure that the objects placed in Explicit memory blocks are long-life and not subject to reclaim by copy GC.

  For details on how to identify long-life objects that cause an increase in the Tenured area memory size, see *9.8 Unused objects statistical functionality in the Tenured area* in the *uCosminexus Application Server Maintenance and Migration Guide*.

- **Survival period should be known (only when you use the Explicit Memory Management functionality API)**

  If the Explicit management heap is used via the Explicit Memory Management functionality API, the Explicit heap blocks are not subject to GC. Therefore, used objects are not automatically reclaimed.

  The objects placed in Explicit memory blocks need to be explicitly released by an application. However, if the survival period of the objects is not known, the objects cannot be explicitly released. Therefore, make sure to place only those objects, the survival period of which is known.

(2) Objects that are effective when placed

The Explicit Memory Management functionality prevents long-life objects that will be destroyed and reclaimed by Full GC after a certain period of time from being promoted to the Tenured area. Therefore, this functionality does not need to be applied to objects that are not reclaimed even by Full GC, such as objects that are used until the application stops.

The objects that are effective when placed in the Explicit heap are as follows:

- Objects that are generated and destroyed in a fixed life cycle.
- Objects that are not properly reclaimed after they are used and destroyed because the life cycle is longer than the time period after which the objects are promoted by copy GC

Placing the preceding objects in the Explicit heap can prevent unnecessary objects from remaining in the Tenured area and suppress occurrence of Full GC.

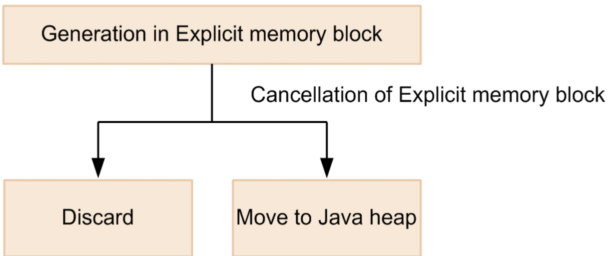
7.5.2 Life cycle and state transition of objects

This subsection describes the life cycle and state transition of objects to be placed in Explicit memory blocks.

You need to explicitly generate and release the objects placed in Explicit memory blocks by using the Explicit Memory Management functionality API, on the basis of survival period. The survival period and the life duration of objects vary according to the application processing.

The following figure shows the life cycle of objects to be placed in Explicit memory blocks.

Figure 7–8: Life cycle of objects to be placed in Explicit memory blocks



The object is generated directly in Explicit memory block. Then, if the Explicit memory block is released by using the Explicit Memory Management functionality API, the object is destroyed or moved to the Java heap depending on the state. For details on the operations when release processing is executed, see *7.8.2 The process of releasing the Explicit memory block when the automatic release functionality is disabled*.

7.6 Life cycle of Explicit memory block and executed processes

This section describes the life cycle of Explicit memory blocks and processes that are executed at each stage.

7.6.1 Life cycle and states of Explicit memory blocks

This subsection describes the life cycle and states of Explicit memory blocks.

The Explicit Memory Management functionality includes the following two methods of releasing the Explicit memory block:

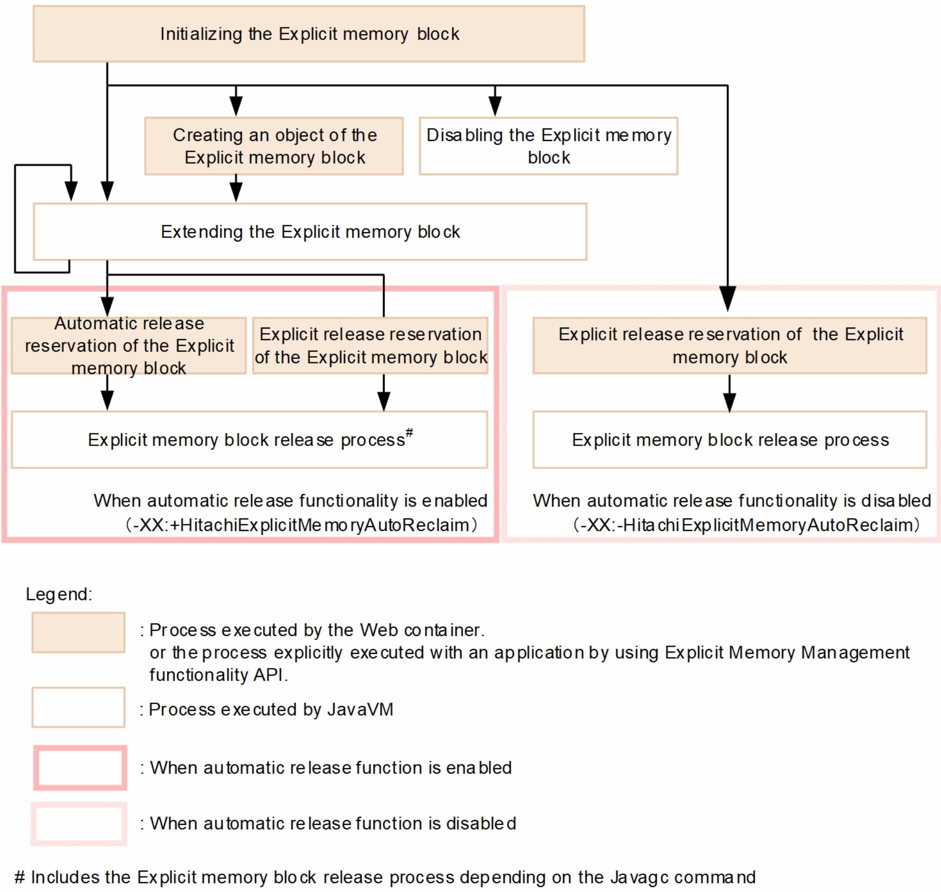
- Automatic releasing of Explicit memory blocks (Release process executed when the automatic release functionality is enabled)
- Explicit releasing of Explicit memory blocks (Release process executed when the automatic release functionality is disabled)

The specification method and processing varies according to the releasing process of Explicit memory blocks. The sections 8.7 onwards describe the releasing processes in detail.

(1) Life cycle of Explicit memory block

The following figure shows the life cycle of an Explicit memory block.

Figure 7–9: Life cycle of Explicit memory block



Each stage in the life cycle, which is shown in the figure, is described below.

Initializing and disabling the Explicit memory block

The Explicit memory block is initialized and generated.

- Web container initializes the Explicit memory block, which stores the objects related to an HTTP session.
- In an application, if you want to place any object in the Explicit heap, explicitly initialize the Explicit memory block by using the Explicit Memory Management functionality API.

The Explicit memory block might be disabled depending on the status at the time of initialization.

The processes executed during the initialization of the Explicit memory block and the conditions by which an Explicit memory block is disabled are described in *7.6.2 Initializing the Explicit memory block*.

Generating objects in the Explicit memory block

In an application, if you want to store any object in an Explicit memory block, generate and place the object in the Explicit memory block by using the Explicit Memory Management functionality API.

Details on generating objects in the Explicit memory block are described in *7.6.3 Directly generating objects in the Explicit memory block*.

Extending the Explicit memory block

If the area to place objects becomes insufficient during use, JavaVM expands the Explicit memory block.

Details on expanding the Explicit memory block are described in *7.6.4 Extending the Explicit memory block*.

Reserving release and releasing the Explicit memory block

The behavior of the processes of reserving release and releasing the Explicit memory block varies according to whether the automatic release functionality in the Explicit Memory Management functionality is enabled (`-XX:+HitachiExplicitMemoryAutoReclaim`) or disabled (`-XX:-HitachiExplicitMemoryAutoReclaim`).

If the automatic release functionality is enabled (-XX:+HitachiExplicitMemoryAutoReclaim)

- **Explicit release reserving of the Explicit memory block**
  The release is reserved for Explicit memory blocks that are specified by the Explicit Memory Management functionality API. For details on the explicit release reserving of the Explicit memory block when the automatic release functionality is enabled, see *7.7.1 Explicit release reserving of the Explicit memory block when the automatic release functionality is enabled*.
- **Automatic release reserving of the Explicit memory block**
  A Release is reserved for Explicit memory blocks processed by automatic release functionality in the Explicit Management Heap. For details on the automatic release reserving of the Explicit memory block when the automatic release functionality is enabled, see *7.7.2 Automatic release reserving of the Explicit memory block when the automatic release functionality is enabled*.
- **Releasing of the Explicit memory block**
  The process releases the objects that are placed in the Explicit memory blocks reserved by the explicit release reserving or automatic release reserving. For details on the process of releasing the Explicit memory block when the automatic release functionality is enabled, see *7.7.3 The process of releasing the Explicit memory block when the automatic release functionality is enabled*.

If you use the `javagc` command, you can execute the release process at any time, for Explicit memory blocks that are not released. For details on the release process of Explicit memory blocks by using the `javagc` command, see *7.9 Releasing Explicit memory blocks by using the `javagc` command*.

If the automatic release functionality is disabled (-XX:-HitachiExplicitMemoryAutoReclaim)

- **Explicit release reserving of Explicit memory block**

The reserve release in Explicit memory blocks when the objects placed in the Explicit memory block become unnecessary.

1. Web container reserves release of the Explicit memory block, which has stored objects related to the HTTP session.
2. In an application, if you have stored any object in the Explicit memory block, explicitly reserve the release of the Explicit memory block by using the Explicit Memory Management functionality API.

For details on the explicit release reserving of the Explicit memory block when the automatic release functionality is disabled, see *7.8.1 Explicit release reserving of the Explicit memory block when the automatic release functionality is disabled*. Note that when the explicit release reserving is executed, the Explicit memory block is not yet destroyed.

- **Releasing of Explicit memory block**

  When copy GC or Full GC occurs, the Java VM releases the Explicit memory blocks for which a reservation for release is made. It also destroys the objects placed in Explicit memory blocks at the same time. However, some objects are not destroyed and move to the Java heap.

  For details on the process of releasing the Explicit memory block when the automatic release functionality is disabled, see *7.8.2 The process of releasing the Explicit memory block when the automatic release functionality is disabled*.

(2) States of Explicit memory block

An Explicit memory block transits through the states such as `enabled`, `released` and `reserved for release` at each stage in the life cycle.

Moreover, an active Explicit memory block maintains a sub-status as described in the following table.

Table 7–4: Sub-status of Explicit memory block

Sub-status	Status of Explicit memory block
Enable	This is the initial status. In this status, you can use all functionality of the Explicit memory block.
Disable	In this status, you cannot move Java objects to the corresponding Explicit memory block. The status might change to this state when you extend the Explicit memory block.

7.6.2 Initializing the Explicit memory block

This section describes about the initialization and the execution of Explicit memory blocks and the processes that are executed during initialization.

(1) Execution timing

In an application, if you want to place any object in the Explicit heap, the Explicit memory block is initialized by invoking the following Explicit Memory Management functionality API.

- `BasicExplicitMemory.BasicExplicitMemory()`
- `BasicExplicitMemory.BasicExplicitMemory(String name)`

Besides these APIs, the Explicit memory block is also initialized when the object, which you specified in the automatic placement configuration file, is generated. The Web container performs initialization and places the first object in the Explicit memory block, in which the J2EE server places objects. For details on the execution timing, see *7.4 When using J2EE server objects placed in Explicit heap*.

(2) Executed details

The Explicit memory block is initialized. However, the memory area for an Explicit memory block is not secured at this stage.

The initialization is not performed in the following cases:

- **When you try to initialize the Explicit memory block after the maximum limit is exceeded**
  This refers to the case when the number of existing Explicit memory blocks is 1,048,575.
- **When the Explicit Memory Management functionality is OFF**
  This refers to the case when the `-XX:-HitachiUseExplicitMemory` option is not specified.

In such cases, although the constructor is successfully executed, it is handled as an invalid Explicit memory block. All the processes related to the initialized Explicit memory block (`ExplicitMemory` instance) become invalid.

7.6.3 Directly generating objects in the Explicit memory block

This section describes how to directly generate objects in the Explicit memory block and the processes that are executed.

In an application, if you want to generate objects in the Explicit memory block, use the API described in *(1) Execution timing*. When you execute the API, an object of the class that you specified in the argument, is generated in the Explicit memory block. However, objects that are generated in the initialization process by the constructor of that object are generated in the Java heap.

(1) Execution timing

If you use the Explicit Memory Management functionality in your application, you can directly generate an object in the Explicit memory block by invoking one of the following Explicit Memory Management functionality APIs.

- `ExplicitMemory.newInstance(Class type)`
- `ExplicitMemory.newInstance(Class type, Object... args)`
- `ExplicitMemory.newInstance(java.lang.reflect.Constructor cons, Object... args)`
- `ExplicitMemory.newArray(Class type, int number)`
- `ExplicitMemory.newArray(Class type, int[] dimensions)`

Besides these API, an object is also directly generated in the Explicit memory block when the object that you specified in the automatic placement configuration file of Explicit Management Heap is generated and the Explicit memory block is initialized.

You need not be aware of the objects placed by the J2EE server.

(2) Executed details

This subsection describes the details executed for each API.

Table 7–5: Details executed for each API

API	Executed details
`ExplicitMemory.newInstance(Class type)`	This API instantiates the class that you specify in type argument and places that class in the Explicit memory block shown by receiver.
`ExplicitMemory.newInstance(Class type, Object... args)`	
`ExplicitMemory.newInstance(java.lang.reflect.Constructor cons, Object... args)`	This API instantiates the class shown by `java.lang.reflect.Constructor` and places it in the Explicit memory block shown by receiver.
`ExplicitMemory.newArray(Class type, int number)`	This API instantiates the array of classes that you specify in type argument, of the length that you specify in number argument and places it in the Explicit memory block shown by receiver.
`ExplicitMemory.newArray(Class type, int[] dimensions)`	This API instantiates the array of classes that you specify in type argument, of the number of dimensions that you specify in dimensions argument and places it in the Explicit memory block shown by receiver.

For details on APIs, see *10.3 ExplicitMemory class* in the *uCosminexus Application Server API Reference Guide*.

However, if you cannot secure the required area in the Explicit memory block at the placement destination, the generated objects are not placed in the Explicit memory block. The generated objects are placed in the Java heap.

For the reasons why the area cannot be secured and the executed processes, see the description of why the extension process cannot be executed in *7.6.4 Extending the Explicit memory block*.

7.6.4 Extending the Explicit memory block

This section describes the process of extending Explicit memory blocks. When you execute the expansion process, the free space in an Explicit memory block increases.

(1) Execution timing

JavaVM executes the extension at the following timings:

- When placing the first object in the Explicit memory block
- When the Explicit memory block does not have the free space required to place an object

When you try to place an object in the Explicit memory block from an application that uses the Explicit Memory Management functionality API, if the object size exceeds the free space in the placement target Explicit memory block, the expansion process is executed.

After initializing the Explicit memory block, when an object is placed in the Explicit memory block for the first time, the expansion process is invariably executed.

The Web container performs initialization and places the first object in the Explicit memory block, in which the J2EE server places objects. For details on the execution timing, see *7.4 When using J2EE server objects placed in Explicit heap*.

(2) Executed details

The JavaVM secures memory area from OS and the appropriate Explicit memory block is expanded. The memory securing API is used to secure the memory area.

However, the expansion process is not executed in the following cases:

- **If you try to extend beyond the maximum limit of the Explicit heap**

  This is the case when the value obtained by adding the size that you are trying to extend to the total size of all Explicit memory blocks is more than the value that you specified in the `-XX:HitachiExplicitHeapMaxSize` option.

  The sub-status of the corresponding Explicit memory block changes to `Disable` and placement of the object to the Explicit memory block is canceled.

  Objects cannot be placed thereafter in the Explicit memory block that is changed to `Disable` state.

  For details on the `-XX:HitachiExplicitHeapMaxSize` option, see the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

- **If you try to extend the Explicit memory block having Disable sub-status**

  If you try to extend the Explicit memory block, the sub-status of which is `Disable,` placement of the object in the Explicit memory block is canceled.

The following table describes the changes in sub-status when extension of the Explicit memory block fails, for each reason.

Table 7–6: Changes in sub-status when extension of the Explicit memory block fails

Cause of extension failure	Change in sub-status
Failed to secure memory area from OS	`Enable -> Disable`
You tried to extend beyond the maximum limit of the Explicit heap	`Enable -> Disable`
You tried to extend the Explicit memory block having `Disable` sub-status	No change

7.6.5 Moving the objects from the Java heap to the Explicit memory block based on a reference relation

When objects move from the Java heap to the Explicit memory block, the objects in the Java heap that are being referenced from objects in the Explicit memory block automatically move to the Explicit memory block. Therefore, you need not set the movement from the Java heap to the Explicit memory block for the objects that have relationship with the moving objects. However, if you specify the `-XX:+ExplicitMemoryUseExcludeClass` option, objects of the classes that are coded in the configuration file for Explicit Memory Management functionality application exclusion do not move to the Explicit memory blocks.

Note that the Explicit memory blocks, which are created by the automatic placement functionality, are targeted for movement of objects based on a reference relationship from the Java heap to the Explicit memory block. The Explicit memory blocks created by the Explicit Management Heap API are not targeted.

Reference note

Take caution if the following phenomena occur after many objects are moved to the Explicit heap when Full GC occurs. In such a case, consider taking measures to prevent objects subject to automatic movement based on reference relations from moving to the Explicit heap.

- The processing of automatic release of the Explicit memory blocks takes time
- Small amount of Tenured area is used

Use the following functionalities for not moving the objects to the Explicit memory block:

1. Functionality for controlling object movement to the Explicit memory block
2. Functionality for specifying classes to be excluded from an application of the Explicit Memory Management functionality

The first functionality prevents objects from moving to the Explicit heap when Full GC occurs. This functionality can reduce the time of automatic release of Explicit memory blocks. The second functionality prevents objects of the classes specified in a configuration file from moving to the Explicit heap when copy GC occurs. This functionality can reduce the number of objects that move to the Explicit heap, although it depends on the classes you specify. If you use the second functionality, the first functionality is also enabled. Use the second functionality when a large number of objects are to be moved to the Explicit heap and the processing of automatic release of Explicit memory blocks takes time even if you use the first functionality.

(1) Execution timing

This processing is performed when copy GC or Full GC occurs.

(2) Executed details

After copy GC or Full GC processing ends, the Java VM checks for any Explicit memory blocks for which no reservation for release is made. The Java VM examines the reference relationship from the objects that are the basis of investigation, and continues the investigation until there are no more references. The areas other than the Java heap are not targeted for the investigation of a reference relationship. The objects referenced from the Explicit memory block are targeted for movement.

- For copy GC

  In addition to the preceding action, the Java VM performs operation according to the following rules when copy GC occurs:

  - Move an object when the object in the Explicit memory block that is being referenced, promotes.
  - Do not target objects for investigation if they are referring to the Metaspacr area, Explicit heap and the Tenured area.
  - Even if an Explicit memory block is reserved for release, consider it as targeted for moving.

    This case corresponds to the ones when objects cannot be moved because the area of the Explicit memory block cannot be secured and there is no free space in the movement destination Java heap when the object moves to the Java heap. In this case, Full GC is performed and free space is secured in the Java heap. The objects move to the Java heap after Full GC ends.

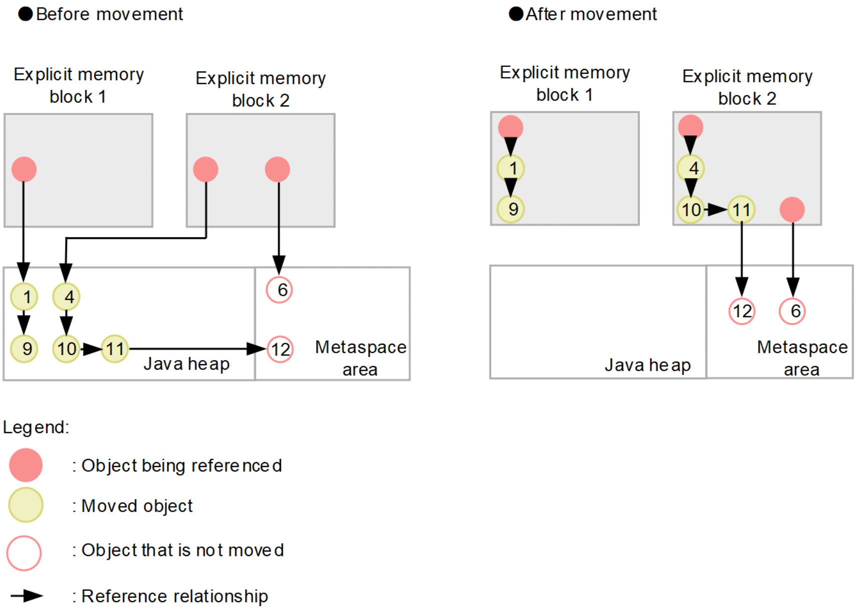
- For Full GC

  In addition to the preceding action, the Java VM performs operation according to the following rule when Full GC occurs:

  - If you specify `1` in the `-XX:ExplicitMemoryFullGCPolicy` option, the objects targeted for movement on the basis of a reference relation are not moved to Explicit memory blocks. The objects in the New area are moved to the Tenured area.

Figures 7-10 and 7-11 describe the flow of the movement of objects in accordance with these rules with examples. Specifying `0` in the `-XX:ExplicitMemoryFullGCPolicy` option is a prerequisite for the flow of the movement of objects described here.

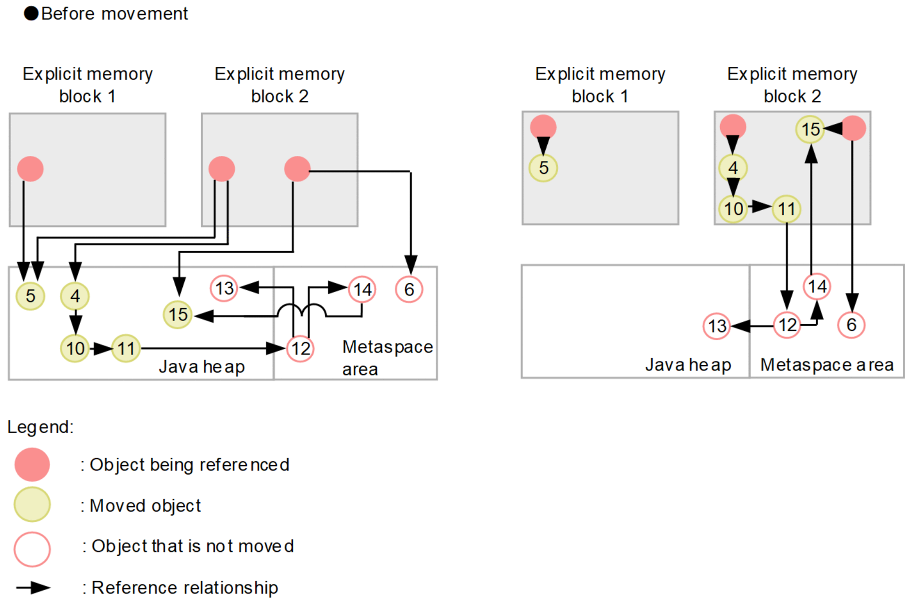
Figure 7–10: Objects that move on the basis of a reference relationship (Example 1)



The objects in the figure move in the following sequence:

1. Object 1 is being referenced from an object in the Explicit memory block 1. Therefore, the object 1 moves to the Explicit block 1.
2. Object 9 also moves to the Explicit memory block 1 because it is being referenced from the object 1.
3. In the same way as in points 1 and 2, the object 4, object 10, and object 11 move to the Explicit memory block 2.
4. Object 6 is being referenced from an object in the Explicit memory block 2. However, it is not an object in the Java heap and hence it does not move.
5. In the same way as in point 4, the object 12 also does not move.

Figure 7–11: Objects that move on the basis of a reference relationship (Example 2)



The objects in the figure move in the following sequence:

1. Object 13 is in the Java heap and can be accessed from an object in the Explicit memory block 2. However, it does not move because the investigation terminates at object 12.
2. Like object 13, the object 15 is referenced from the Metaspace area. However, in addition to the reference, the object can be accessed from an object in the Explicit memory block 2 without involving the Metaspace area or any other Explicit memory block. Therefore, it moves to the Explicit block 2.
3. Although the object 5 is being referenced from the Explicit memory block 1 as well as the Explicit memory block 2, it moves to the Explicit memory block 1.

   Note that the object 5 is being referenced from the Explicit memory block 1 as well as the Explicit memory block 2. In such cases, although it moves to either the Explicit memory block 1 or 2, it is not defined to which Explicit memory block it will move to.

In the case of following conditions, operation will be different than described in the example.

- **In case you cannot secure free space in the Explicit memory block**

  This corresponds to the case when there is no free space in placement destination the Explicit memory block for placing objects targeted for placement at the time of placing objects in the Explicit memory block. In such cases, you cannot place object in the Explicit memory block. Objects, which cannot be placed, are placed in Java heap area. If you are using API in an incorrect way, an API level exception might occur. For details, see *10.7 Exception class* in the *uCosminexus Application Server API Reference Guide*.

7.6.6 Event log output at each stage in the life cycle

An event log is output at each stage in the life cycle of the Explicit memory block. The event log is output when an event leading to output occurs.

The following table describes the mapping of each stage in the life cycle and the timing of an event log output. An event leading to log output varies according to the set log output level.

Table 7–7: Mapping of each stage in the life cycle and output event log

Stage in life cycle	Timing of event log output	Log output level
Initialization of the Explicit memory block	Initialization of the Explicit memory block	verbose
	Initialization of the Explicit memory block (detailed information output)	debug
	Failure in initializing the Explicit memory block	verbose
Extension of the Explicit memory block	Change of sub-status of the Explicit memory block to Disable	verbose
Explicit release reserving of the Explicit memory block	Release reserving of the Explicit memory block by finalizer	verbose
Automatic release automatic reserving of the Explicit memory block		verbose
Automatic release explicit reserving of the Explicit memory block		verbose
Explicit releasing process of the Explicit memory blocks	Explicit releasing process of the Explicit memory block	normal
	Explicit releasing process of the Explicit memory block (detailed information output)	verbose
	Java heap overflow during explicit release processing of the Explicit memory block	normal
	Movement of object to the Java heap due to explicit release of the Explicit memory block	debug
Automatic releasing process of the Explicit memory block	Automatic releasing process of the Explicit memory block	normal
	Java heap overflow during automatic releasing process of the Explicit memory block	normal
Direct generation of objects in the Explicit memory block	Generation of objects in the Explicit memory block	verbose

In addition to the preceding event log data, event log data is output independently of the stage in the life cycle when GC occurs.

For details on setting event log acquisition in the Explicit Memory Management functionality, see *3.3.17 Settings for Acquiring the JavaVM Material* in the *uCosminexus Application Server Maintenance and Migration Guide*. For details on the event log details of the Explicit Memory Management functionality, see *4.19 Event log of the Explicit Memory Management functionality* in the *uCosminexus Application Server Maintenance and Migration Guide*.

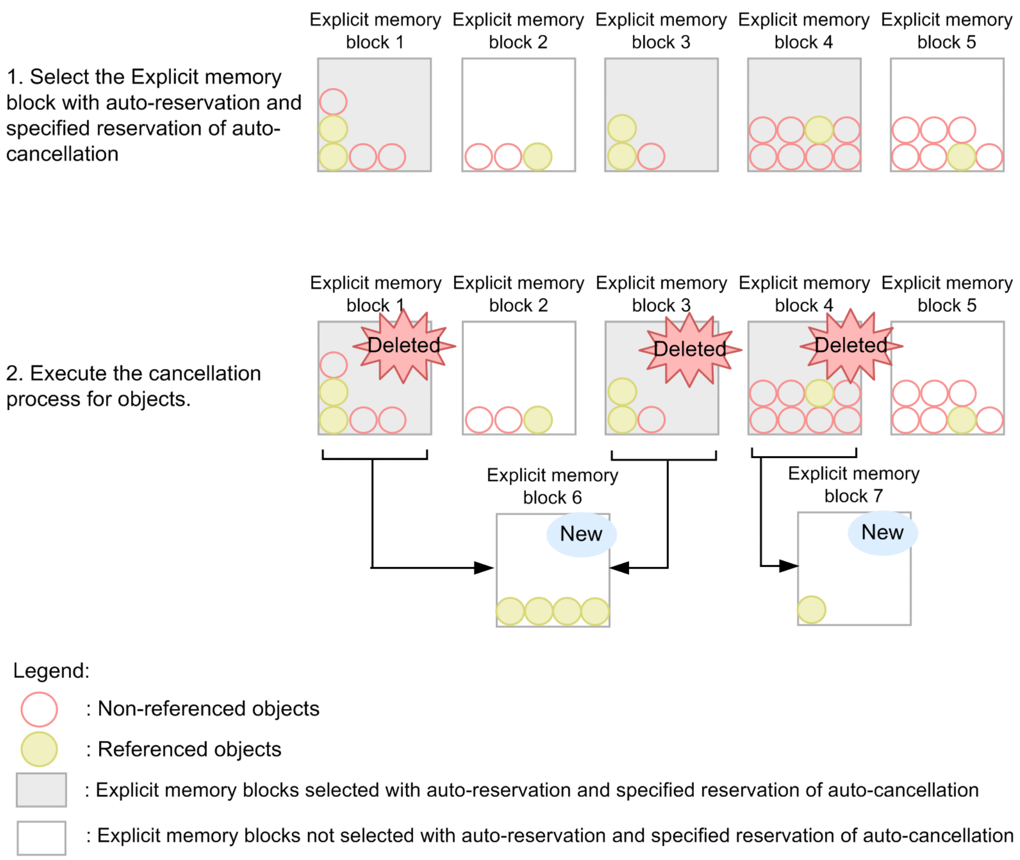
7.7 Releasing Explicit memory blocks when the automatic release functionality is enabled

This section describes the automatic release functionality of the Explicit management heap.

The functionality of automatically releasing the Explicit memory block is executed in two steps; reserving the release and the release process. You can perform effective processing by respectively reserving multiple Explicit memory blocks and executing the release process together.

There are two types of automatic release reserving; automatic release explicit reserving and automatic release automatic reserving. The following figure shows the automatic releasing of Explicit memory blocks.

Figure 7–12: Automatic releasing of the Explicit memory block



The following subsections describe the processing executed when the automatic release functionality is enabled.

7.7.1 Explicit release reserving of the Explicit memory block when the automatic release functionality is enabled

In explicit release reserving of the Explicit memory block when the automatic release functionality is enabled, explicitly reserve release for the Explicit memory block by using the API.

(1) Execution timing

You can reserve explicit release of the Explicit memory block when the automatic release functionality is enabled, by invoking one of the following Explicit Memory Management functionality APIs.

- `ExplicitMemory.reclaim(ExplicitMemory... areas)`
- `ExplicitMemory.reclaim(ExplicitMemory area)`
- `ExplicitMemory.reclaim(ExplicitMemory area0,ExplicitMemory area1)`
- `ExplicitMemory.reclaim(Iterable<ExplicitMemory> areas)`
- `BasicExplicitMemory.finalize()`

(2) Executed details

When you invoke an API described in (1), the Explicit memory block area that you specified in the argument of the API is reserved for release.

If you are using the API in an incorrect way, an API level exception might occur. For details, see *10.7 Exception class* in the *uCosminexus Application Server API Reference Guide*.

7.7.2 Automatic release reserving of the Explicit memory block when the automatic release functionality is enabled

In automatic release reserving of the Explicit memory block when the automatic release functionality is enabled, JavaVM automatically reserves release for the Explicit memory block. The Explicit memory blocks placed by the automatic placement functionality are targeted.

(1) Execution timing

This processing is performed by the Java VM when GC occurs.

(2) Executed details

When GC occurs, the Java VM automatically makes a reservation for releasing Explicit memory blocks.

7.7.3 The process of releasing the Explicit memory block when the automatic release functionality is enabled

The process of releasing the Explicit memory block when the automatic release functionality is enabled, is executed for the Explicit memory blocks that are reserved in advance by automatic release reserving and explicit release reserving. The release processing deletes the unnecessary Explicit memory blocks from the memory.

Note that if the objects that are being referenced from outside (Explicit memory blocks which are not targeted for releasing) exist, the objects are moved to a new Explicit memory block.

(1) Execution timing

The Java VM releases Explicit memory blocks according to the reservation made for automatic release when the same type of GC that caused the reservation to be made occurs.

(2) Executed details

The executed details are same as in the case of the processing of releasing the Explicit memory block when the automatic release functionality is disabled, except for the behavior of objects that are being referenced from the Explicit memory blocks, which are not targeted for releasing. For the details that are executed in the process of releasing Explicit memory blocks, see *7.8.2 The process of releasing the Explicit memory block when the automatic release functionality is disabled*.

In the case of the following conditions, the operation will be different.

- **In the case you cannot secure free space in the Explicit memory block**

  This corresponds to the case when there is no free space in the placement destination Explicit memory block for placing objects targeted for placement at the time of placing objects in the Explicit memory block. In such cases, you cannot place an object in the Explicit memory block. The objects, which cannot be placed, are placed in the Java heap area.

- **If the Java heap overflows when moving objects to the Java heap**

  This corresponds to the case when objects cannot be moved because the area of the Explicit memory block cannot be secured and there is no free space in the movement destination Java heap when the object moves to the Java heap. In this case, Full GC is performed and free space is secured in the Java heap. The objects move to the Java heap after Full GC ends.

  If the free space required for moving Java objects cannot be secured even when Full GC is performed, a log file is output and the objects are placed in the Explicit memory blocks again. For details on the log files that are output, see *4.19 Event log of the Explicit Memory Management functionality* in the *uCosminexus Application Server Maintenance and Migration Guide*.

- **If sufficient free space cannot be secured by Full GC**

  This corresponds to the case where the Java heap ran out of free space and the free space required for moving Java objects cannot be secured even by performing Full GC. In such cases, JavaVM aborts as it does in the case of an insufficient C heap. However, in the case of an insufficient C heap, the required memory size is output as *nnn* in the prompt `request` *nnn* `bytes`. When JavaVM aborts, 0 is always output as *nnn*.

7.8 Releasing Explicit memory blocks when the automatic release functionality is disabled

This section describes the explicit release functionality of Explicit Management Heap.

The functionality of explicitly releasing the Explicit memory block is executed by dividing into two steps of reserving for release and release process, same as the automatic release functionality of Explicit Management Heap. Reserve multiple Explicit memory blocks respectively and execute the release process together.

The following subsections describe the processing executed in the explicit release functionality.

7.8.1 Explicit release reserving of the Explicit memory block when the automatic release functionality is disabled

The functionality of releasing the Explicit memory block is executed by dividing into two steps of reserving for release and the actual release process. You can perform an effective processing by respectively reserving multiple Explicit memory blocks and executing the release process together.

(1) Execution timing

In an application, if you place any object in the Explicit Heap, you can reserve release of the Explicit memory block by invoking one of the following Explicit Memory Management functionality APIs.

- `ExplicitMemory.reclaim(ExplicitMemory... areas)`
- `ExplicitMemory.reclaim(ExplicitMemory area)`
- `ExplicitMemory.reclaim(ExplicitMemory area0,ExplicitMemory area1)`
- `ExplicitMemory.reclaim(Iterable<ExplicitMemory> areas)`
- `BasicExplicitMemory.finalize()`

Web container reserves the release of the objects placed by the J2EE server. For details on the execution timing, see *7.4 When using J2EE server objects placed in Explicit heap*.

(2) Executed details

When you invoke an API described in (1), the Explicit memory block area that you specified in the argument of the API is reserved for release.

If you are using the API in an incorrect way, an API level exception might occur. For details, see *10.7 Exception class* in the *uCosminexus Application Server API Reference Guide*.

7.8.2 The process of releasing the Explicit memory block when the automatic release functionality is disabled

The process of releasing is executed for Explicit memory blocks that are already reserved for releasing. The release processing actually deletes unnecessary Explicit memory blocks from the memory.

(1) Execution timing

JavaVM executes the release process at following timings:

- When copy GC occurs
- When Full GC occurs

(2) Executed details

After copy GC or Full GC processing ends, the Java VM checks for any Explicit memory blocks for which no reservation for release is made. If one or more corresponding Explicit memory blocks exist, those Explicit memory blocks are released. The Explicit memory blocks are released by the memory releasing API of the OS. At that time, the objects inside the released Explicit memory blocks are destroyed.

However, the objects, which are implementing the finalize method and which are not being referenced from anywhere, from among the objects in Explicit memory blocks to be released, are not destroyed. The objects are registered in the finalize queue and moved to the Java heap.

If objects in Explicit memory blocks, which are targeted for releasing, correspond to the following conditions, the operation will be different.

(a) If the objects are being referenced from outside (from a location other than the Explicit memory blocks that are targeted for releasing)

This corresponds to the case when the objects in the Explicit memory block, which are targeted for releasing, are being referenced from the objects in the following areas:

- Java heap
- Metaspace area
- Explicit memory block which is not targeted for releasing

Executed details in each case are described below.

If an object is being referenced from the Java heap or the Metaspace area

If an object is being referenced from an object in the Java heap or the Metaspace area, the object is not destroyed.

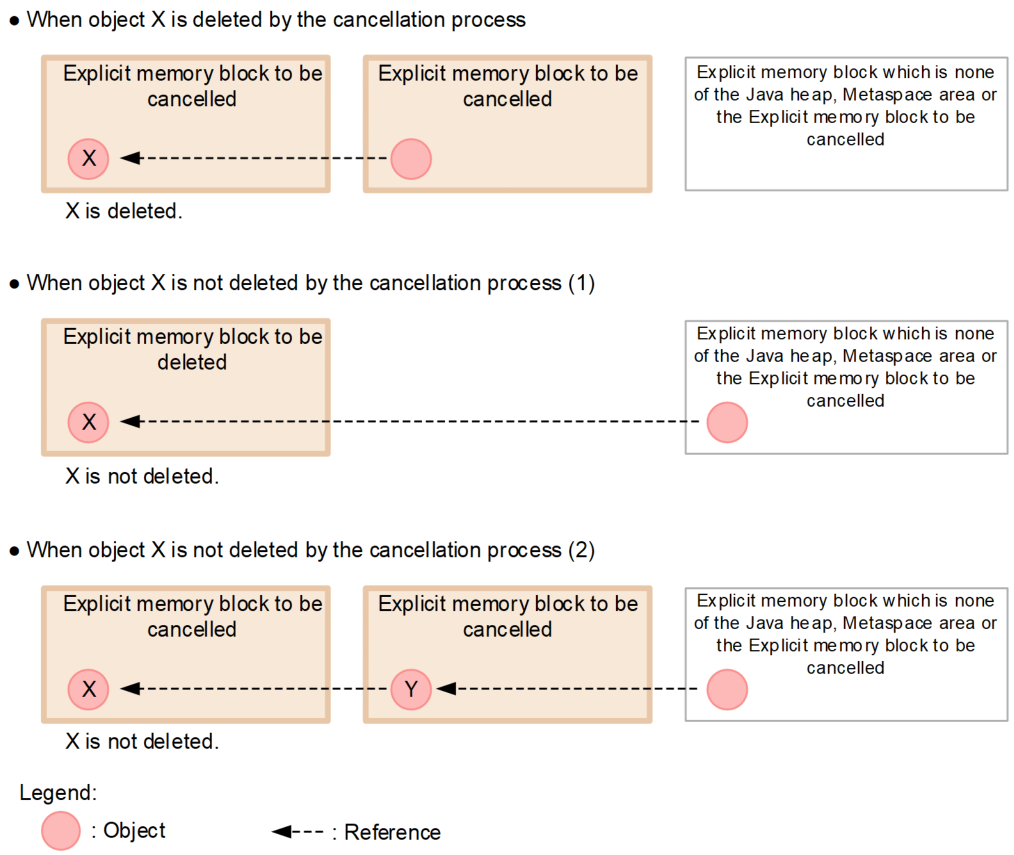
The corresponding objects are preferentially moved to a Tenured area in the Java heap However, the objects are moved to a New area if there is no free space in the Tenured area or if the Tenured area has overflown. Even if an object is being referred from an object in the Tenured area that is already not in use, the object is targeted for moving.

If the objects are being referenced from Explicit memory blocks that are not targeted for releasing

If an object is being referenced from an object in the Explicit memory block, which is not targeted for releasing, the object is not destroyed. Even if the reference source object is in the Explicit memory block that is targeted for releasing, if the object is going to be moved in the Java heap without destroying, the object being referenced is also not deleted.

The following figure shows the operation if an object is being referenced from outside when releasing the Explicit memory block.

Figure 7–13: Operation if an object is being referenced from outside when releasing the Explicit memory block



The description given below takes object X as an example. Object X is an object that is included in the Explicit memory block targeted for releasing.

If object X is being referenced from an object in the Explicit memory block, which is targeted for releasing, object X is deleted.

If object X is being referenced from the Java heap, Metaspace area, or from an object in an Explicit memory block that is not targeted for releasing, object X is not deleted.

Even when object X is being referenced from an object Y in an Explicit memory block that is targeted for releasing, if the reference source object Y is being referenced from an object in the Java heap, Metaspace area, or from an object in an Explicit memory block that is not targeted for releasing, object X is not deleted. In such cases, both objects X and Y move to the Java heap.

(b) If the Java heap overflows when moving objects to the Java heap

This corresponds to the case when an attempt is made to move objects being referenced from outside to the Java heap and an object cannot be moved because there is no free space in the movement destination Java heap.

In this case, Full GC is performed and free space is secured in the Java heap. The objects move to the Java heap after Full GC ends.

(c) If sufficient free space cannot be secured by Full GC

This corresponds to the case where the Java heap ran out of free space and the free space required for moving Java objects cannot be secured even by performing Full GC. In such cases, JavaVM aborts as it does in the case of insufficient C heap.

However, in the case of an insufficient C heap, the required memory size is output as *nnn* in the prompt `request` *nnn* `bytes`. When JavaVM aborts, 0 is always output as *nnn*.

7.9 Releasing Explicit memory blocks by using the javagc command

This section describes the releasing of the Explicit management heap by using the `javagc` command.

You can execute releasing of Explicit memory blocks by using the `javagc` command at any time. As a result, you can explicitly release the Explicit memory blocks, which were not released by release processing when the automatic release functionality is enabled.

7.9.1 Execution timing

Release processing is executed when you specify `-ehgc` option and execute `javagc` command.

Important note

Full GC is performed when Explicit memory blocks are released by using the `javagc` command. Hence, it is not appropriate for the processing related to running applications. We recommend that you execute release processing when the application is not running, such as at the time of undeploying and at nighttime.

7.9.2 Executed details

When the `javagc` command is executed, Full GC is performed by the Java VM and the string `EMJavaGC Command` is output as a cause of GC to extended `verbosegc` information. After that, the following Explicit memory blocks are released:

- Explicit memory blocks that are reserved by explicit release reservation, when automatic release functionality of the Explicit Memory Management functionality is enabled
- Explicit memory blocks generated by the explicit management heap automatic placement configuration file or `JavaVM`
- Explicit memory blocks that were not released in the previous release processing

For the strings that are output as causes of GC, see *-XX:[+|-]HitachiVerboseGCPrintCause (Option to output the cause of GC)* in the *uCosminexus Application Server Definition Reference Guide*.

Note that release processing is not performed in the following cases:

- **When you try to release Explicit memory blocks exceeding the maximum limit**
  This refers to the case when the number of existing Explicit memory blocks is 1,048,575.
- **When the Explicit Memory Management functionality is OFF**
  This refers to the case when `-XX:-HitachiUseExplicitMemory` option is specified.

In this case, although the constructor is successfully executed, memory blocks are handled as invalid Explicit memory blocks (ExplicitMemory instances).

7.10 Reducing time required for automatic release processing of Explicit memory blocks

This section describes the functionality, which reduces the time required for the processing of automatic release of Explicit memory blocks, when using automatic placement functionality of the Explicit Memory Management functionality. For reducing automatic release processing time, use the *functionality for controlling object movement to Explicit memory blocks*. In addition to this functionality, use the *functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality*. The functionality for controlling object movement to Explicit memory blocks is a prerequisite functionality for the functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality.

If you use the preceding functions, the objects referenced from automatically moved objects no longer move from the Java heap to Explicit memory blocks based on a reference relation when GC occurs. As a result, the Explicit heap area usage decreases. Thus, you can reduce the time required for the processing of automatic release of Explicit memory blocks. For details on movement based on reference relation of objects, see *7.6.5 Moving the objects from the Java heap to the Explicit memory block based on a reference relation*.

7.10.1 Checking whether the application is effective

The functionality for controlling object movement to Explicit memory blocks prevents objects from moving to the Explicit heap when Full GC occurs. You can determine whether application of this functionality is effective by checking Explicit memory block information included in the thread dump and event log of the Explicit Memory Management functionality. If the usage of Tenured area is less and Explicit memory blocks satisfying the following conditions are present, application of the functionality is effective. Therefore, review the use of the functionality.

- `EM_NAME` in Explicit memory block information is `null` (it is an Explicit memory block for which automatic release processing is executed once).
- If you compare the value of `EH_TOTAL` in Explicit memory block information with another Explicit memory block, it is an extremely large Explicit memory block.
- In the log data that was output to the event log for the Explicit Memory Management functionality when Full GC occurred, the `<EH_USED_AF>` value is much greater than the `<EH_USED_BF>` value.

The functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality is a functionality to specify the object, which is the cause, and not allow it to move to the Explicit heap when the processing of automatic release of Explicit memory blocks take times even by using the functionality for controlling object movement. If you apply the functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality, objects of the classes specified in the configuration file are excluded from application. You can determine whether application of this functionality is effective by checking Explicit memory block information included in the thread dump. If the usage of Tenured area is less and the Explicit memory block includes classes satisfying the following conditions, it means application of the functionality is effective and hence review the use of the functionality.

- *EM_NAME* in Explicit memory block information is *NULL* (it is an Explicit memory block for which automatic release processing is executed once).
- If you compare the value of *EH_TOTAL* in Explicit memory block information with other Explicit memory block, it is an extremely large Explicit memory block.
- Objects having large value of *ISIZE* in object statistics and low value of *FRATIO* in object release rate information in Explicit memory block information exist and its class is other than the classes provided by Java SE.

For details on the output contents of Explicit memory block information included in the thread dump, see *5.5 JavaVM Thread Dump* in the *uCosminexus Application Server Maintenance and Migration Guide.* For details on the event log of the Explicit Memory Management functionality, see *5.11 Event log of Explicit Memory Management functionality* in the *uCosminexus Application Server Maintenance and Migration Guide*.

7.10.2 Mechanism of reducing time required for automatic release processing

The Explicit Memory Management functionality suppresses occurrence of Full GC by moving long-life objects that become unnecessary in a certain period of time to the Explicit heap so that they will be reclaimed by release processing. In this mechanism, because survival time matches easily, the objects placed in an Explicit heap and the objects in reference relationship are moved from the Java heap to the Explicit memory block on the basis of a reference relationship and managed collectively in an Explicit heap.

However, depending on the operation, objects, which are moved to the Explicit memory block, might greatly increase size of the Explicit memory block and automatic release processing might take long time due to that. Because the system stops during automatic release processing of the Explicit memory block, long time of automatic release processing might create a problem in the system. Explicit memory blocks having large size are called huge blocks. Depending on the movement based on a reference relationship, the objects having different life span move to one block and repetition of this makes the block huge. When a reference relationship is complex and an application developer cannot understand it, huge blocks will generate easily.

Important note

The following table describes the types of Java objects. Life span of Java objects varies depending on the type and some objects might be appropriate and some not for placing in an Explicit heap.

Table 7–8: Types of Java object

Sr.No.	Category	Type of object	Release timing	Appropriate memory area for placement
1	Short-lived objects	Objects temporarily used in request processing and response processing	When copy GC occurs	Java heap (New area)#1
2	Long-lived objects	Objects that become unnecessary over a certain period of time	When executing automatic release processing	Explicit heap
3		Objects required for operating the application and used until the application stops	When application stops	Java heap (Tenured area)#2

#1 If you place the objects in the Explicit heap, generation and automatic release processing of an Explicit memory block occurs frequently and it causes overhead. Hence, an Explicit heap is not appropriate.

#2 If you place the objects in the Explicit heap, huge blocks generate and automatic release processing takes a long time. Hence the Java heap is not appropriate.

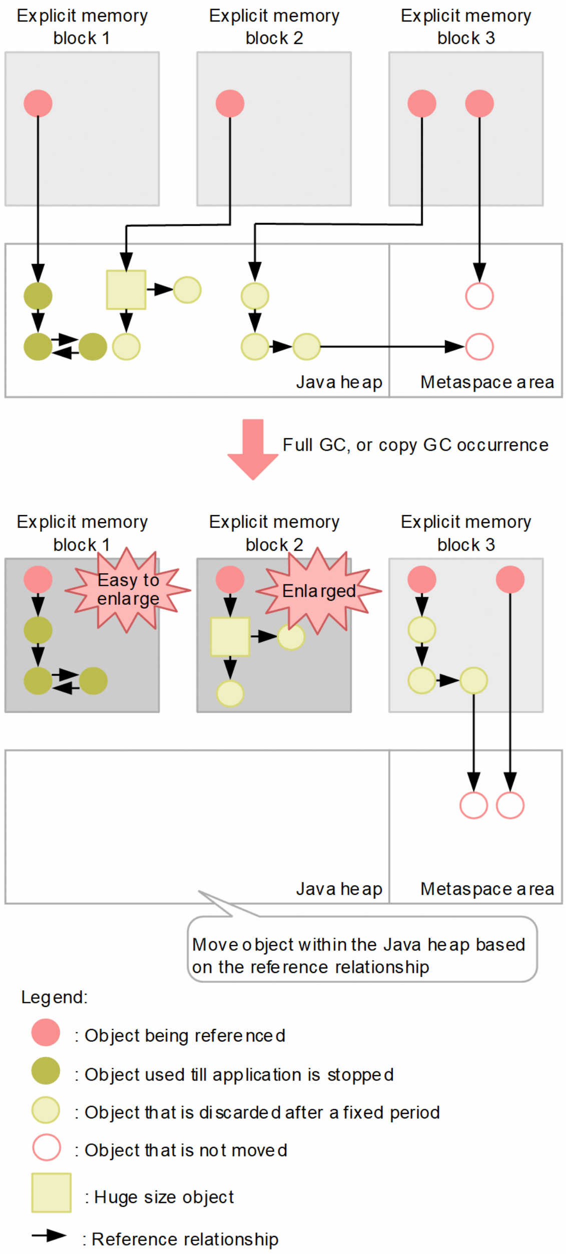
The following subsections describe the mechanism of reducing the time required for automatic release processing, by comparing the cases of using and not using the functionality.

(1) If not using the functionality for controlling object movement to the Explicit memory block and functionality for specifying classes to be

excluded from the application of the Explicit Memory Management functionality

This subsection describes the mechanism if you are not using the functionality for controlling object movement to the Explicit memory block and functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality. The following figure shows an example of three Explicit memory blocks that move objects on the basis of a reference relationship.

Figure 7–14: If not using the functionality for controlling object movement to the Explicit memory block and functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality

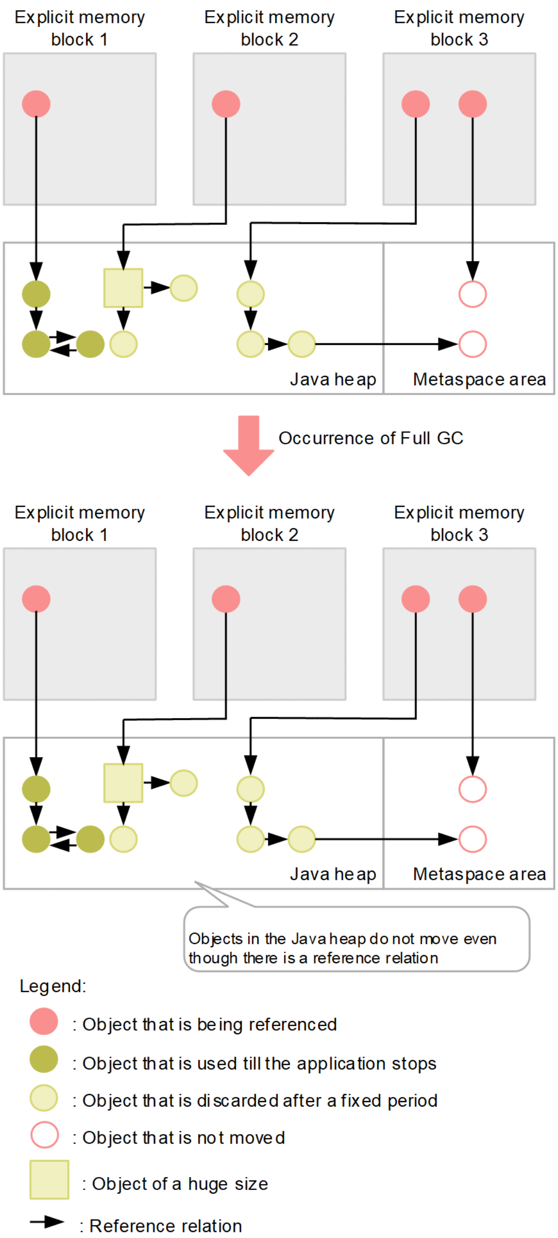


If neither functionality is used, objects are moved from the Java heap to Explicit memory blocks based on reference relations when GC occurs. In case of the Explicit memory block 1, moved object is the object, which is used until the application stops. Because this object is not reclaimed in automatic release processing, it continues to remain in the Explicit memory block. Because the total size of the object in the Explicit memory block 1 is not huge as shown in this figure, there is no problem at this stage. However, the objects might move to the Explicit memory block each time GC occurs, depending on the reference relations. Because the movement based on this reference relationship continues until the application stops, the Explicit memory block 1 might become a huge block. In the case of the Explicit memory block 2, because the moved object is huge, it becomes a huge block. Thus, if the objects that are not appropriate are placed in large numbers in the Explicit heap, the Explicit memory block becomes huge block and automatic release processing takes a long time.

(2) If using the functionality for controlling object movement to the Explicit memory block

This subsection describes the mechanism if you are using the functionality for controlling object movement to the Explicit memory block. The following figure shows an example of three Explicit memory blocks that move objects on the basis of a reference relationship.

Figure 7–15: If using the functionality for controlling object movement to the Explicit memory block



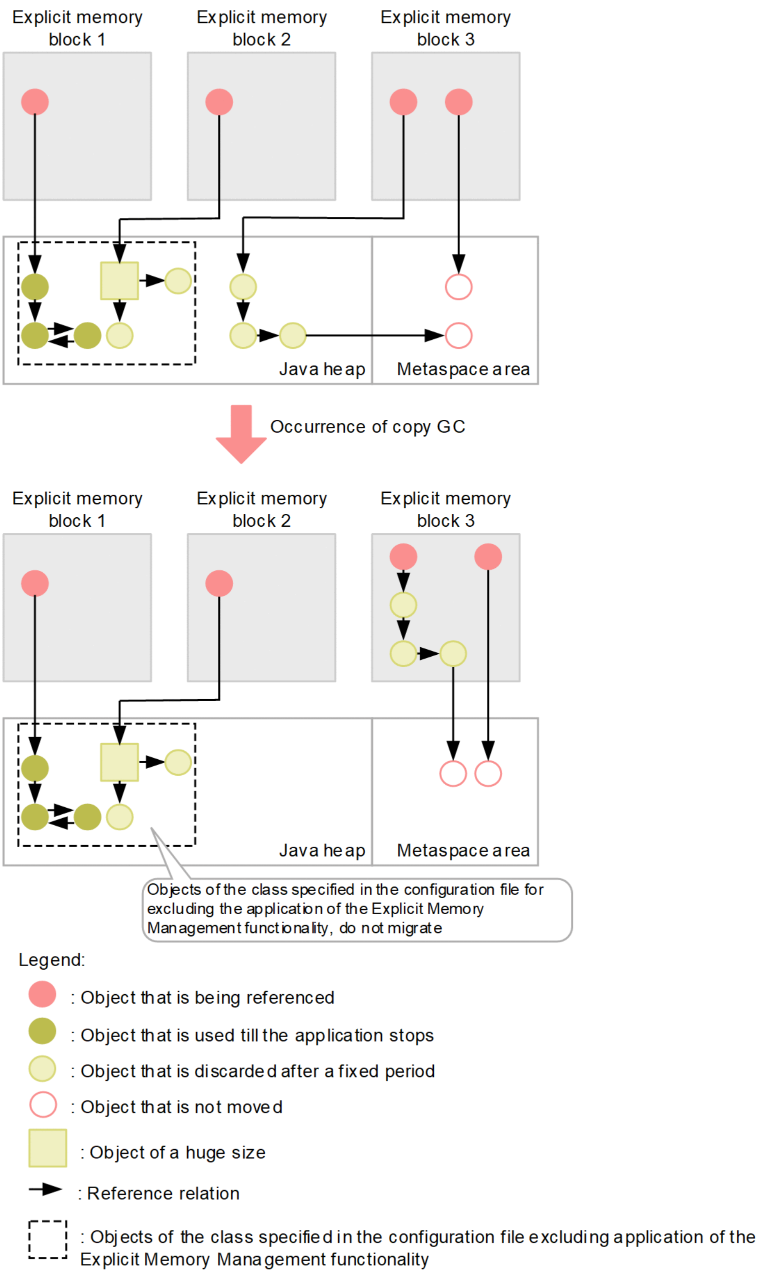
If the functionality for controlling object movement to Explicit memory blocks is used, no objects move from the Java heap to Explicit memory blocks based on reference relations even when Full GC occurs. If you use this functionality, you might have to reset the memory size of the Java heap area because the objects placed in the Java heap increase.

If huge blocks are generated even if you use this functionality, use the functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality.

(3) If using the functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality in addition to the functionality for controlling object movement to the Explicit memory block

This subsection describes the mechanism if you are using the functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality in addition to the functionality for controlling object movement to the Explicit memory block. The figure below shows an example of three Explicit memory blocks that move objects on the basis of a reference relationship. Here, assume that the classes of objects, which are moved to the Explicit memory block 1 and Explicit memory block 2, are set in the configuration file for Explicit Memory Management functionality application exclusion.

Figure 7–16: If using the functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality in addition to the functionality for controlling object movement to the Explicit memory block



The functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality provides the configuration file for Explicit Memory Management functionality application exclusion. The objects of the classes specified in this file are excluded from the targets of Explicit Memory Management. If this functionality is used, these objects do not move from the Java heap to Explicit memory blocks when copy GC occurs. These objects move to Tenured area at the time of rising. If you use this functionality, you might have to reset the memory size of the Java heap area because the objects placed in the Java heap increase.

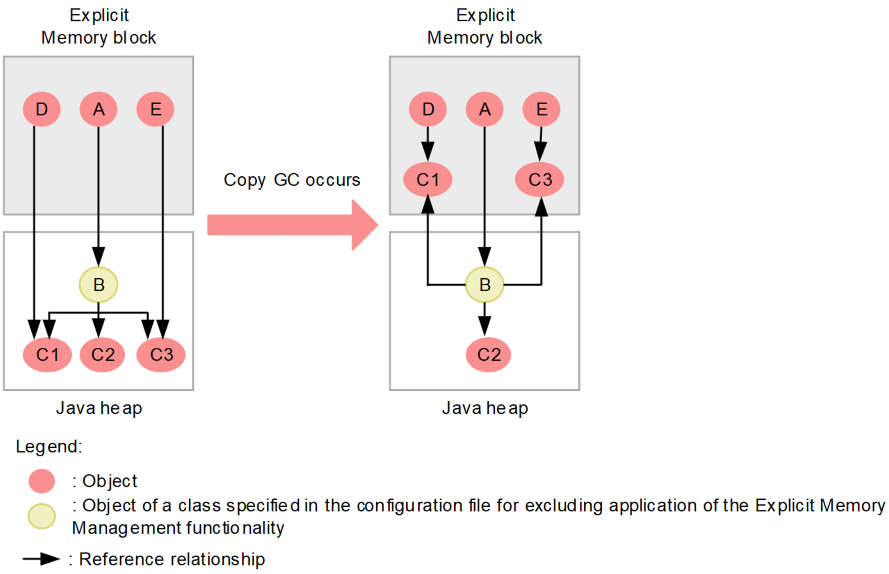
Configuration files for Explicit Memory Management functionality application exclusion used in the functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality have the following two types:

- Configuration file for Explicit Memory Management functionality application exclusion provided by the system (`sysexmemexcludeclass1100.cfg`)
- Configuration file for Explicit Memory Management functionality application exclusion, for which file path is provided in the option `-XX:ExplicitMemoryExcludeClassListFile` (`exmemexcludeclass.cfg` or any file name)

If you perform settings (specifying `-XX: ExplicitMemoryUseExcludeClass` option) for using the functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality, the classes specified in `sysexmemexcludeclass1100.cfg` are excluded from the application of the Explicit Memory Management functionality and the objects of those classes do not move to Explicit heap. If you want to specify more objects to be excluded from the application of the functionality, specify classes of the objects to be excluded from application in `exmemexcludeclass.cfg` or in a configuration file for Explicit Memory Management functionality application exclusion having any file name. If you want to apply Explicit Memory Management functionality to objects of classes specified in `sysexmemexcludeclass1100.cfg`, specify those classes in the configuration file for disabling application exclusion of the Explicit Memory Management functionality (`exmemnotexcludeclass.cfg`). Thus, you can reduce the objects to be moved to the Explicit heap depending on the classes specified in configuration file. The classes excluded from the application of the Explicit Memory Management functionality specify object release rate of the Explicit heap information, which is output in thread dump, for reference. For details on object release rate, see *7.10.3 Using object release rate information of the Explicit memory block*.

From among the objects, for which reference path passes through the objects excluded from the application of the Explicit Memory Management functionality, the objects that are not referenced from the path of objects other than Explicit Memory Management functionality application exclusion are also excluded from the application of the Explicit Memory Management functionality. The following figure shows an example of multiple reference paths from objects, which are targeted for exclusion from the application of the Explicit Memory Management functionality.

Figure 7–17: Example of having multiple reference paths from objects targeted for exclusion from the application of the Explicit Memory Management functionality



In this figure, object B is an object excluded from the application of the Explicit Memory Management functionality. The following are the reference paths passing through object B:

- A->B->C1
- A->B->C2

- A->B->C3

Among these, because there is a reference path D->C1 for object C1 and E->C3 for object C3, these objects move to the Explicit memory block on the basis of a reference relationship. On the other hand, because object C2 does not have a reference path other than the path passing through object B, it is excluded from the application of the Explicit Memory Management functionality and does not move to the Explicit memory block.

7.10.3 Using object release rate information of the Explicit memory block

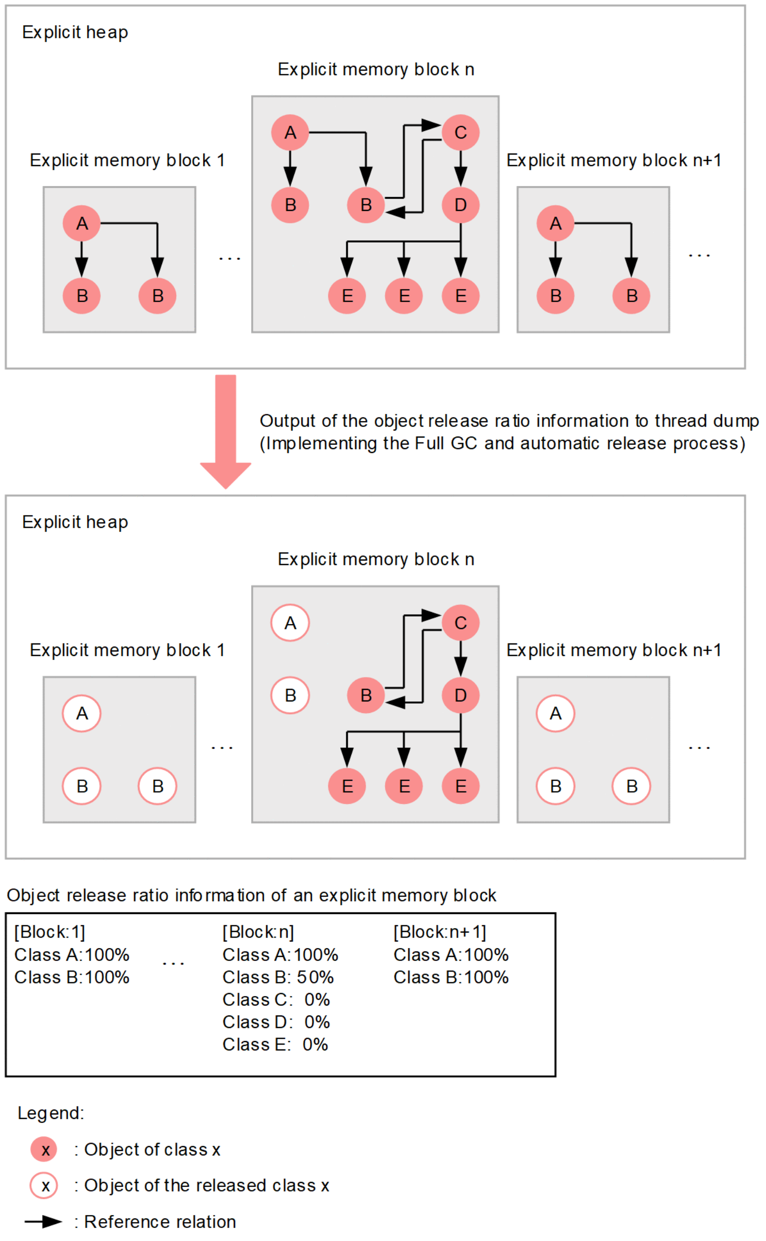
The huge blocks, which cause a long time for automatic release processing of the Explicit memory block, are generated because long-lived objects, which are used until the application stops, are generated and placed because a user program or framework is used. For effectively using the Explicit Memory Management functionality, you must identify the objects, which cause generation of these huge blocks, and make sure that those objects are not placed in the Explicit heap.

(1) Outputting object release rate information of Explicit memory block

You can identify the objects that cause generation of huge blocks if you use object release rate information. The object release rate information is the rate of objects released by automatic release processing of the Explicit memory block. It can be understood that the objects having low release rate in huge blocks are the objects, which cause generation of huge blocks. If you specify `freeratio` option in `eheapprof` command and execute the command, you can output the object release rate information in the Explicit heap information of extended thread dump. Refer to this information and specify classes of the objects in the configuration file for Explicit Memory Management functionality application exclusion. For details on how to specify in the configuration file for Explicit Memory Management functionality application exclusion, see *7.13.3 Controlling application target of the Explicit Memory Management functionality by using a configuration file*.

The following figure shows an example of output of object release rate information.

Figure 7–18: Example of output of object release rate information



In this figure, the Java VM performs Full GC and automatic release processing for Explicit memory blocks to obtain object release rate information. Because these processing might stop the application for few seconds, we recommend that you output the object release rate information when developing the system or when the work is stopped.

The object release rate information, which is output here, is calculated on the basis of the result of first automatic release processing that is generated when outputting the information. Hence, we recommend that you acquire the object release rate information multiple times for increasing the accuracy.

(2) Executed details

If you specify `-freeratio` option in `eheapprof` command and execute the command, JavaVM executes the following processing:

1. Performing Full GC

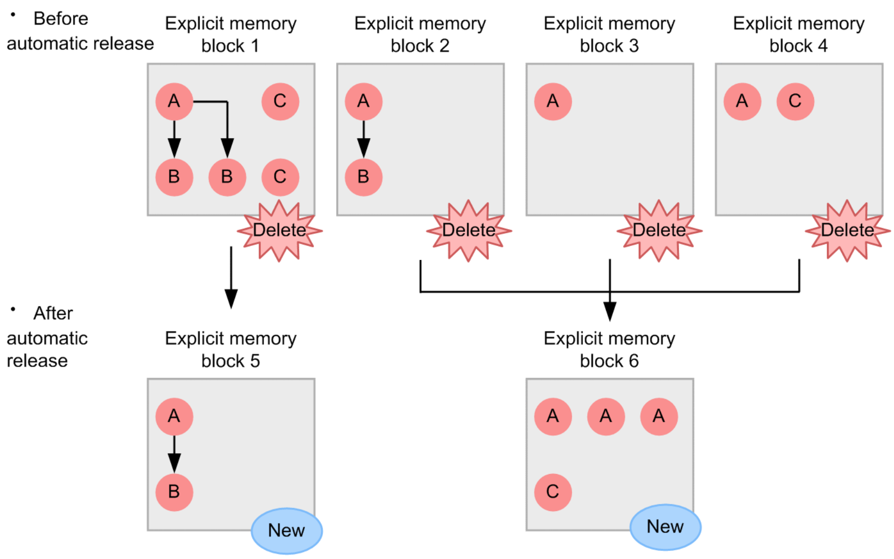
2. Automatic release reservation of Explicit memory blocks generated by automatic placement functionality and Explicit memory blocks, for which explicit release reservation is performed before executing `eheapprof` command

   - is output to object release rate of Explicit memory blocks, which were not targeted for automatic release reservation.
3. Automatic release processing of Explicit memory blocks, for which automatic release reservation described in step 2, is performed

   `JavaVM` obtains and retains the information of number of objects for each class in the Explicit memory block unit before and after automatic release processing.
4. Output of the object release rate information, which is calculated from the information acquired in step 3, to extended thread dump

The following figure shows an example of calculating object release rate information.

Figure 7–19: Example of calculating object release rate information



• Calculation result of information regarding the object release ratio of the Explicit memory block 5

Class name	Number of objects before automatic release	Number of objects after automatic release	Object release ratio (%)
A	1	1	0
B	2	1	50
C	2	0	100

• Calculation result of object release ratio information of the Explicit memory block 6

Class name	Number of objects before automatic release	Number of objects after automatic release	Object release ratio (%)
A	3	3	0
B	1	0	100
C	1	1	0

Legend:

x : Object of class x

→ : Reference relation

If there are objects that are referred from outside (the Explicit memory block not targeted for releasing), move the objects to a new Explicit memory block when executing automatic release processing. Similar to the Explicit memory block 6, if multiple Explicit memory blocks exist before automatic release, calculate object release rate from the total value of number of objects in multiple Explicit memory blocks and the number of objects in a new Explicit memory block.

For details on how to specify `eheapprof` command and output contents of Explicit heap information, see *eheapprof (Output of extended thread dump containing the Explicit heap detailed information)* in the *uCosminexus Application Server Command Reference Guide*.

7.10.4 Notes on reducing the time required for automatic release processing

This subsection describes points to be considered when using the functionality for controlling object movement to the Explicit memory block and functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality, which are used for reducing the time required for automatic release processing.

- In a case where objects in the Java heap are referenced from objects in Explicit memory blocks generated by the automatic placement functionality, the objects in the Java heap are moved to the Explicit heap when Full GC occurs. You can prevent objects from being moved by using the functionality for controlling object movement to Explicit memory blocks. In a system in which Full GC ordinarily occurs, however, enabling this functionality might increase the Full GC occurrence frequency. If frequent occurrence of Full GC becomes problematic for the system, retune the size of the memory space used by the Java VM. For details on memory tuning, see *7. JavaVM Memory Tuning* in the *uCosminexus Application Server System Design Guide*.
- The functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality does not let move objects of the classes specified in the configuration file, from among the objects of the Java heap referenced from objects in Explicit memory blocks, which are created by using automatic placement functionality, to Explicit heap. If you enable this functionality, because the objects that were moved to the Explicit heap move to Tenured area, the usage of Tenured area increases. As a result, the Full GC occurrence frequency might increase. If frequent occurrence of Full GC becomes problematic for the system, retune the size of the memory space used by the Java VM. For details on memory tuning, see *7. JavaVM Memory Tuning* in the *uCosminexus Application Server System Design Guide*.
- The functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality reads the configuration file when starting the J2EE server or the batch server. As a result, if you specify many classes in the configuration file, starting of the J2EE server or the batch server might take longer time. Compare the start time of the J2EE server or the batch server, and automatic release processing time, and review the use of this functionality.
- The functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality determines whether a class is excluded from the application of the Explicit Memory Management functionality, when loading the class. As a result, if you specify many classes in a configuration file, class loading time might increase.
- The functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality excludes objects of the classes specified in the configuration file from the target of the Explicit Memory Management functionality. Therefore, in this file, if you specify the class of objects that produce an effect when located in the Explicit heap, you might not gain the effect of suppressing occurrence of Full GC.
- Because the functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality does not execute *7.6.5 Moving the objects from the Java heap to the Explicit memory block based on a reference relation*, consider the objects as excluded from the application of the Explicit Memory Management functionality. As a result, if you specify direct generation of an object in an application, the object is generated in Explicit memory block even if you specify class of that object in the classes to be excluded from the application of the Explicit Memory Management functionality. For details on direct generation of objects in an application, see *7.6.3 Directly generating objects in the Explicit memory block*.

If you specify the objects stored in an HTTP session in the classes to be excluded from the application of the Explicit Memory Management functionality, the objects are excluded from the application of the Explicit Memory Management functionality and they do not move to the Explicit memory block. For details on the objects stored in an HTTP session, see *7.4.1 Objects related to HTTP session*.

- While GC is suppressed in cases such as when the application is being debugged, Full GC and automatic release processing cannot be performed. The `eheapprof` command executed with the `-freeratio` option while GC is suppressed cannot obtain object release rate information. In this case, in the thread dump you obtained, only the statistical information for the objects in Explicit memory blocks has been output as the Explicit heap detailed information.

7.11 Reducing memory usage of the Explicit heap that is used in an HTTP session

This section describes the functionality for reducing memory usage of the Explicit heap that is used in an HTTP session. Use the *memory saving functionality of the Explicit heap that is used in an HTTP session* to reduce the memory usage.

If you use this functionality, the relationship between an HTTP session and the Explicit memory block in Application Server becomes many-to-one. Utilization of the Explicit memory block improves because you can share a single Explicit memory block in multiple HTTP sessions. Thus, you can reduce memory usage of the Explicit heap that is secured by an HTTP session.

7.11.1 Checking whether the application is effective

You can determine whether the application of the memory saving functionality of the Explicit heap, which is used in an HTTP session, is effective by checking the Explicit memory block information included in the thread dump. The functionality is effective in the case of multiple Explicit memory blocks meeting the conditions described below. Therefore, review use of the functionality.

- *EM_TYPE* in Explicit memory block information is "R".
- *EM_USED* contains many Explicit memory blocks of the size smaller than those described below.

  If you are using the automatic placement functionality in the Explicit Memory Management functionality

  8 KB

  If you are not using the automatic placement functionality in the Explicit Memory Management functionality

  32 KB

For output contents of the Explicit memory block information in the thread dump, see *5.5 JavaVM Thread Dump* in the *uCosminexus Application Server Maintenance and Migration Guide*.

Estimation of the memory size of the Explicit heap area by using statistics, does not vary depending on whether this functionality is used. For details on how to estimate the memory size, see *7.11.4 How to estimate using statistical information* in the *uCosminexus Application Server System Design Guide.*

However, output contents of the statistics file partly differ. For details on the differences, see *7.11.3 Points to be considered when using the memory saving functionality of the Explicit heap that is used in an HTTP session*.

7.11.2 Mechanism of reducing memory usage

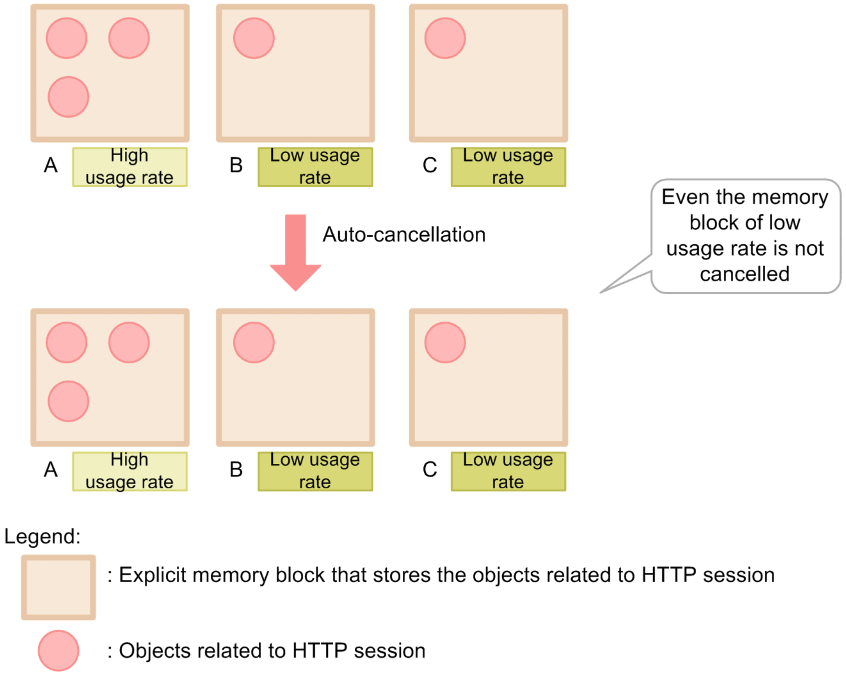
The Explicit memory block in the Explicit heap area, in which an HTTP session is stored, is created every time the application creates an HTTP session. Objects related to the HTTP session are placed in the created Explicit memory block.

This subsection describes the mechanism of reducing the memory usage by comparing the cases when you do not use this functionality and when you use this functionality.

(1) When you do not use the memory saving functionality of the Explicit heap that is used in an HTTP session

The following figure shows an example where there are three Explicit memory blocks in which objects related to an HTTP session are stored. From among these three blocks, B and C Explicit memory blocks are less utilized. The Explicit memory blocks are associated with the HTTP sessions that are not used for long time.

Figure 7–20: When you are not using the memory saving functionality of the Explicit heap that is used in an HTTP session



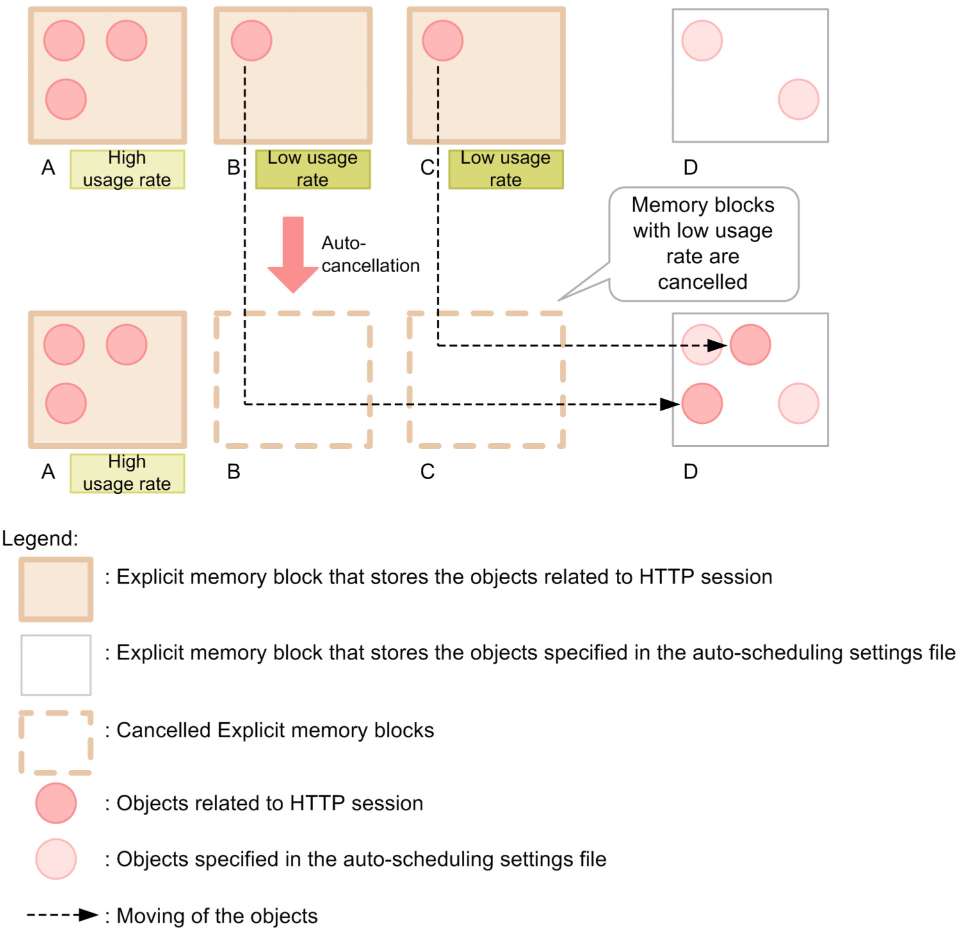
If you are not using the memory saving functionality of the Explicit heap that is used in an HTTP session, the Explicit memory blocks A-C are not released even when an automatic release occurs. In this case, used size of the Explicit memory block, in which objects related to an HTTP session are stored, matches with total bytes of the objects related to an HTTP session. Also, the number of Explicit memory blocks matches with the number of active sessions.

However, even when you store a small-sized object, the Explicit memory block is secured with a size that is above a fixed size. Therefore, the memory of the Explicit heap area, more than the Explicit memory size actually required, is consumed.

(2) When you use the memory saving functionality of the Explicit heap that is used in an HTTP session

The following figure shows an example where there are three Explicit memory blocks in which objects related to an HTTP session are stored. From among these three blocks, B and C Explicit memory blocks are less utilized. The Explicit memory blocks are associated with the HTTP sessions that are not used for a long time. The example also shows D as an Explicit memory block in which objects specified in an automatic placement configuration file are stored.

Figure 7–21: When you are using the memory saving functionality of the Explicit heap that is used in an HTTP session



If you are using the memory saving functionality of the Explicit heap that is used in an HTTP session, the memory saving functionality of the Explicit heap that is used in an HTTP session is executed when automatic release occurs. Explicit memory blocks B and C, which are less utilized, are released. Objects related to an HTTP session, which was stored in this Explicit memory block, move to D.

Thus, the utilization of Explicit memory blocks improves by moving and consolidating the objects stored in the less-utilized Explicit memory blocks to the other area and by releasing the less-utilized Explicit memory blocks.

7.11.3 Points to be considered when using the memory saving functionality of the Explicit heap that is used in an HTTP session

This section describes the points to be considered when using the memory saving functionality of the Explicit heap that is used in an HTTP session.

Output contents of statistics file partly differ depending on whether you are using the memory saving functionality of the Explicit heap that is used in an HTTP session.

Reference note

Due to the differences described here, the area that allocates the memory size of the objects related to an HTTP session, that is to be stored in the Explicit memory block to be released, changes from an area used in an HTTP session to an area used in the application. However, the memory size used by the entire system does not change. Therefore, whether you use this functionality does not impact the estimation of the memory size of the Explicit heap area performed using statistics.

This subsection describes the differences in the output contents.

(1) Number of Explicit memory blocks obtained by an HTTP session

The number of Explicit memory blocks output in the following items differs:

- `HTTPSessionEMemoryBlockCount.HighWaterMark`
- `HTTPSessionEMemoryBlockCount.LowWaterMark`
- `HTTPSessionEMemoryBlockCount.Current`

If you are not using this functionality

The number of active HTTP sessions in the system is output.

If you are using this functionality

Because a value reflecting an internal operation is output, the value output is different than the number of active HTTP sessions in the system.

(2) Size of the Explicit heap area used in the application

The size of the Explicit heap area output in the following items differs:

- `ApplicationEHeapSize.HighWaterMark`
- `ApplicationEHeapSize.LowWaterMark`

If you are not using this functionality

The size of Explicit memory used in the automatic placement functionality is output.

If you are using this functionality

The total of "size of Explicit memory that is targeted for automatic release by this functionality + size of Explicit memory used in the automatic placement functionality" is output.

7.12 Implementing the Java program that uses the Explicit Memory Management functionality API

This section describes the implementation if you use the Explicit Memory Management functionality in your application. Implement the Explicit Memory Management functionality by using the Explicit Memory Management functionality API.

You can use the Explicit Memory Management functionality in classes in the `JP.co.Hitachi.soft.jvm.MemoryArea package`. For details on the API, see the *uCosminexus Application Server API Reference Guide*. Note that all the Explicit Memory Management functionality APIs are thread-safe.

You can implement the following two types of processes in the Explicit Memory Management functionality API:

- Implementing to place objects in the Explicit heap
- Implementing to obtain statistics of the Explicit Memory Management functionality

7.12.1 Implementing to place objects in the Explicit heap

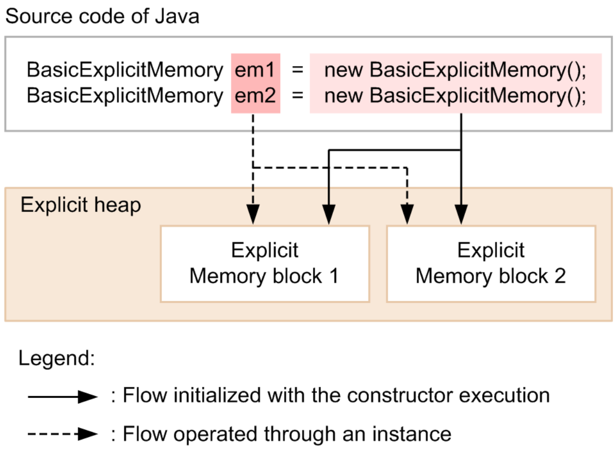
This subsection describes the overview of classes of the Explicit Memory Management functionality API and the basic way using the API.

(1) Relationship between the ExplicitMemory instance and the Explicit memory block

The Explicit memory blocks in the Explicit heap map 1:1 to the instances of the ExplicitMemory class handled by the Explicit Memory Management functionality API.

The following figure shows the relationship between the ExplicitMemory instance and the Explicit memory block.

Figure 7–22: Relationship between ExplicitMemory instance and Explicit memory block



The Explicit memory block is initialized if you execute a constructor of the ExplicitMemory class. Thereafter, the initialized instance becomes an interface for operating Explicit memory blocks. Instance em1 in the figure maps to the Explicit memory block 1 while instance em2 maps to the Explicit memory block 2.

(2) Class configuration of the Explicit Memory Management functionality API

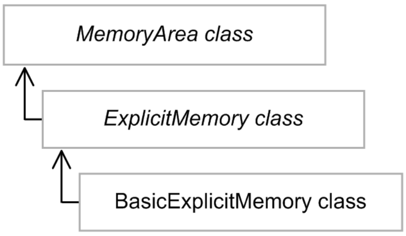
The following table describes classes in the Explicit Memory Management functionality API.

Table 7–9: Classes in the Explicit Memory Management functionality API

Class	Explanation
`MemoryArea` class	This is an abstract class that represents the Java heap or the Explicit memory block.
`ExplicitMemory` class	This is an abstract class that represents Explicit memory block. Use functionalities of this class through the `BasicExplicitMemory` class, which is a derived class.
`BasicExplicitMemory` class	This is a class that represents Explicit memory block handled in an application. In application, implement the Explicit Memory Management functionality by using the API of this class.

The following figure shows the class hierarchy.

Figure 7–23: Class hierarchy in the Explicit Memory Management functionality API



Note: *Part in Italics* indicates an abstract class.

(3) How to use the Explicit Memory Management functionality API

The mapping of basic operations and methods is as follows:

- Directly generating the objects in Explicit memory block
  Use the `newArray` method or `newInstance` method.
- Releasing Explicit memory blocks
  Use the `reclaim` method.

An example of using the BasicExplicitMemory class is described below. This example creates two Explicit memory blocks.

Example of using the BasicExplicitMemory class

```java
BasicExplicitMemory em1 = new BasicExplicitMemory();
BasicExplicitMemory em2 = new BasicExplicitMemory();
Object o1 = em1.newInstance(Object.class);
Object o2 = em2.newInstance(Object.class);
ExplicitMemory.reclaim(em1);
```

The BasicExplicitMemory instances are generated on lines 01 and 02. In this example, em1 maps to the Explicit memory block 1 while em2 maps to the Explicit memory block 2.

Directly generate the objects in the Explicit memory block by using the `newInstance` method on lines 04 and 06. On line 04, place an object of the Object class in the Explicit memory block 1 by invoking the `newInstance` method from `em1` instance. On line 06, place an object of the Object class in the Explicit memory block 2 by invoking the `newInstance` method from em2 instance.

Destroy the Explicit memory block when it becomes unnecessary. An execution of the `ExplicitMemory.reclaim(em1)` method on line 08 is a processing to release the Explicit memory block 1. The processing releases the Explicit memory block 1 and at the same time, destroys object o1 that was generated on line 04. Note that when releasing Explicit memory blocks, entire area corresponding to the appropriate Explicit memory block is targeted for release and not the objects.

In this example, survival period of the Explicit memory block 1 is from line 01 to line 08.

7.12.2 Implementing to obtain statistics of the Explicit Memory Management functionality

This section describes the implementation for obtaining statistics of the Explicit Memory Management functionality in your application. You can debug and analyze failures by obtaining statistics.

If you have implemented the Explicit Memory Management functionality in your application, you can obtain the following information as statistics:

- Status of using the Explicit heap
- Explicit memory block size represented by the ExplicitMemory instance
- Information of Explicit memory block

Also, you can execute the following processes as processes associated with obtaining statistics:

- Setting and obtaining name of Explicit memory block
- Determining whether Explicit memory block is processable
- Determining whether Explicit memory block can be reserved for release

This subsection describes the implementation of each process that uses the Explicit Memory Management functionality API.

(1) Obtaining the status of using the Explicit heap

This point describes how to obtain information of the Explicit heap. The Explicit heap represents all Explicit memory blocks. For details on how to obtain information of each Explicit memory block, see *(3) Obtaining information of Explicit memory block*.

Used API

`getMemoryUsage()`

This API creates an instance of the `java.lang.management.MemoryUsage` class and returns the instance.

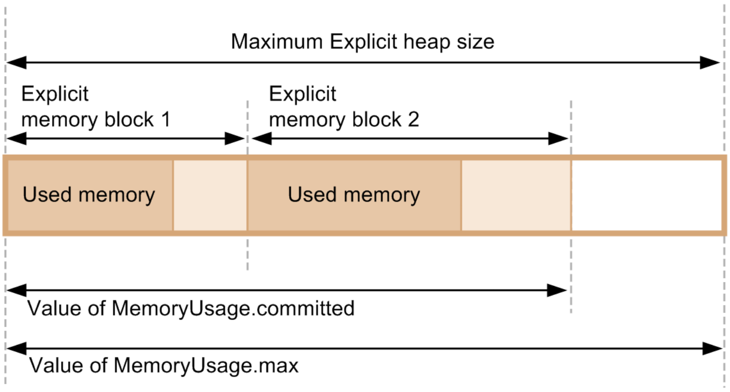
Information described in the following table is set in each field in the returned instance, as the information at the time of creating the instance.

Table 7–10: Information in each field (instance of MemoryUsage class)

Field	Setting details
init	0
used	Used memory size of the Explicit heap (units: bytes)
committed	Secured size of the Explicit heap (units: bytes)
max	Maximum Explicit heap size specified in `-XX:HitachiExplicitHeapMaxSize` (units: bytes) However, the value is 0 if the Explicit Memory Management functionality is OFF (if `-XX:-HitachiUseExplicitMemory` is set)

The following figure shows the values shown by each field.

Figure 7–24: Values shown by each field (instance of MemoryUsage class)



Note: Value of MemoryUsage.used is the total value of 'used memory'.

(2) Explicit memory block size represented by ExplicitMemory instance

Obtain the Explicit memory block size that is represented by the `ExplicitMemory` instance, as the status of using Explicit memory block. By using the Explicit memory block size, you can check the status of using the memory in the Explicit Memory Management functionality.

Used API

- `freeMemory()`
- `usedMemory()`
- `totalMemory()`

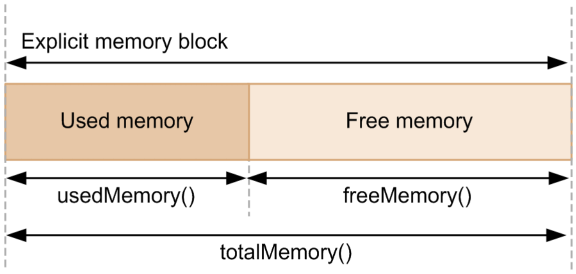
The following table describes the status of using Explicit memory block that you can obtain with each API. You can obtain the size as a long type value.

Table 7–11: Status of using Explicit memory block that you can obtain with each API

API	Status of using Explicit memory block that you can obtain
`freeMemory()`	Usable memory size (units: bytes)
`usedMemory()`	Used memory size (units: bytes)
`totalMemory()`	Total secured size (units: bytes)

The following figure shows the values that you can obtain with each API.

Figure 7–25: Values that you can obtain with each API



(3) Obtaining information of Explicit memory block

Obtain the number of the Explicit memory blocks that have an entity in the Explicit heap. The released or invalid Explicit memory blocks are not targeted. If you obtain the number of valid Explicit memory blocks, you can calculate the average memory size used in each Explicit memory block.

Used API

`countExplicitMemories()`

This API counts the number of memory blocks in the Explicit heap and returns it as a value of the `int` type. The Explicit memory blocks meeting the conditions are targeted for counting.

- Valid Explicit memory blocks
  Valid Explicit memory blocks are targeted irrespective of their sub-status (Enable or Disable).
- Explicit memory blocks that are reserved for releasing

(4) Setting and obtaining name of the Explicit memory block

You can set name to an instance corresponding to the Explicit memory block. You can also obtain the set name. An Explicit memory block instance has a name for easy handling in an application. An instance becomes easy to use if you set a name to it.

The set value is also output in an event log of the Explicit Memory Management functionality.

Used API

- `setName()`
  This API sets name.
- `getName()`
  This API obtains name.

If you do not set a name in your application, the following default name is set.

```
BasicExplicitMemory-<ID>
```

ID is a value managed by JavaVM.

Important note

When naming an Explicit memory block, do not use a name starting with "CCC#". The J2EE server uses names starting with "CCC#".

The J2EE server uses the following Explicit memory block names:

- `CCC#HttpSession`
  It is an Explicit memory block that places an HTTP session.
- `CCC#HttpSessionManager`
  It is an Explicit memory block that places the objects for managing an HTTP session.

(5) Determining whether the Explicit memory block is processable

The Explicit memory block might become non-processable in cases such as failure in securing memory. You can determine whether an Explicit memory block is processable.

Used API

`isActive()`

The following table describes the mapping of the status of the Explicit memory block (`ExplicitMemory` instance), when the API is invoked, and the return value of the API.

Table 7–12: Mapping of the status of the Explicit memory block when isActive() is invoked and the return value of the API

Status of Explicit memory block	Sub-status	Return value
Released	--	`false`
Invalid	--	`false`
Reserved for releasing	--	`false`
Valid	Enable	`true`
	Disable	`false`

Legend:
--: Not applicable

(6) Determining whether the Explicit memory block can be reserved for release

You can refer to the `ExplicitMemory` instance corresponding to an Explicit memory block even after the Explicit memory block is reserved for releasing or released. You can check the status of the `Explicit memory` instance from an application by using the API.

Used API

`isReclaimed()`

The following table describes the mapping of the status of the Explicit memory block (`ExplicitMemory` instance), when the API is invoked, and the return value of the API.

Table 7–13: Mapping of the status of the Explicit memory block when isReclaimed() is invoked and the return value of the API

Status of Explicit memory block	Sub-status	Return value
Released	--	`true`

Status of Explicit memory block	Sub-status	Return value
Invalid	--	`true`
Reserved for releasing	--	`true`
Valid	Enable	`false`
	Disable	`false`

Legend:

--: Not applicable

7.13 Settings in the execution environment

This section describes the settings in the execution environment for using the Explicit Memory Management functionality.

> **Important note**
>
> On the J2EE server, objects related to an HTTP session are set to be placed in the Explicit heap area by default.
>
> Before starting the operations, estimate the required Explicit heap memory size and tune JavaVM option (`-XX:HitachiExplicitHeapMaxSize` option) to an appropriate value. For details on the tuning of the Explicit heap, see *7.3.2 Tuning procedure* in the *uCosminexus Application Server System Design Guide*.

7.13.1 Common settings for using the Explicit Memory Management functionality (setting JavaVM options)

This subsection describes common settings for using the Explicit Memory Management functionality.

You need to do the following settings when using the Explicit Memory Management functionality:

- J2EE server
- Batch server
- Java application
- Automatic placement configuration file settings

If you want to control the application target of the Explicit Memory Management functionality, you need to perform the following setting:

- Setting configuration file for Explicit Memory Management functionality application exclusion

(1) J2EE server settings

Perform the J2EE server settings in an Easy Setup definition file.

For the common settings for using the Explicit Memory Management functionality, specify the JavaVM options in the JavaVM startup parameter (`add.jvm.arg`), which is inside the `<configuration>` tag of the logical J2EE server (j2ee-server) in an Easy Setup definition file.

The following table describes the definitions of the JavaVM options in the Explicit Memory Management functionality.

Table 7–14: Definitions of the JavaVM options in the Explicit Memory Management functionality

JavaVM option	Setting details	Default value
`-XX:` `[+|-]HitachiUseExplicitMemory`	The option sets whether you want to use the Explicit Memory Management functionality.	In the case of new installation The value depends on the execution environment[#1].

JavaVM option	Setting details	Default value
		IN the case of version upgrade -XX:-HitachiUseExplicitMemory
-XX:[+|-]HitachiAutoExplicitMemory	The option sets whether you want to use the automatic placement functionality in the Explicit Memory Management functionality.	-XX:-HitachiAutoExplicitMemory
-XX:HitachiAutoExplicitMemoryFile:String	This option specifies the path of the automatic placement configuration file that is used when using the automatic placement functionality in Explicit Memory Management functionality.	Empty string
-XX:[+|-]HitachiExplicitMemoryAutoReclaim	The option specifies whether you want to use the automatic release functionality in the Explicit Memory Management functionality.	-XX:+HitachiExplicitMemoryAutoReclaim
-XX:[+|-]HitachiExplicitMemoryCompatibleToV8	This option specifies whether to use the same method as 08-00 for securing Explicit memory blocks. Enable this option if you do not want to use new functions in 08-50 or later, and want to run an application, which runs in 08-00, as is in 08-50.	-XX:-HitachiExplicitMemoryCompatibleToV8
-XX:HitachiExplicitHeapMaxSize[#2]	This option sets maximum size of the Explicit heap area. (units: bytes)	-XX:HitachiExplicitHeapMaxSize =64m
-XX:HitachiExplicitMemoryLogLevel:String	In *String*, set the log level of event log to be output by the Explicit Memory Management functionality. Specify one of the following values: none normal verbose debug	-XX:HitachiExplicitMemoryLogLevel:normal
-XX:HitachiExplicitMemoryJavaLog:String	In *String*, specify the output destination path name of event log to be output by the Explicit Memory Management functionality.	In Windows -XX:HitachiExplicitMemoryJavaLog:Cosminexu-installation-directory\CC\server\public\ejb\<server name>\logs In UNIX -XX:HitachiExplicitMemoryJavaLog:/opt/Cosminexus/CC/server/public/ejb/<server name>/logs
-XX:HitachiExplicitMemoryJavaLogFileSize=*Integer value*	In *Integer value,* specify file size of event log. (units: bytes)	-XX:HitachiExplicitMemoryJavaLogFileSize =4m
-XX:ExplicitMemoryFullGCPolicy=*Numeric-value*	This option specifies whether to control object movement from the Java heap to Explicit memory blocks based on reference relations when Full GC occurs.	-XX:ExplicitMemoryFullGCPolicy=0

JavaVM option	Setting details	Default value
`-XX:[+|-]ExplicitMemoryUseExcludeClass`	The option specifies whether you want to enable the functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality.	Recommended mode: `-XX:+ExplicitMemoryUseExcludeClass`
`-XX:ExplicitMemoryExcludeClassListFile:`*String*	The option specifies the path of the configuration file for Explicit Memory Management functionality application exclusion to be used when using the functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality.	Empty string
`-XX:ExplicitMemoryNotExcludeClassListFile` *String*	The option specifies the path of configuration file for disabling application exclusion of the Explicit Memory Management functionality to be used when using the functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality.	Empty string

#1

In the case of a new installation, the default value varies according to the execution environment as described below.

In the case of a J2EE server

`-XX:+HitachiUseExplicitMemory`

In the case of a batch server and a Java application

`-XX:-HitachiUseExplicitMemory`

#2

For details on the estimation, see *7.11 Explicit heap tuning* in the *uCosminexus Application Server System Design Guide*.

An example of definitions in an Easy Setup definition file is given below.

```
<param-name>add.jvm.arg</param-name>
 <param-value>-Xms512m</param-value>
 <param-value>-Xmx512m</param-value>
 <param-value>-XX:+HitachiUseExplicitMemory</param-value>
 <param-value>-XX:HitachiExplicitHeapMaxSize=64m</param-value>
```

Reference note

You can also perform the J2EE server settings from the [Setting startup parameters] screen (defining the logical J2EE server) in the management portal. For details on how to perform the settings in the management portal, see *10.8.23 Setting startup parameters (J2EE server)* in the *uCosminexus Application Server Management Portal User Guide*.

For details on the JavaVM options to be specified and tags to be specified in an Easy Setup definition file, see the *uCosminexus Application Server Definition Reference Guide*.

(2) Batch server settings

Perform the batch server settings in an Easy Setup definition file.

For the common settings for using the Explicit Memory Management functionality, specify the JavaVM options in the JavaVM startup parameter (`add.jvm.arg`), which is inside the `<configuration>` tag of the logical J2EE server (j2ee-server) in an Easy Setup definition file.

For the JavaVM options to be specified, see *(1) J2EE server settings*.

(3) Java application settings

Perform the settings of the Java application that you run using the `cjclstartap` command, in the option definition file for Java applications (`usrconf.cfg`).

For the common settings for using the Explicit Memory Management functionality, specify the JavaVM options in the JavaVM startup parameter (`add.jvm.arg`) in the option definition file for Java applications (`usrconf.cfg`).

For JavaVM options to be specified, see *(1) J2EE server settings*.

An example of definitions in the option definition file for Java application (`usrconf.cfg`) is given below.

```
add.jvm.arg=-Xms512m
add.jvm.arg=-Xmx512m

add.jvm.arg=-XX:+HitachiUseExplicitMemory
add.jvm.arg=-XX:HitachiExplicitHeapMaxSize=64m
```

For details on the option definition file for Java applications (`usrconf.cfg`), see *12.2.1 usrconf.cfg (Option definition file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

(4) Automatic placement configuration file settings

If you want to use the Explicit Memory Management functionality by using an automatic placement configuration file, you need to specify the `-XX:+HitachiAutoExplicitMemory` option and set the objects that you want to place in the Explicit heap.

Specify the objects that you want to place in the Explicit heap, in the `AutoExplicitMemoryText` parameter, which is inside the `<configuration>` tag of the logical J2EE server (J2EE-Server) in an Easy Setup definition file.

An example is given below.

```
 :
<param>
<param-name>AutoExplicitMemoryText</param-name>
<param-value>
<![CDATA[
com.sample.*, java.util.ArrayList
com.sample.Main.main(java.lang.String[]), java.util.LinkedList
]]>
</param-value>
</param>
 :
```

For details on how to create an automatic placement configuration file, see *7.13.2 Using the Explicit Memory Management functionality by using the automatic placement configuration file*.

You can also perform the automatic placement configuration file settings from the [Setting startup parameters] screen (defining logical J2EE server) in the management portal or from any user file (file specified in the `-XX:HitachiAutoExplicitMemoryFile` property).

(5) Setting configuration file for Explicit Memory Management functionality application exclusion

If you use a configuration file for Explicit Memory Management functionality application exclusion and control application of the Explicit Memory Management functionality to the objects to be moved on the basis of a reference relation, you must specify the following options and set the classes not to be moved to the Explicit heap.

- `-XX:ExplicitMemoryFullGCPolicy=0`
- `-XX:+ExplicitMemoryUseExcludeClass`

Code the setting of classes not to be moved to the Explicit heap in the configuration file for Explicit Memory Management functionality application exclusion.

The following is the coding example:

```
com.sample.ClassA
com.sample.ClassB
```

If you want to move some classes from among the classes, which are coded in the configuration file for Explicit Memory Management functionality application exclusion, to the Explicit heap, code the classes, for which the settings of application exclusion of the Explicit Memory Management functionality are to be disabled, in configuration file for disabling application exclusion of the Explicit Memory Management functionality.

For details on how to create a configuration file for Explicit Memory Management functionality application exclusion and configuration file for disabling application exclusion of the Explicit Memory Management functionality, see *7.13.3 Controlling application target of the Explicit Memory Management functionality by using a configuration file*.

7.13.2 Using the Explicit Memory Management functionality by using the automatic placement configuration file

Set the automatic placement functionality in the Explicit Memory Management functionality by using the automatic placement configuration file. If you use the automatic placement configuration file, you can use the Explicit Memory Management functionality without making changes to the Java program.

In the automatic placement configuration file, specify objects that you want to place in the Explicit heap and the location of generating the objects. Note that the objects that are being referenced from the objects that you specify in this file (objects generated in the Explicit memory block) move to the Explicit memory block from the Java heap. For details on the object movement, see *7.6.5 Moving the objects from the Java heap to the Explicit memory block based on a reference relation*.

This subsection describes the coding format of the automatic placement configuration file and points you need to consider during coding, if you use the Explicit Memory Management functionality by specifying the `-XX:+HitachiAutoExplicitMemory` option and by using the automatic placement configuration file.

You can code the contents of the automatic placement configuration file by using one of the following items:

- Easy Setup definition file

- [Setting startup parameters] screen (defining the logical J2EE server) in the management portal
- Any user file (file specified in the `jvm.userprf.File` property)

(1) Coding format of the automatic placement configuration file

Coding format of the automatic placement configuration file is as follows.

```
<Generation point>#, <fully classified class name of the specified object> #
Comment
:
<Generation point>#, <fully classified class name of the specified object>
```

#

Examples of the specification of generation point are described below.

Example of generation point	Meaning
*	This specifies generation of the user-specified objects in all the methods that are included in all classes in all packages, as the generation point.
`com.sample.*`	This specifies generation of the user-specified objects in the methods that are included in classes in all packages that start with com.sample, as the generation point. Therefore, if lower packages (`com.sample.abc` or `com.sample.abc.test`) exist, those are also targeted.
`com.sample.Main`	This specifies generation of the user-specified objects in all the methods (including constructor and static initializer) that are included in `com.sample.Main` class, as the generation point.
`com.sample.Main.main(java.lang.String[])`	This specifies generation of the user-specified objects in the `main(java.lang.String[])` method that is defined in the com.sample.Main class, as the generation point.

Tip

- A single byte space (*0x20*) or tab (*\t* or *0x09*) is the blank character that separates syntax elements.
- One or more line feed characters (*\n* or *0x0A*) or return characters (*\r* or *0x0D*) continue at end-of-line.
- Comments start with # and all characters in between # and end-of-line are considered as a comment.
- * character in generation point indicates all classes existing in same or subpackages. Meaning of * in import declaration of Java language and * in generation point differs on the point that * in generation point targets classes in subpackages also.

(2) Example of coding the automatic placement configuration file

An example of coding the automatic placement configuration file is described below.

```
# comment
com.sample.*, java.util.ArrayList # comment
com.sample.Main.main(java.lang.String[]), java.util.LinkedList
```

This point describes contents of the coding example.

1. All first lines are comments.

2. Specify in such a way that the java.util.ArrayList object, generated by the class and method, which is included in all packages starting with com.sample. * is placed in the Explicit memory block. Text from # until end-of-line is considered as a comment.
3. Specify in such a way that the `java.util. LinkedList` object generated by the `com.sample.Main.main(java.lang.String[])` method is placed in the Explicit memory block.

> **Reference note**
>
> You can describe the entries that specify a class in JavaVM (for example, class in packages starting with java, javax), as a generation point for the user-specified object. However, specified entries might be treated as if they do not exist. If an entry is treated as if it does not exist, an error message is not output in the Explicit Management Heap log.

(3) Notes for the automatic placement configuration file

The following point describes the points to be considered when you specify the automatic placement configuration file.

- If you use the automatic placement functionality, class loading time increases. As a result, the JavaVM starting time or application deployment time on Application Server might increase.
- Using the automatic placement functionality might increase the copy GC processing time.
- The automatic placement functionality targets only those objects that are generated with `new`. The functionality does not target the objects generated with JNI or reflection.
- Describe all class names and method arguments, including classes in the `java.lang` package in a fully classified class name.

  For example:

  Incorrect: `String`

  Correct: `java.lang.String`
- You cannot describe a class name that uses generics. Describe the class names (raw type) that are not parameterized.

  For example:

  Incorrect: `java.util.HashMap<java.lang.String, java.lang.Object>`

  Correct: `java.util.HashMap`
- Describe nested class names separated by "$" and not ".".

  For example:

  Incorrect: `java.util.AbstractMap.SimpleEntry`

  Correct: `java.util.AbstractMap$SimpleEntry`
- For constructor, code the same method name as the class name or *init*. For the constructor of MyMain class, code as shown below.

  For example:

  `MyMain.MyMain()` or `MyMain.`*init*`()`
- If a method having the same name as a class exists, it is not possible to determine whether you are specifying a constructor or a method. Therefore, it is treated as if you have specified both a constructor and method.

  For example:

  A constructor containing an `int` argument of `MyMain.MyMain(int)` # `MyMain` class and a # `MyMain(int)` method are both generation points

- Describe *clinit* for static initializer. For static initializer of the `MyMain` class, code as shown below.

  For example:

  `MyMain.`*clinit*`()`

- If you want to specify generation of an object, by assigning at the time of field declaration, as the generation point, describe default constructor in generation point.
- You cannot specify an array in a fully classified class name of the user-specified object.

  For example:

  `java.lang.String[]`

- If a line containing a non-existing class name, method name or a method (native method and abstract method) not having byte code exists, treat that line as if it does not exist
- If you specify an internal class of J2SE in class name of the user-specified object, the Explicit Memory Management functionality might be replaced by an appropriate class name. For example, `java.util.HashMap` is replaced with `java.util.HashMap$Entry`.
- If you specify a huge class or method, which is close to the limit of the Java language specifications, as a generation point, automatic placement might fail. In that case, "Invalid class file format" is output as *MESSAGE* of explicit management heap automatic placement error in event log of the Explicit Memory Management functionality. In such cases, review size reduction of class or method.

7.13.3 Controlling application target of the Explicit Memory Management functionality by using a configuration file

If objects in the Java heap are referenced from objects in Explicit memory blocks generated by the automatic placement functionality, the objects in the Java heap are moved to the Explicit heap based on reference relations when GC occurs. The functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality excludes the objects, which are to be moved on the basis of this reference relation, from application target of the Explicit Memory Management functionality by using a configuration file and does not let the objects move to Explicit heap. You can use this functionality to exclude objects that are not reclaimed even by Full GC, such as objects used until the application stops, from the targets of the Explicit Memory Management functionality. For details on the movement based on reference relation of objects, see *7.6.5 Moving the objects from the Java heap to the Explicit memory block based on a reference relation*.

(1) Types of configuration files

The following two types of files are used in the functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality:

- **Configuration file for Explicit Memory Management functionality application exclusion**

  Specify classes of the objects, which you do not want to move to an Explicit heap. The objects of the classes specified in this file do not move to the Explicit heap when GC occurs. The objects move to Tenured area at the time of rising.

  A configuration file for Explicit Memory Management functionality application exclusion contains files provided by the system. If you enable the functionality for specifying classes to be excluded from the application of the Explicit Memory Management functionality, the configuration file for Explicit Memory Management functionality application exclusion provided by the system is used. The following is the file path of the configuration file for Explicit Memory Management functionality application exclusion, which is provided by the system:

  In Windows

  JDK-installation-directory`\lib\explicitmemory\sysexmemexcludeclass1100.cfg`

In UNIX

`/opt/Cosminexus/jdk/lib/explicitmemory/sysexmemexcludeclass1100.cfg`

If you want to add classes for exclusion from application target of the Explicit Memory Management functionality, update the file in the following file path or create a new file:

In Windows

JDK-installation-directory`\usrconf\exmemexcludeclass.cfg`

In UNIX

`/opt/Cosminexus/jdk/usrconf/exmemexcludeclass.cfg`

If you create a new configuration file for Explicit Memory Management functionality application exclusion, specify the file path in `-XX:ExplicitMemoryExcludeClassListFile` option.

- **Configuration file for disabling application exclusion of the Explicit Memory Management functionality**

  Specify the classes, for which the settings of application exclusion are to be disabled, from among the classes specified in the configuration file for Explicit Memory Management functionality application exclusion. The objects of the classes specified in this file move to the Explicit heap when GC occurs.

  If you want to disable the classes excluded from application target of the Explicit Memory Management functionality, update the file in the file path given below or create a new file. You can also disable settings of the classes specified in the configuration file for Explicit Memory Management functionality application exclusion, which is provided by the system.

  In Windows

  JDK-installation-directory`\usrconf\exmemnotexcludeclass.cfg`

  In UNIX

  `/opt/Cosminexus/jdk/usrconf/exmemnotexcludeclass.cfg`

  If you create a new configuration file for disabling application exclusion of the Explicit Memory Management functionality, specify file path in `-XX:ExplicitMemoryNotExcludeClassListFile` option.

(2) Specifying configuration file and scope of application of the Explicit Memory Management functionality

Priority is given to specification of configuration file for disabling application exclusion of the Explicit Memory Management functionality than the specification of configuration file for Explicit Memory Management functionality application exclusion.

This subsection describes specification of configuration file and scope of application of the Explicit Memory Management functionality considering *com.sample* package as an example. *com.sample* package contains two classes - `ClassA` and `ClassB`. Specify each configuration file as shown below.

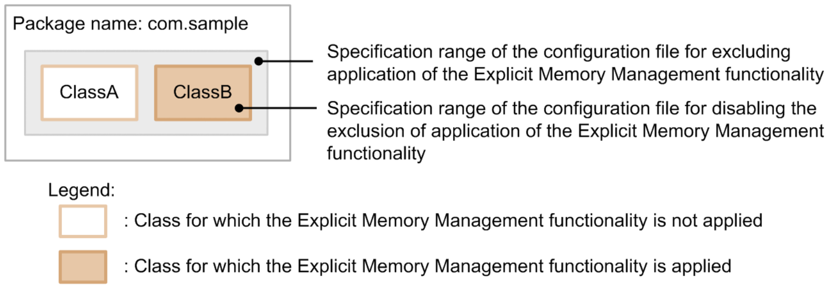
- **Example of specifying a configuration file for Explicit Memory Management functionality application exclusion**

```
com.sample.*
```

- **Example of specifying a configuration file for disabling application exclusion of the Explicit Memory Management functionality**

```
com.sample.ClassB
```

Both ClassA and ClassB are included in specification of the configuration file for Explicit Memory Management functionality application exclusion. However, because specification of the configuration file for disabling application exclusion of the Explicit Memory Management functionality is preferred, only `ClassA` is excluded from the application of the Explicit Memory Management functionality and Explicit Memory Management functionality is applied to `ClassB`, as shown in the following figure.



(3) Format for coding a configuration file

The following is the format for coding a configuration file.

- **When using a type other than array**

```
Fully-qualified-class-name-of-specified-class# #Comment
:
Fully-qualified-class-name-of-specified-class#
```

#

You can omit the class name by using *.

- **When using array type**

```
[-part-in-number-of-dimensions-of-array#L Fully-qualified-class-name-of-sp
ecified-class;
```

#

In case of a multi-dimensional array, specify *[* continuously for the number of dimensions. In case of three-dimensional array, it will be *[[[*.

(Example) In case of one-dimensional array of `aaa.bbb.Myclass` class

`[Laaa.bbb.Myclass;`

Tip

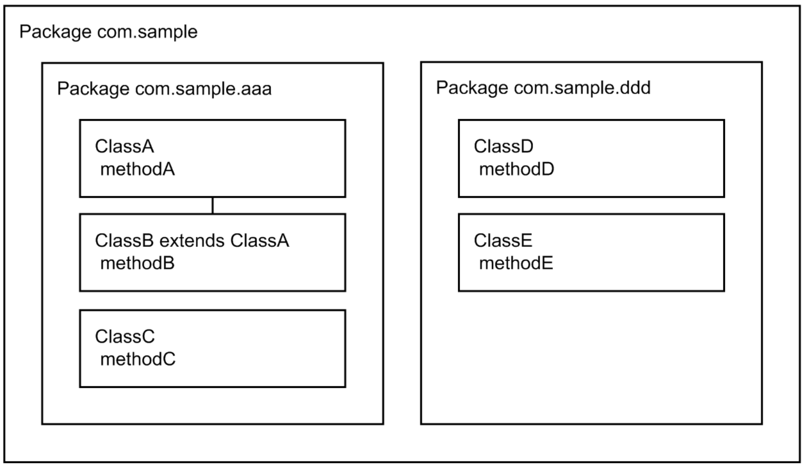
- Code one class name on a line.
- You can code up to 1,024 characters on one line. This number includes null characters and comments. If you code 1,025 or more characters on one line, parsing fails, a warning message is output, the line is ignored, and read processing continues.
- You can omit the class name if you specify *package-name*`.*`. Unlike import declaration * in Java language, classes in sub-package are also targeted.
- One or more line feed characters *(\n* or *0x0A*) or recovery characters (*\r* or *0x0D*) are considered as the end-of-line.
- Blank characters are considered as single space character (*0x20*) or tab characters (*\t* or *0x90*). If you code blank characters in the configuration file, those are ignored.
- A comment starts with # and all characters starting from # to end-of-line are considered as a comment.

(4) Coding example of configuration file

Coding examples of the configuration file for Explicit Memory Management functionality application exclusion and configuration file for disabling application exclusion of the Explicit Memory Management functionality are described below.

The coding example described here is a class structure shown in the following figure having package name as *com.sample*.

Figure 7–26: Example of class structure



- **When specifying in a fully qualified class name**

  The following is an example of a configuration file for Explicit Memory Management functionality application exclusion when specifying in a fully qualified class name:

  ```
  com.sample.aaa.ClassA
  com.sample.aaa.ClassC
  com.sample.ddd.ClassD
  ```

  In this example, the objects of `ClassA` class, `ClassC` class, and `ClassD` class move to Tenured area.

- **When specifying by omitting the class name**

  The following are coding examples of a configuration file for Explicit Memory Management functionality application exclusion and configuration file for disabling application exclusion of the Explicit Memory Management functionality when specifying by omitting class name.

  - **Coding example of configuration file for Explicit Memory Management functionality application exclusion**

    ```
    com.sample.*
    ```

  - **Coding example of configuration file for disabling application exclusion of the Explicit Memory Management functionality**

    ```
    com.sample.aaa.ClassB
    com.sample.ddd.ClassE
    ```

  In this example, not only the classes in the same package but all the classes including the classes in subpackages are targeted to move to Tenured area because of the coding in the configuration file for Explicit Memory Management functionality application exclusion. However, the objects of `ClassB` class and `ClassE` class are targeted to move to the Explicit memory block because of the coding in the configuration file for Explicit Memory

Management functionality application. Hence, the objects of `ClassA` class, `ClassC` class and `ClassD` class move to Tenured area.

> **Tip**
>
> About whether to specify in fully qualified class name or by omitting class name, we recommend that you specify in such that the amount of coding in a configuration file is less. Both coding examples have same control. In this case, specifying by omitting class name is preferable.

7.13.4 Settings for using the function on the J2EE server

This subsection describes settings for using the Explicit Memory Management functionality on the J2EE server. Set whether to target the following objects for placement in the Explicit heap, as the J2EE server properties:

- Objects related to HTTP session

By default, objects are set to be placed in the Explicit heap. However, if you change the settings in the JavaVM option described in *7.13.1 Common settings for using the Explicit Memory Management functionality (setting JavaVM options)* to not to use the Explicit Memory Management functionality, property settings of the J2EE server become invalid.

(1) How to set

Perform the J2EE server settings in an Easy Setup definition file. Specify definitions of the Explicit Memory Management functionality in the `<configuration>` tag of the logical J2EE server (j2ee-server) in an Easy Setup definition file.

The following table describes definitions of the Explicit Memory Management functionality in an Easy Setup definition file.

Table 7–15: Definitions of the Explicit Memory Management functionality in an Easy Setup definition file

Parameter to be specified	Setting details
`ejbserver.server.eheap.httpsession.enabled`	Specify whether to place the objects related to an HTTP session in the Explicit heap.

For details on the parameters to be specified, see *4.11 Parameters applicable to logical J2EE servers* in the *uCosminexus Application Server Definition Reference Guide*.

Relationship between the JavaVM options and each property is described below.

Relationship between the JavaVM options and ejbserver.server.eheap.httpsession.enabled property

The placement destination of the objects related to an HTTP session varies according to the values specified in the prerequisite JavaVM options and the `ejbserver.server.eheap.httpsession.enabled` property. The following table describes the placement destination of objects related to an HTTP session.

Table 7–16: Placement destination of the objects related to an HTTP session according to the values of the JavaVM options and the ejbserver.server.eheap.httpsession.enabled property

JavaVM option	Value of ejbserver.server.eheap.httpsession.enabled property	Placement destination
`-XX:+HitachiUseExplicitMemory`	`true`	Explicit heap area
	`false`	Java heap area
	Other value (such as an incorrect string or value not specified)	Explicit heap area
`-XX:-HitachiUseExplicitMemory`	`true`	Java heap area
	`false`	
	Other value (such as an incorrect string or value not specified)	
Not specified	`true`	Java heap area
	`false`	
	Other value (such as an incorrect string or value not specified)	

(2) An example of definitions in an Easy Setup definition file

An example of definitions in an Easy Setup definition file is given below.

- **An example of definitions in an Easy Setup definition file**

```
<configuration>
  <logical-server-type>j2ee-server</logical-server-type>
  <param>
    <param-name>ejbserver.server.eheap.httpsession.enabled</param-name>
    <param-value>true</param-value>
  </param>
      :
 </configuration>
```

7.14 Precautions for using the Explicit Memory Management functionality

This section describes the points to be considered when using the Explicit Memory Management functionality.

7.14.1 Setting initial size and maximum size of the Java heap

You must specify the same value in the initial size (-Xms) and maximum size (-Xmx) of the Java heap. If you specify different values, the copy GC occurrence frequency might increase.

We recommend this setting even if you do not use the Explicit Memory Management functionality.

Supplement:

If the initial size and maximum size of the Java heap are different, sizes of all areas change at the following timings:

- When copy GC ends
  The size of the New area changes dynamically.
- When Full GC ends
  The sizes of the Tenured area and New area change dynamically.

The size of the New area is mainly determined according to the Tenured area size and the value specified in the `-XX:NewRatio` option.

If occurrence of Full GC is suppressed by the Explicit Memory Management functionality, the system loses the opportunity to resize the Tenured area. With this, size of the New area becomes almost fixed.

Therefore, even if you specify a maximum size that is larger than the initial size, the timing of the extension of the New area is lost and the size is the value that you specified as the initial size. If you specify a small value for the initial size of the New area, copy GC might occur more frequently than when the Explicit Memory Management functionality is not used.

7.14.2 Notes on using Explicit heap in objects related to an HTTP session

- Since an HTTP session is generated, all session attributes (objects) set using the `setAttribute` method are not released and remain in Explicit heap until you destroy the HTTP session. At that time, it is irrespective whether you have set it in the HTTP session. Therefore, if you repeat executions of the `setAttribute` method without destroying the HTTP session, an Explicit heap overflow might occur, preventing you from gaining the effect of suppressing Full GC. For confirming whether the Explicit heap overflow has occurred, see *7.14.3 Checking and measures when there is an overflow from the Explicit heap* in the *uCosminexus Application Server System Design Guide*.
- If you are not using the automatic release functionality (in the case of `-XX:-HitachiExplicitMemoryAutoReclaim`), when deleting an HTTP session, objects stored in the session to which there are references from outside, move from the Explicit heap to the Tenured area of the Java heap. Because this increases the size of the used Tenured area, occurrence of Full GC cannot be suppressed.
  To prevent movement of objects from the Explicit heap to the Java heap, you must delete references to the objects stored in an HTTP session before deleting the session.
  The same applies to the case where references to objects, which are obtained by using the following API, are remaining:
  - `getAttribute(String)`
  - `getAttributeNames()`

Note that if you are using the automatic release functionality (in the case of -XX:+HitachiExplicitMemoryAutoReclaim), the objects do not move to the Tenured area of the Java heap.

- In the following cases, objects are placed in the Java heap and not in the Explicit heap:
  - If you generate a new session when the number of Explicit memory blocks has reached the maximum value and you cannot create new Explicit memory block (when 1,048,575 Explicit memory blocks are existing at the same time)
  - If the maximum size of the Explicit heap area is exceeded
  - If you could not secure the Explicit memory block

  In the preceding cases, you might not be able to suppress occurrence of Full GC because objects are created in the Java heap.
- In JSP, the HttpSession object is implicitly created, by default. To prevent the Explicit heap overflow caused by generation of unnecessary `HttpSession` objects, perform the settings in such a way that the `HttpSession` object is not explicitly created in JSPs that do not require session. Use the session attribute in page directive to perform the setting.
- When you evaluate the effect of suppressing occurrence of Full GC, do not run a test under conditions where successive session creations continue without destroying the session. Because Explicit memory blocks are not released, there is a high possibility of Explicit heap overflow.

  Note that for Explicit memory blocks, a reservation for release is made when the session is destroyed. The Explicit memory blocks are released when GC occurs subsequently. If session destruction and generation are repeated too many times within one GC interval, creation of new Explicit memory blocks can occur while reservations for releasing Explicit memory blocks remain. As a result, the number of Explicit memory blocks increases and the Explicit heap might overflow.

  When you evaluate the effect of suppressing occurrence of Full GC, run a test under conditions where sessions are managed appropriately.
- The objects, which you have stored in a session, are placed in the Java heap immediately after generation. After copy GC occurs several times, normally, the objects move to the Explicit heap at the time when they are promoted to the Tenured area. Therefore, if objects are deleted in a short time or if the session is quickly destroyed, the objects are not placed in the Explicit heap.

7.14.3 Maximum number of characters in the name of the Explicit memory block to be output to the thread dump

The name of the Explicit memory block is output to Explicit heap details that are output to the thread dump of `JavaVM`. The maximum limit for the number of characters in the name of an Explicit memory block is 2,000 characters. If you set a name of an Explicit memory block exceeding 2,000 characters in the `setName` method of the `JP.co.Hitachi.soft.jvm.MemoryArea` class, the name part which exceeds 2,000 characters is not output to the thread dump.

8 User Log Output for Applications

This chapter gives an overview of the log output for J2EE applications, batch applications, and EJB client applications. This chapter also describes the log output methods.

8.1 Organization of this chapter

The logs, output by J2EE applications, batch applications, and EJB client applications, are called *user logs*. This chapter describes the user logs output for applications.

When an error occurs, you collect and analyze the user log that is output, and examine the causes of the error. You can acquire user logs in a batch as the `snapshot` log or acquire an individual log. For details on collecting the `snapshot` log containing user logs, see *2.3.3 Collecting the Snapshot Log* in the *uCosminexus Application Server Maintenance and Migration Guide*.

The following table describes the organization of this chapter.

Table 8–1: Organization of this chapter (User log output for applications)

Category	Title	Reference location
Description	Overview of the user log output	*8.2*
	Log format	*8.3*
Implementation	Methods used in the user log output	*8.4*
	Implementation for user log output	*8.5*
Setup	Creating and setting loggers and handlers	*8.6*
	How you can use your own filter/ formatter/ handler	*8.7*
	Setting the user log output for J2EE applications	*8.8*
	Setting the user log output for batch applications	*8.9*
	Setting the user log output for EJB client applications (For using the cjclstartap command)	*8.10*
	Implementing and setting the user log output for EJB client applications (For using the vbj command)	*8.11*
Notes	Notes for using the user log functionality	*8.12*

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

The reference location of the user log output varies according to the type of applications. The following table describes the reference locations.

Table 8–2: Reference locations related to the user log output

Reference location	Type of application		
	J2EE application	Batch application	EJB client application
8.2 Overview of the user log output	Y	Y	Y
8.3 Log format	Y	Y	Y
8.4 Methods used in the user log output	Y	Y	Y
8.5 Implementation for user log output	Y	Y	N
8.6 Creating and setting loggers and handlers	Y	Y	N
8.7 How you can use your own Filter/ formatter/ handler	Y	Y	N
8.8 Setting the user log output of J2EE applications	Y	N	N

Reference location	Type of application		
	J2EE application	Batch application	EJB client application
8.9 Setting the user log output of batch applications	N	Y	N
8.10 Setting the user log output of EJB client applications (When using the cjclstartap command)	N	N	Y
8.11 Implementing and setting the user log output of EJB client applications (When using the vbj command)	N	N	Y
8.12 Notes for using the user log functionality	Y	Y	Y

Legend:

Y: Reference

N: Do not reference

8.2 Overview of the user log output

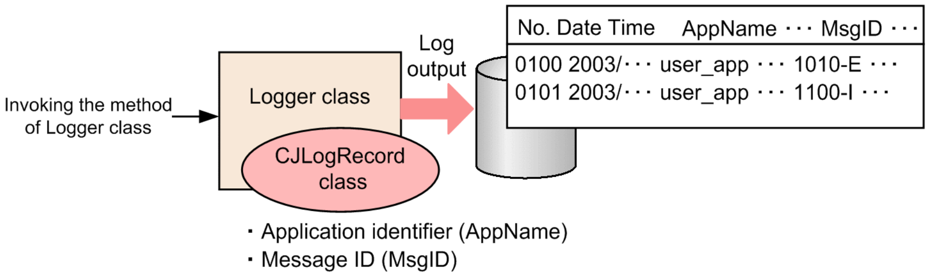
This section gives an overview of the user log output.

8.2.1 Overview of the user log output

The log, output by J2EE applications, batch applications, and EJB client applications, is called *user log*. With Cosminexus, you can output user logs in the Hitachi Trace Common Library format (*user log functionality*). Therefore, you can handle system logs and application logs in the same format, and this results in improving the reliability of log operations of the entire system.

The following figure shows the procedure for log output by using the user log functionality.

Figure 8–1: Procedure of the user log functionality



You use the J2SE standard log output functionality (*Java logging API*) to output user logs. For using this functionality, execute the user log output with the Java logging APIs.

Reference note

You cannot execute the user log output from a resource adapter. Note that you can execute the user log output from a Message-driven Bean that is invoked from the resource adapter.

8.2.2 Mechanism of the user log output

You can use the Java logging APIs of J2SE for executing J2EE applications, batch applications, and EJB client applications to output user logs. A Java logging API is a highly versatile API that can output multiple items such as memories, consoles, and files. However, the logic is simple, so for applying the logic to a mission critical system, an application developer must implement reliable and durable classes for log output.

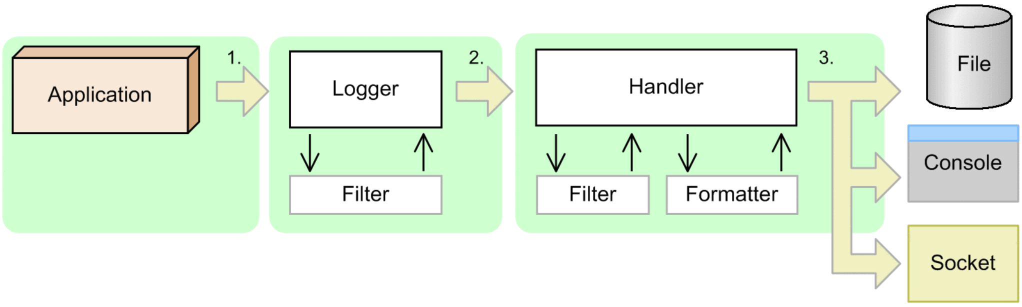
If you use the user log functionality, a highly reliable user log can be output, even if the classes for the log output are not implemented by the application developer.

You can output logs, which are output from J2EE applications, batch applications, and EJB client applications developed by using the Java logging APIs, in a format output by a component software of another Cosminexus system (*Hitachi Trace Common Library format*) by using *Hitachi Trace Common Library*. By using this library you can handle user logs in the same format as the other system logs, and the log operation can be highly reliable and unified.

You execute the user log output in accordance with the mechanism of the J2SE Java logging. With the Java logging, you use two types of elements, *loggers* and *handlers*. Note that loggers and handlers are the objects of the `Logger` class and the `Handler` class respectively.

The following figure shows the mechanism of Java logging.

Figure 8–2: Mechanism of Java logging



The following points describe the above figure:

1. Output user logs from applications by using logger.

   The user logs are output by using the methods of the `Logger` class, when the application is processing.

2. Logger adds additional information such as levels and message strings to the log, which is output from the application, converts to `LogRecord`, and passes to the handler.

   At that time, you can have fine control above the control specified as a log level, by using the filter (object of the `Filter` class) that is connected to the logger.

3. Based on the received `LogRecord`, the handler outputs logs to a file, console or a socket.

   You specify the output destination and the output format as a handler property, in advance. In handler, you can have fine control by using the filter connected to the handler. You can also use the formatter (object of the `Formatter` class) to output messages formatted in any format.

With Application Server, the file handlers are provided to output logs in the Hitachi Trace Common Library format, in a file. The following sub-sections describe the provided file handlers for respective applications.

- **For J2EE applications or batch applications**

  `CJMessageFileHandler` is provided as a file handler. You can specify the log output destination files, log levels, number of log files, and filters and formatters to be used for `CJMessageFileHandler`, when setting up a system. For details on setting the user log output of J2EE applications and batch applications, see *8.8 Setting the user log output of J2EE applications* and *8.9 Setting the user log output of batch applications*.

  In a user log, if you want to output the message ID corresponding to the application name and message contents that are output by the log, you must implement the settings in J2EE applications or batch applications. In such cases, implement the settings by using a class for the extended `LogRecord` provided by Application Server (*CJLogRecord class*). For details on how to use the `CJLogRecord` class, see *8.4 Methods used in the user log output*. For details on the APIs of the `CJLogRecord` class, see *7. APIs used in the user log functionality* in the *uCosminexus Application Server API Reference Guide*.

  Important note

  For directly implementing the settings of handlers and loggers in J2EE applications, you must have the security permission `LoggingPermission("control")` for the application to be executed. For details on how to specify the security permission `LoggingPermission("control")`, see *8.8.2 Setting security policy*.

- **For EJB client applications**

`CJMPMessageFileHandler` is provided as a file handler. How to set up a user log of an EJB client application varies according to the command used to start the EJB client application. For details on how to set up the user log output for EJB client applications, see *8.10 Setting the user log output of EJB client applications (When using the cjclstartap command)* or *8.11 Implementing and setting the user log output of EJB client applications (When using the vbj command)*.

8.3 Log format

When you use the user log functionality, log is output in the following format:

Number Date Time	AppName	pid tid	MsgID	Message text	CRLF

The following table describes the output contents of items in the above format.

Table 8–3: Log format

Field	Output contents
`Number`	The serial number of trace code (four digits) is output. The number starts from `0000` and returns to `0000`, when the number reaches `9999`.
`Date`	The date (`yyyy/mm/dd` format), during the output, is output.
`Time`	The time (`hh:mm:ss.nnn` format), during the output, is output.
`AppName`	The application-distinguished name is output. You specify the application-distinguished name within 16 bytes. If the length limit is exceeded, the value after that limit is truncated.
`pid`	The process-distinguished name (hexadecimal) is output. This value differs from the value that manages the OS. In the case of a log output by using `CJMessageFileHandler`, the hash value that is assigned to the `Runtime` instance by JavaVM is output. In the case of a log output by using `CJMPMessageFileHandler`, the lower level 32 bits, during the time (in milliseconds) at which Hitachi Trace Common Library is loaded by JavaVM, is output.
`tid`	The thread-distinguished name (hexadecimal) is output. This is a hash value assigned to the `Thread` instance by JavaVM. This value differs from the value that manages the OS.
`MsgID`	The message ID is output. You specify a message ID within 21 bytes. If the length limit exceeds, the value after that limit is truncated.
`Message text`	This is a message body. This is a string that does not contain the control characters such as `CR (0x0D)`, `LF (0x0A)`, `NULL (0x00)`, and `EOF (0x1A)`. You specify a length from 0 through 4,095 characters. If the length limit exceeds, the value after that limit is truncated. Note that if you include control characters, output contents are not guaranteed.
`CRLF`	The record terminal code (`0x0D` or `0x0A`) is output.

8.4 Methods used in the user log output

This section describes the methods of the `Logger` class which are used for the user log output and the package to which the `CJLogRecord` class belongs. For details on the list of methods of the `CJLogRecord` class and the functionality and grammar, see *7. APIs used in the user log functionality* in the *uCosminexus Application Server API Reference Guide*.

8.4.1 Methods of the Logger class used in the user log output

You use the following log method for receiving and passing `AppName` and `MsgID` by using the `CJLogRecord` method:

```
void log(LogRecord record)
```

8.4.2 Package to which the CJLogRecord class belongs

You must import the following package to use the `CJLogRecord` class with the source program:

```
com.hitachi.software.ejb.application.userlog
```

The package is stored in:

```
Cosminexus-installation-directory\CC\client\lib\HiEJBClientStatic.jar
```

For the implementation example of programs when using the user log functionality, see *8.5 Implementation for user log output*.

8.5 Implementation for user log output

You code using the Java logging APIs to output logs in J2EE applications or batch applications. If you want to output the names and message IDs of J2EE applications or batch applications to a user log, you use the `CJLogRecord` class provided with Cosminexus.

The `CJLogRecord` class is a class that inherits the `LogRecord` class of the Java logging APIs. You can create a `CJLogRecord` object with the specified message ID and application name. You can output any message ID and application name to a user log by specifying the object created in this class, in the argument of a log method, in the `Logger` class.

Example to output a user log having application name "UserApp" and message ID "USER10000-E":

```
try{
  //Execute process that outputs an error
}
catch(Error ex){
   logger.log(CJLogRecord.create(Level.SEVERE, "Catch an exception", "User
App", "USER10000-E"));
}
```

For details on APIs, see the *uCosminexus Application Server API Reference Guide*.

8.6 Creating and setting loggers and handlers

To output a user log by using the Java logging APIs, you create loggers and handlers, and specify the required settings. You specify the parameters such as the application-distinguished name (`AppName`) and the message ID (`MsgID`), required for the log output, in arguments of the `create` method in the `CJLogRecord` class provided with Cosminexus. You can also create your own class to customize the log filtering and format of output contents.

Note that for a user log output, you are required to specify the properties such as the output destination of logs and number of configuration files in the execution environment. For details on user log settings in the execution environment, see *8.8 Setting the user log output of J2EE applications* or *8.9 Setting the user log output of batch applications*.

This section gives an overview of creating and setting loggers and handlers used for the user log output of J2EE applications or batch applications.

For details on the user log output of EJB client applications, see *8.10 Setting the user log output of EJB client applications (When using the cjclstartap command)* or *8.11 Implementing and setting the user log output of EJB client applications (When using the vbj command)*.

8.6.1 Creating and setting loggers

You create a *logger* by specifying logger names. The contents that you specified while setting a system are used for creating a logger.

With each application, you acquire the logger created by specified logger name and output a log by using the acquired logger. Using the methods of the `Logger` class, you can create a logger and specify the log output. The specified log is converted to `LogRecord`, passed to the handler, and output to a log file or a console.

In addition, you can also specify filters, log levels, and handlers used in loggers, for choosing the log in logger, as and when required.

8.6.2 Creating and setting handlers

A *handler* is created and set up by using the contents that are specified when you set up a system.

When using `CJMessageFileHandler`, you can create multiple file handlers by changing the handler name.

You can specify the following items in the file filter created with `CJMessageFileHandler`:

- Log file settings such as the output destination file, number of files, and the size of user logs
- Log acquisition level
- Filters and formatters to be used

Note that when the application name and message ID of a log output by a handler can be the same, you can specify this value as a property of `CJMessageFileHandler`. If you want to change application names and message IDs of the logs output for messages, use and implement the `CJLogRecord` class such that you can output the application name and message ID for each log output processing in the application.

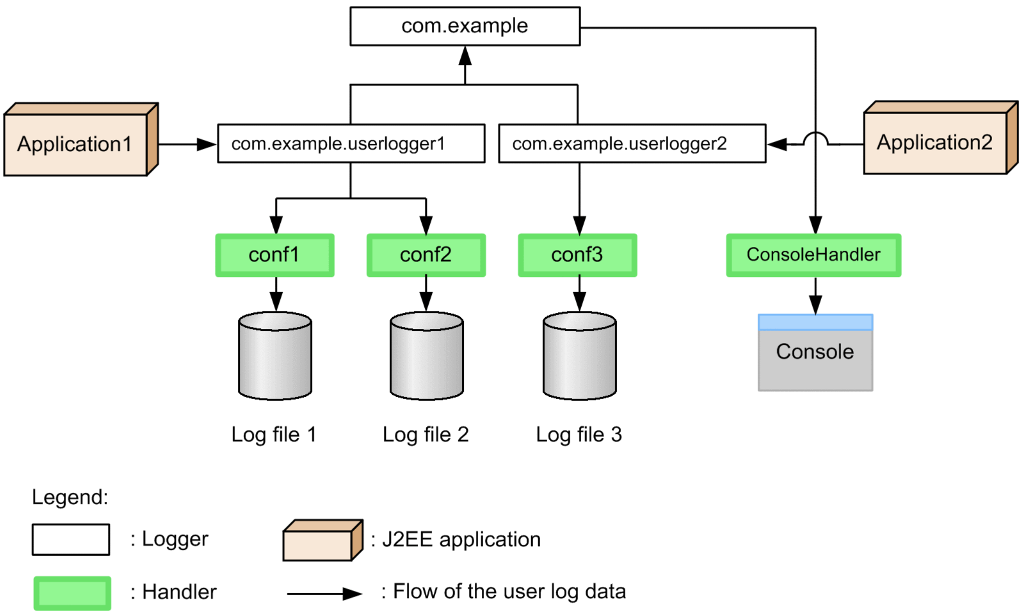
8.6.3 Guidelines for creating and setting loggers and handlers

The following are the guidelines for creating and setting loggers and handlers:

- You can connect multiple file handlers for one logger. However, you cannot connect to the file handlers, having the same output destination, from multiple loggers.
- If you want to change the output destination of log for each application, create a logger for each application.
- A logger can have hierarchical relationship. If a logger has hierarchical relationship, the log messages acquired by a lower level logger are propagated to an upper level logger. Stop the propagation of a logger as and when required. Particularly, a root logger exists by default at the top of the logger. In the case of the J2SE default settings, `ConsoleHandler` is connected to the root logger. If you do not stop the propagation to the upper level logger, all messages are output to the console from `ConsoleHandler` of the root logger.
- The handler outputs messages for each instance, and therefore if one output message is sent to multiple handlers, the output message is output multiple times. For example, the `ConsoleHandler` messages of two locations are output twice to the console.
- If you are using multiple log files in one application, create a handler for each output destination.

The following figure shows an example of creating loggers and handlers.

Figure 8–3: Example of creating loggers and handlers



In this example, two types of loggers (`com.example.userlogger1` and `com.example.userlogger2`) are created for J2EE applications 1 and 2. To output two types of log files from `com.example.userlogger1` depending on the output levels and the output contents of the log, two types of the `CJMessageFileHandler` handlers (`conf1` and `conf2`) are created. With this configuration, you can output important user logs on and above the `SEVERE` level to the log file 1, and all the user logs on and above the `INFO` level to the log file 2. On the other hand, only one type of log file is output from `com.example.userlogger2`. In such cases, of all the logs specified from a J2EE application, the user logs that are up to the level specified in the logger of `com.example.userlogger2` and the `conf3` handler are output to the log file 3. If you want to output the log to the console, use the `ConsoleHandler` handler of the standard J2SE.

Set up the size and number of files of log files appropriately depending on the user log quantity, which is output by applications, and the specified output level.

8.7 How you can use your own Filter/ formatter/ handler

This section describes the usage method for using the `Filter` class, `Formatter` class, or `Handler` class that you have created on your own, with the user log functionality. In this section, the class created by the user is called the *user-created class*.

You can perform log filtering or formatting of the output contents by creating a user-created class. You create a `Filter` class, `Formatter` class, or `Handler` class as a user-created class, include it in a library JAR or a container extension library, and use the class.

You can use a user-created class with the user log functionality by the following two methods:

- Using library JAR
  You can use this method for J2EE applications. You cannot use this method for batch applications.
- Using container extension library
  You can use this method in J2EE applications or batch applications.

The following subsections describe each method.

8.7.1 Using library JAR

In this method, you create a user-created class of the `Filter` class, `Formatter` class, or `Handler` class type in an application, add it to the logger, and use the class. In such cases, the following processes are executed:

- Firstly, instantiate a `Handler` class in the application.
- After that, connect the instantiated `Filter` class and `Formatter` class to the instance of the `Handler` class.
- Finally, add the instance of the connected `Handler` class to the logger.

You create the user-created class in this case according to the specification of J2SE `java.util.logging`. You can include the created class in the library JAR for WAR, EJB-JAR or import, in the same way as a normal user class, and then use the class.

The creation procedure for the user-created class, when using the class by including in the library JAR, is as follows:

1. Set up security policy in security policy file (`server.policy`).
   For details on the security policy settings, see *8.8.2 Setting security policy*.
2. Create library JAR for import containing your own `Handler` class, `Filter` class, and `Formatter` class.
3. Specify to import the class of the created library JAR, by using the server management commands.
4. Create instances of your own class in the source program of the application.
5. Implement the processing of connecting to the `Logger` class and `Handler` class.
   Note the following points when performing implementation by using the log manager (`LogManager`) of the J2SE1.4 specifications.
   - You cannot customize the log manager by using properties (such as `java.util.logging.class` and `java.util.logging.file`). If you customize the log manager, creating a user log might fail.

- You cannot invoke the `readConfiguration(InputStreamins)` method of the log manager in a source program. If you invoke the `readConfiguration(InputStreamins)` method and initialize the configuration of the `Logger` class, the log system built using the user log functionality fails.

For details on the coding, see *8.12 Notes for using the user log functionality*.

8.7.2 Using container extension library

In this method, you specify the class name of the user-created classes of the `Filter` class, `Formatter` class, or `Handler` class type in the property key of the user log functionality, build a log configuration containing the user-created class, when starting a J2EE server, and then use the class. This method differs from the J2EE standard method.

You specify a JAR file, containing the user-created class, as a container extension library and specify the class path to the created library. As a result, when you start the J2EE server, the `CJMessageFileHandler` class specified in the property key, formatter and filter are created and executed, enabling you to build a log configuration.

The procedure is as follows:

1. Create a JAR file (container extension library JAR) containing the user-created classes of the `Formatter` class, `Filter` class and `Handler` class type.
   Here, the file name is set as `myloglib.jar`.

2. Place `myloglib.jar` at any location.
   Here, the description is given based on the prerequisite that `myloglib.jar` is placed at the following location:
   - In Windows
     `c:\mylib`
   - In UNIX
     `/usr/mylib`

3. Specify the class path to the placed library.
   For example, in the case of a J2EE server, you specify the following settings in `usrconf.cfg` (option definition file):
   - In Windows
     `add.class.path=C:\mylib\myloglib.jar`
   - In UNIX
     `add.class.path=/usr/mylib/myloglib.jar`

4. In the property key for the user log functionality of `usrconf.properties` (user property file), you specify the full class name containing the package name.

8.8 Setting the user log output of J2EE applications

This section describes the setting methods to output logs, output by J2EE applications, in the Hitachi Trace Common Library format. This section also describes an example of a user log output for applications. If you do not want to output the logs of J2EE applications, these settings are not required.

You must perform the following settings to output logs in the Hitachi Trace Common Library format:

- J2EE server settings
- Security policy settings

8.8.1 J2EE server settings

Edit an Easy Setup definition file and specify the log output destination, log levels, number of log files, filters and formatters to be used, from the handler.

(1) Setting contents

In the Easy Setup definition file, you specify the settings in the `<configuration>` *tag* of a logical J2EE server (`j2ee-server`), to output the user logs of a J2EE application, with the parameters starting with `ejbserver.application`. The parameters starting with `ejbserver.application` are given below. In *handler-name*, specify the handler name used to distinguish key values. In *logger-name*, specify the logger name, which is specified when acquiring `Logger` instances.

- `ejbserver.application.userlog.CJLogHandler.`*handler-name*`.appname`
  For every handler, specify a default value for the J2EE application name (value of the `AppName` field) in the message output to a log file.
- `ejbserver.application.userlog.CJLogHandler.`*handler-name*`.count`
  For every handler, you specify number of log files.
- `ejbserver.application.userlog.CJLogHandler.`*handler-name*`.encoding`
  For every handler, you specify the encoding of the character string output to a log file.
- `ejbserver.application.userlog.CJLogHandler.`*handler-name*`.filter`
  For every handler, you specify the filter name to be used.
- `ejbserver.application.userlog.CJLogHandler.`*handler-name*`.formatter`
  For every handler, you specify the formatter name to be used.
- `ejbserver.application.userlog.CJLogHandler.`*handler-name*`.level`
  For every handler, you specify an upper limit of the log acquisition level.
- `ejbserver.application.userlog.CJLogHandler.handler-name.limit`
  For every handler, you specify the size of the log file.
- `ejbserver.application.userlog.CJLogHandler.`*handler-name*`.msgid`
  For every handler, specify the default value of a message ID (value of the `MsgID` field) of the message output in a log file.
- `ejbserver.application.userlog.CJLogHandler.`*handler-name*`.path`
  For every handler, you specify the output destination and the prefix of a log file. The output log file name is *Prefix 1-through-16-number*`.log`. Specify this key without fail.

- ejbserver.application.userlog.CJLogHandler.*handler-name*.separator

  For every handler, you specify the default value of the element separating character used to output the log file messages in one sentence.
- ejbserver.application.userlog.loggers

  Declares the logger name to be used. Specify this key without fail.
- ejbserver.application.userlog.Logger.*logger-name*.handlers

  For every logger, you specify the handler name to be used. Specify this key without fail.
- ejbserver.application.userlog.Logger.*logger-name*.level

  For every logger, you specify the log acquisition level of the logger.
- ejbserver.application.userlog.Logger.*logger-name*.useParentHandlers

  For every logger, you specify whether the log record of the level, which passes through the logger, is to be propagated to the handler used by the parent logger.
- ejbserver.application.userlog.Logger.*logger-name*.filter

  For every logger, you specify the filter used for choosing a message in the logger.

You must specify at least the following three parameters to output the user logs of J2EE applications:

- ejbserver.application.userlog.CJLogHandler.*handler-name*.path
- ejbserver.application.userlog.loggers
- ejbserver.application.userlog.Logger.*logger-name*.handlers

For details on the Easy Setup definition file, see *4.3 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

(2) Notes

- You can connect multiple handlers to a logger. However, you cannot connect a file handler (CJMessageFileHandler) with the same Path settings to multiple loggers. A file handler performs instantiation by referring to the specification of connection to a logger (value of ejbserver.application.userlog.Logger.<logger name>.handlers). In such cases, if a handler having the same prefix (value of ejbserver.application.userlog.CJLogHandler.*handler-name*.path) as the log output destination is instantiated, opening of the log file fails.
- You can specify the settings and setup of the handler and the logger in the Easy Setup definition file. However, when you directly create a handler or change the configuration of a logger in a J2EE application, you must have the security permission of LoggingPermission("control") for the application to be executed. For details on how to set up the security permissions of LoggingPermission("control"), see *8.8.2 Setting security policy*.

8.8.2 Setting security policy

This section describes the setting of the security policy.

You must set up the security policy when changing the configuration of the Logger class of the J2SE1.4 specifications, creating a FileHandler class, and directly implementing the logging functionality of the standard J2SE in a source program of the application. You define the security polity in server.policy (security policy file for J2EE servers) or web.policy (SecurityManager definition file).

Note that when defining the security policy in `server.policy`, specify the settings by using the Smart Composer functionality command, after building the system.

You need not set up the security policy when specifying the output for a logger, which is built on the basis of the parameters of the Easy Setup definition file. You must set up the security policy in the following cases:

- When creating a J2SE standard file handler in the source code of a user application
- When changing the configuration of a logger by invoking the `addHandler` method in the `Logger` class

In such cases, the security policy used for the Java logging API operations is required. Specify the following security permissions as and when required.

The setting contents of `server.policy` are given below.

(1) When creating filters and formatters with reflection

You add the following line when creating the `Filter` class or the `Formatter` class with reflection:

```
permission java.lang.reflect.ReflectPermission "suppressAccessChecks";
```

All the `Handler` classes acquire the properties from the log manager (`LogManager`) and generate the `Formatter` class or `Filter` class by using the Reflection functionality at the time of execution. Therefore, you must have permissions related to Reflection.

(2) When setting properties of log manager (LogManager)

You add the following line when setting the properties of a log manager:

```
permission java.util.PropertyPermission "*", "read, write";
```

A log manager must have reading and writing permissions (`set**` of `Property`) for the property values used for log output.

(3) When using J2SE standard file handler (When using the classes (FileHandler and CJMessageFileHandler) that output File)

You add the following line when using the classes (`FileHandler` and `CJMessageFileHandler`) that output File:

```
permission java.io.FilePermission "<<ALL FILES>>", "read, write";
```

You must have permissions to actually output the log to a file. You must have reading and writing permissions when you want to output the log to a file.

(4) When changing a log system by using the Logger.addHandler method of the Java logging API

You add the following line when using the logging API of the J2SE1.4 specification:

```
permission java.util.logging.LoggingPermission "control";
```

You must specify the security permissions for using the Java logging API. You cannot use logging APIs, if this value is not specified.

(5) Setting example

The following is an example of setting `server.policy` (security policy file for a J2EE server), when changing the log system by using the `Logger.addHandler` method of the Java logging APIs, from Servlets of J2EE applications.

Setting example

```
//
// Grant permissions to JSP/Servlet
//
grant codeBase "file:${ejbserver.http.root}/web/${ejbserver.serverName}/-" {
   permission java.lang.RuntimePermission "loadLibrary.*";
   permission java.lang.RuntimePermission "queuePrintJob";
   permission java.net.SocketPermission "*", "connect";
   permission java.io.FilePermission "<<ALL FILES>>", "read, write";
   permission java.util.PropertyPermission "*", "read";
   permission javax.security.auth.AuthPermission "getSubject";
   permission javax.security.auth.AuthPermission "createLoginContext.*";

   //For J2SE Logging Source
   permission java.lang.reflect.ReflectPermission "suppressAccessChecks";
   permission java.util.PropertyPermission "*", "read, write";
   permission java.util.logging.LoggingPermission "control";

};
```

For details on how to define `server.policy` (security policy file for J2EE servers), see *2.2.4 server.policy (Security policy file for J2EE servers)* in the *uCosminexus Application Server Definition Reference Guide*.

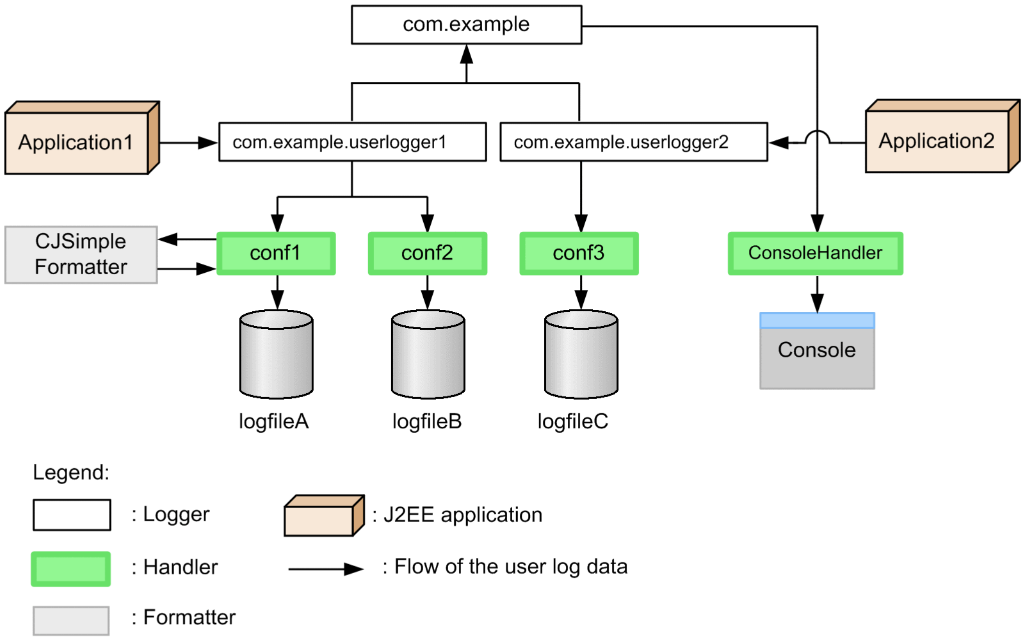
8.8.3 Examples of the user log output of applications

This section describes the settings used to output the user log of J2EE applications, with specific examples.

(1) Example used for user log output

This subsection describes the settings used to output the user log of J2EE applications, with the following example. The following figure gives an overview of the used example.

Figure 8–4: Example of user log output of J2EE applications



In company A, the operation history of J2EE applications is output to the log file as a business history, by using the logger functionality. Among the J2EE applications of Company A, the J2EE applications, for which operation history is to be output, are of two types; *Application1* and *Application2*. For each J2EE application, logs of different message levels are output to different files. The directories of J2EE application names are created and stored in respective log files, in the above figure.

(a) Features of "Application1"

`com.example.userlogger1` is the logger name of *Application1*.

Application1 is a complex and large J2EE application. If a critical error of the `SEVERE` level occurs, a message containing the trace information of Java Exception is retained in `logfileA`, to quickly identify the cause. The messages of the `INFO` level and below are output to `logfileB` as the trace log of operations. To output two types of log files from `com.example.userlogger1`, depending on the output level and the output contents of the log, two types of `CJMessageFileHandler` handlers (`conf1` and `conf2`) are created.

Details of "logfileA"

- To acquire the trace information, you use "`CJSimpleFormatter`" as an output formatter to `logfileA`.
- Only `SEVERE` level messages are output to `logfileA`, so a log file of a very large size is not required. However, as the trace information is output, a size of approximately 40 megabytes is required to accumulate 10,000 records, with a maximum size of records per message as 4,096 bytes. Therefore, set the file size to 10 megabytes and number of files to 4.
- Set the name of the J2EE application to "`my_app1`" for distinguishing the messages output by `Application1`.

Details of "logfileB"

- All the messages of the `INFO` level and below are output, so `logfileB` requires a larger file size. The log disc capacity, calculated from the amount of messages per day and the retention period, is approximately 256 megabytes. The maximum number of files is 16, therefore set the file size to 16 megabytes and number of files to 16.

- Set the name of the J2EE application to "my_app1" for distinguishing the messages output by Application1.

(b) Features of "Application2"

com.example.userlogger2 is the logger name of Application2.

Application2 is built with high quality log messages and is a small J2EE application. Only the necessary minimum messages are output to logs, so the messages of WARNING and above level are retained in "logfileC". One log file is output from com.example.userlogger2, so the CJMessageFileHandler handler, which is called conf3, is created.

Details of "logfileC"

- Only the WARNING level messages are output. The maximum length per message is approximately 200 bytes, approximately 2 megabytes size is required for accumulating 10,000 records. As a result, you set the file size to 1 megabyte and number of files to 2.
- You set the name of the J2EE application to "my_app2" for distinguishing the messages output by Application2.

(c) Settings for debugging

You also specify the settings for debugging during development. For displaying all the message contents that are sent to "com.example.userlogger1" and "com.example.userlogger2", you connect "ConsoleHandler" of the java.util.logging to "com.example" logger. Because all the message contents propagated from a child logger are to be displayed in this logger, you set the log acquisition level of the logger and the handler to ALL.

(2) Example of setting the user log output

The following example shows the settings for the user log output, with the example shown in *(1) Example used for user log output*.

(a) Example of setting the Easy Setup definition file

The following example describes the settings of the Easy Setup definition file (when defining physical tier):

```
<configuration>
  <logical-server-type>j2ee-server</logical-server-type>
<!-- Perform settings so that you cannot propagate the log record passed -->
<!-- to logger, to handler used by parent logger (because root logger -->
<!-- exists by default). -->
  <param>
    <param-name>ejbserver.application.userlog.Logger.com.example.useParentHa
ndlers</param-name>
    <param-value>false</param-value>
  </param>
<!-- Specify J2EE application name, output destination, size, -->
<!-- number of files, log acquisition level -->
<!-- and formatter name to be used of "logfileA". -->
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf1.appname</pa
ram-name>
    <param-value>my_app1</param-value>
  </param>
  <param>
```

```
    <param-name>ejbserver.application.userlog.CJLogHandler.conf1.path</param
-name>
    <param-value>application1/logfileA</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf1.limit</para
m-name>
    <param-value>10485760</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf1.count</para
m-name>
    <param-value>4</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf1.level</para
m-name>
    <param-value>SEVERE</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf1.formatter</
param-name>
    <param-value>com.hitachi.software.ejb.application.userlog.CJSimpleFormat
ter</param-value>
  </param>

<!-- Specify J2EE application name, output destination, size, -->
<!-- number of files, log acquisition level of "logfileB" -->
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf2.appname</pa
ram-name>
    <param-value>my_app1</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf2.path</param
-name>
    <param-value>application1/logfileB</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf2.limit</para
m-name>
    <param-value>16777216</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf2.count</para
m-name>
    <param-value>16</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf2.level</para
m-name>
    <param-value>INFO</param-value>
  </param>

<!-- By using settings of "conf1" and "conf2" handler names, used by-->
<!-- "com.example.userlogger1",-->
<!-- initialize and connect the file handler (CJMessageFileHandler). -->
<!-- Here, logger and handler is created. -->
```

```
  <param>
    <param-name>ejbserver.application.userlog.Logger.com.example.userlogger1
.handlers</param-name>
    <param-value>com.hitachi.software.ejb.application.userlog.CJMessageFileH
andler;conf1,
                  com.hitachi.software.ejb.application.userlog.CJMessageFile
Handler;conf2</param-value>
  </param>

<!-- Specify log acquisition level of "com.example.userlogger1". -->
<!-- Match it with higher level of "conf1" and "conf2", and set to "INFO"
. -->
  <param>
    <param-name>ejbserver.application.userlog.Logger.com.example.userlogger1
.level</param-name>
    <param-value>INFO</param-value>
  </param>

<!-- Specify output destination and log acquisition level of "logfileC". -->
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf3.appname</pa
ram-name>
    <param-value>my_app2</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf3.path</param
-name>
    <param-value>application2/logfileC</param-value>
  </param>
  <param>
    <param-name>ejbserver.application.userlog.CJLogHandler.conf3.level</para
m-name>
    <param-value>WARNING</param-value>
  </param>

<!-- By using settings of "conf3" handler name, used by -->
<!-- "com.example.userlogger2",-->
<!-- initialize and connect the file handler (CJMessageFileHandler). -->
<!-- Here, logger and handler are created. -->
  <param>
    <param-name>ejbserver.application.userlog.Logger.com.example.userlogger2
.handlers</param-name>
    <param-value>com.hitachi.software.ejb.application.userlog.CJMessageFileH
andler;conf3</param-value>
  </param>

<!-- Specify log acquisition level of "com.example.userlogger2". -->
  <param>
    <param-name>ejbserver.application.userlog.Logger.com.example.userlogger2
.level</param-name>
    <param-value>WARNING</param-value>
  </param>

<!-- Perform settings for -->
<!-- debugging************************************************-->
<!-- Specify log acquisition level of "ConsoleHandler". -->
  <param>
    <param-name>java.util.logging.ConsoleHandler.level</param-name>
```

```
    <param-value>INFO</param-value>
  </param>

<!-- Specify "ConsoleHandler" handler name to be used in "com.example" logge
r,-->
<!-- and connect to handler. Here, logger and handler are created. -->
  <param>
    <param-name>ejbserver.application.userlog.Logger.com.example.handlers</p
aram-name>
    <param-value>java.util.logging.ConsoleHandler</param-value>
  </param>

<!-- Specify log acquisition level of "com.example" logger. -->
  <param>
    <param-name>ejbserver.application.userlog.Logger.com.example.level</para
m-name>
    <param-value>ALL</param-value>
  </param>

<!-- If debugging is not required, cancel the setting of propagation to -->
<!--parent logger. -->
<!--
  <param>
    <param-name>ejbserver.application.userlog.Logger.com.example.userlogger1
.useParentHandlers</param-name>
    <param-value>false</param-value>
  </param>
-->
<!--
  <param>
    <param-name>ejbserver.application.userlog.Logger.com.example.userlogger2
.useParentHandlers</param-name>
    <param-value>false</param-value>
  </param>
-->
<!-- If debugging is not required, cancel the creation of "com.example". -->
<!--
  <param>
    <param-name>ejbserver.application.userlog.loggers</param-name>
    <param-value>com.example.userlogger1, com.example.userlogger2</param-val
ue>
  </param>
-->
<!-- *********************************************************************
**-->

<!-- Declare the usage of logger. -->
  <param>
    <param-name>ejbserver.application.userlog.loggers</param-name>
    <param-value>com.example,com.example.userlogger1,com.example.userlogger2
</param-value>
  </param>
</configuration>
```

(b) Example of setting Application1

The following example describes the source code of `Application1`:

```
import java.util.logging.*;
import com.hitachi.software.ejb.application.userlog.*;

public class application1{

    static Logger logger = Logger.getLogger("com.example.userlogger1");

    public static void exec(){

        logger.log(
                   CJLogRecord.create(Level.INFO,
                   "application1 start.","AP1_10000-I"));

        try{

            throw new Exception("Exception1!");

        }
        catch(Exception ex){

            logger.log(
                       CJLogRecord.create(Level.SEVERE,
                       "Catch an exception!", ex, "AP1_10100-E"));

        }

        logger.log(
                   CJLogRecord.create(Level.INFO,
                   "application1 end.","AP1_10001-I"));

    }

}
```

The following example describes the output of `application1/logfileA1.log`:

```
     yyyy/mm/dd hh:mm:ss.sss           pid      tid      message-id  messag
e(LANG=ja)
0047 2003/12/06 19:51:32.265  my_app1   00EB7859 012A54F9 AP1_10100-E 2003/1
2/06
19:51:32|application1|exec|Fatal|Catch an exception!|java.lang.Exception:
Exception1!|application1.exec(application1.java.18)|application1.main(applic
ation1.java.64)
```

The following example describes of the output of `application1/logfileB1.log`:

```
     yyyy/mm/dd hh:mm:ss.sss          pid      tid      message-id   messag
e(LANG=ja)
0046 2003/12/06 19:51:32.250  my_app1  00EB7859 012A54F9 AP1_10000-I  applic
ation1 start.
0048 2003/12/06 19:51:32.265  my_app1  00EB7859 012A54F9 AP1_10100-E  Catch
an exception!
0049 2003/12/06 19:51:32.265  my_app1  00EB7859 012A54F9 AP1_10001-I  applic
ation1 end.
```

The following example describes the output to the console screen:

```
Information: application1 start.
2003/12/06 19:51:32 application1 exec
Fatal: Catch an exception!
java.lang.Exception: Exception1!
        at application1.exec(application1.java:18)
        at application1.main(application1.java:64)
2003/12/06 19:51:32 application1 exec
Information: application1 end.
```

(c) Example of setting Application2

The following example describes the source code of `Application2`:

```
import java.util.logging.*;
import com.hitachi.software.ejb.application.userlog.*;

public class application2{

    static Logger logger = Logger.getLogger("com.example.userlogger2");

    public static void exec(){

        logger.log(
                   CJLogRecord.create(Level.INFO,
                   "application2 start.","AP2_20000-I"));

        try{

            throw new Exception("Exception2!");

        }
        catch(Exception ex){

            logger.log(
                        CJLogRecord.create(Level.SEVERE,
                       "Catch an exception!", ex, "AP2_20100-E"));

        }

        logger.log(
                   CJLogRecord.create(Level.INFO,
                   "application2 end.","AP2_20001-I"));

    }

}
```

The following example describes the output of `application2/logfileC1.log`:

```
     yyyy/mm/dd hh:mm:ss.sss           pid      tid      message-id   messag
e(LANG=ja)
0048 2003/12/06 19:51:32.265  my_app2  00EB7859 012A54F9 AP2_20100-E  Catch
an exception!
```

(d) Example of setting Application3

This subsection describes an example of the log output of the J2EE application *Application3*, to the same log file as of *Application1*. In such cases, `Application3` must acquire the logger by using the same logger name in the same process (thread may be different) as `Application1`.

The following example describes the source code of `Application3`:

```
import java.util.logging.*;
import com.hitachi.software.ejb.application.userlog.*;

public class application1{

    static Logger logger = Logger.getLogger("com.example.userlogger1");

    public static void exec(){

        logger.log(,
                   CJLogRecord.create(Level.INFO,
                   "application3 start.","my_app3","AP3_30000-I"));

        try{

            throw new Exception("Exception2!");

        }
        catch(Exception ex){

            logger.log(,
                       CJLogRecord.create(Level.SEVERE,
                       "Catch an exception!", ex, "my_app3","AP3_30100-E"));

        }

        logger.log(
                   CJLogRecord.create(Level.INFO,
                   "application3 end.","my_app3","AP3_30001-I"));

    }

}
```

The following example describes the output of `application1/logfileB1.log`:

```
     yyyy/mm/dd hh:mm:ss.sss           pid      tid      message-id   messag
e(LANG=ja)
0046 2003/12/06 19:51:32.250  my_app1  00EB7859 012A54F9 AP1_10000-I  applic
ation1 start.
0093 2003/12/06 19:51:32.265  my_app3  00EB7859 010CB027 AP3_30000-I  applic
ation3 start.
0095 2003/12/06 19:51:32.265  my_app1  00EB7859 012A54F9 AP1_10100-E  Catch
an exception!
```

8.9 Setting the user log output of batch applications

You specify the same settings for the user log output of batch applications as for the user log output of J2EE applications. For details on the settings, see *8.8 Setting the user log output of J2EE applications*.

However, in the case of batch applications, the security policy settings are not required.

8.10 Setting the user log output of EJB client applications (When using the cjclstartap command)

This section describes the settings used to output the user logs of EJB client applications.

The method of setting user logs of EJB client applications varies with the commands used to start the EJB client applications. This section describes the settings, when starting an application using the `cjclstartap` command.

When using the `cjclstartap` command, you set up a user log in the property file (`usrconf.properties`) of an EJB client application. In the keys starting with `ejbserver.application`, specify the output destination file of user logs, log levels, number of log files, and filters and formatters to be used. For details on the keys that you can specify, see *3.2.2 usrconf.properties (User property file for batch servers)* in the *uCosminexus Application Server Definition Reference Guide*.

Also, you specify a JAR file for the class path in the option definition file (`usrconf.cfg`) of the EJB client application. For details on setting the JAR file for a class path, see *3.7.4 Setting JAR files for class path of EJB client applications* in the *uCosminexus Application Server EJB Container Functionality Guide*.

8.11 Implementing and setting the user log output of EJB client applications (When using the vbj command)

This section describes the settings that you need to do when starting the EJB client applications by using the `vbj` command. When starting the application by using the `vbj` command, you must perform implementation for using the user log functionality.

This section describes the preparations for using the user log functionality in EJB client applications and the flow of processing until the user log is output.

8.11.1 Overview of processing when using the vbj command

`CJMPMessageFileHandler` is provided as a file handler of the user logs of EJB client applications. When using the `vbj` command, you specify an output destination file for the `CJMPMessageFileHandler` log, log levels, number of log files, and filters and formatters to be used, in the configuration file for user logs of EJB client applications.

When implementing the user log functionality, you specify an output destination file of the `CJMPMessageFileHandler` log, log levels, number of log files and filters and formatters to be used, in the configuration file for the user logs of EJB client applications. You must perform coding in such a way that the configuration file for a user log is read in a user application program.

When executing a command to start an EJB client application, the configuration file is read from the user application program, and is specified in the system properties of the EJB client application.

8.11.2 Preparing for use

The following preparations are needed when you use the user log functionality in an EJB client application.

Note that you must perform the settings on the Application Server machine as a prerequisite of using the user log functionality in EJB client applications.

- **Preparing a configuration file for the user log functionality**
  You prepare a configuration file for the user log functionality, used for setting up system properties. You code the configuration file in the J2SE property file format. Specifying a file name and storage directory is optional.
  In the configuration file, you can specify the keys starting with `ejbserver.application.userlog` from among the keys that can be specified in `usrconf.properties` used for J2EE servers. For details on the keys that you can specify, see *2.2.3 usrconf.properties (User property file for J2EE servers)* in the *uCosminexus Application Server Definition Reference Guide*.
- **Implementing the processing of setting system properties**
  You must add the processing for reading the configuration file and for setting system properties, in the source code of EJB client applications. This processing needs to be executed before executing the initialization of the EJB client function.
- **Adding class path of JAR file**
  Add a class path of the JAR file corresponding to the handler to be used, in the class path when starting an EJB client application. For details on specifying the class path, see *3.7.4 Setting a JAR file to the class path of EJB client applications* in the *uCosminexus Application Server EJB Container Functionality Guide*.

> **Reference note**
>
> When using the user log functionality in EJB client applications, you need not set the security policy.

8.11.3 Procedure for the user log output processing

The output of the user logs on EJB client applications is performed with the following procedure:

1. **Setting system properties**
   The system properties are set up by using a configuration file.
2. **Initializing EJB client**
   The log system is set up by invoking the method that initializes the EJB client functionality.
3. **Executing Java logging API**
   The user log is output by executing the Java logging APIs.

This section describes the contents of each processing along with the procedure.

(1) Setting system properties

The system properties for the user log functionality of EJB client applications are set by using a configuration file.

The properties that can be set in the system properties are the keys starting with `ejbserver.application.userlog`, from among the properties that can be specified in `usrconf.properties` for J2EE servers. An example of a setting is as follows:

```
# user-log handler function
ejbserver.application.userlog.CJLogHandler.conf1.appname=my_app1
ejbserver.application.userlog.CJLogHandler.conf1.path=application1/logfileA
ejbserver.application.userlog.CJLogHandler.conf1.limit=10485760
ejbserver.application.userlog.CJLogHandler.conf1.count=2
ejbserver.application.userlog.CJLogHandler.conf1.level=SEVERE

# user-log logger function
ejbserver.application.userlog.Logger.com.example.userlogger1.handlers=com.hi
tachi.software.ejb.application.userlog.CJMPMessageFileHandler;conf1
ejbserver.application.userlog.Logger.com.example.userlogger1.useParentHandle
rs=true
ejbserver.application.userlog.Logger.com.example.userlogger1.level=INFO
ejbserver.application.userlog.loggers=com.example.userlogger1
```

In an EJB client application, you can specify `CJMPMessageFileHandler` or `CJMessageFileHandler` as a handler used for the user log output. You specify the handler to be used, in the `ejbserver.application.userlog.Logger.`*logger-name*`.handlers` key. In the example, the `CJMPMessageFileHandler` class is specified in the `userlogger1` logger.

`CJMPMessageFileHandler` is a handler, which contains the functionality that enables the concurrent output of logs from multiple processes to the same file. This enables you to collect and output the user logs output by multiple processes of an EJB client application. You can use this handler only for the EJB client application.

If you do not want to concurrently output logs from multiple processes to the same file, you can also use same `CJMessageFileHandler` as in the case of the user log output of J2EE applications. If you use `CJMessageFileHandler`, the log output performance is higher as compared to the case where `CJMPMessageFileHandler` is used.

> **Important note**
>
> If you use the `CJMPMessageFileHandler` class, Hitachi Trace Common Library creates a file used for the log management in *user-log-output-directory*`/mmap/`*prefix-of-log-file-name*`.mm`. Do not change or delete this file when you are using this user log output directory.

(2) Initializing EJB client functionality

A log system is set up by invoking the method that initializes the EJB client functionality. The EJB client functionality is initialized at one of the following timings:

- Generating initial context of JNDI (`new InitialContext` method)
- Login by the security functionality API (`login` method of `LoginInfoManager` class)
- Acquiring objects for setting a communication timeout, in the communication timeout functionality API (`getRequestTimeoutConfig` method of `RequestTimeoutConfigFactory` class)

> **Important note**
>
> If initialization fails, you cannot use the functionality used to output the user logs of EJB client applications. However, in the source code of a user application, you can specify and set up the J2SE standard `Handler` class and `Logger` class, or `Handler` class and `Logger` class which you created on your own, and then output the log.

(3) Executing Java logging API

In the processing in application, the Java logging API is executed and a user log is output. When using `CJMPMessageFileHandler`, note the following points:

Points to be considered when using CJMPMessageFileHandler

When using `CJMPMessageFileHandler`, delay might occur until details are actually applied in the file, because a memory mapped file is used. The details are applied in the file when the process ends. However, we recommend the execution of `flush` if the operation continues for a long time or if delay in reflecting the contents in the file is going to cause a problem.

You use the following two methods to execute `flush`:

- Invoke the `flush` method for all Handlers returned by the `Logger.getHandlers` method.
- Specify the `ejbserver.application.userlog.CJLogHandler.`*handler-name*`.autoFlush.enabled` property.

If you specify the `ejbserver.application.userlog.CJLogHandler.`*handler-name*`.autoFlush.enabled` property, `flush` is automatically executed. Therefore, do not use the `flush` method in such cases.

8.11.4 Extending the user log output in EJB client applications

To use your own created class (`Formatter`, `Filter`, and `Handler`), with the user log functionality of EJB client applications, you specify your own class in the class path, when starting JavaVM of EJB client applications.

8.11.5 How to use Filter/ Formatter/ Handler independently created by the user

To use your own created `Filter` class, `Formatter` class, or `Handler` class, with the user log functionality of EJB client applications, you specify the user-created class in the class path, when starting JavaVM of EJB client applications.

8.12 Notes for using the user log functionality

This section describes the notes for using the user log functionality.

8.12.1 Customizing LogManager

You can customize `LogManager` of the J2SE standard by using the properties such as `java.util.logging.config.class`. However, do not perform customization when using the user log functionality provided by Application Server. In the setup of a log system, which uses properties used in the user log functionality, the user log functionality acquires the log configuration from the properties by using `LogManager`, when starting a J2EE server. As a result, if you customize `LogManager`, an attempt to set up a log configuration might fail.

If you execute the `readConfiguration(InputStream ins)` method of `LogManager` and initialize the log configuration in the source code of application, the log configuration set up using the user log functionality fails. Therefore, do not execute this method.

However, you can use the user log functionality even after customization, if the structure is such that the customized `LogManager` completely inherits the already specified log configuration (contents of `LogManager`) and includes the added independent processing.

8.12.2 Notes for using your own filters and formatters

When a log message is output, the `isLoggable` method#1 of handler returns `true`#2, if the filter created by you, which is connected to the handler, throws an exception.

If the formatter created by you, which is connected to the handler, throws an exception, the handler cannot output a message that is formatted by the formatter. The message specified by you is output without any formatting by the formatter.

For details on the contents of the exceptions thrown by the filter and the formatter created by you, see `cjexception.log`.

#1

The `isLoggable` method determines the selection of a log message.

#2

`true` indicates that the message is targeted for output.

8.12.3 Connection between logger and handler

You can connect multiple handlers to a logger. However, you cannot connect a handler (`CJMessageFileHandler` or `CJMPMessageFileHandler`) with the same settings to multiple loggers.

8.12.4 Setting log output mode of EJB client applications

Two types of modes are used to output logs of EJB client applications. The operation mode that creates a subdirectory of a log output destination for processes is called the *subdirectory exclusive* mode. The operation mode that shares a subdirectory of log output destination with multiple processes is called the *subdirectory common* mode.

Because the subdirectory exclusive mode is used for compatibility with the versions earlier than 06-50, we recommend you to use the subdirectory common mode for creating an EJB client application.

Use the subdirectory common mode, if you want to use the user log functionality of EJB client applications or if you want to execute EJB client applications with the `cjclstartap` command.

For details on how to output logs of EJB applications, see *3.8 System log output of EJB client applications* in the *uCosminexus Application Server EJB Container Functionality Guide*. For details on the subdirectory to which logs of EJB applications are output, see *Chapter 7* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

8.12.5 Regarding the key in which the suffix of usrconf.properties ends with .level

In `usrconf.properties` of the J2EE server, among the keys in which the suffix ends with `.level`, the following phenomena occur if you set a key with a value other than `SEVERE`, `WARNING`, `INFO`, `CONFIG`, `FINE,` or `FINEST` in the value range.

1. When you start the server, the `java.util.logging.LogManager` class checks the values of the keys, in which the suffix ends with `.level`, when reading the keys. If a value is out of range, `java.util.logging.LogManager` class outputs an error message to the standard error output.

   (Example) If you specify, `sample.level=Error`

   `Bad level value for property : sample.level` is output

2. If the value of the user log functionality key, in which the suffix ends with `.level` is not appropriate, an error message is output in the same way as in case 1.

However, in both the cases, only a message is displayed and it does not affect the operation.

9 CORBA/OTM Gateway Functionality

This chapter describes the CORBA/OTM gateway functionality.

9.1 Organization of this chapter

This section describes the CORBA/OTM gateway functionality, which enables direct invocation of EJB applications from TPBroker OTM clients (OTM clients) and ORB clients.

The following table describes the organization of this chapter.

Table 9–1: Organization of this chapter (CORBA/OTM gateway functionality)

Category	Title	Reference location
Description	Overview of the CORBA/OTM gateway functionality	*9.2*
Implementation	Implementation procedures for EJB invocation from OTM applications	*9.3*
	Implementation procedures for EJB invocation from ORB clients	*9.4*
Description	Troubleshooting when a CORBA system exception occurs	*9.5*

9.2 Overview of the CORBA/OTM gateway functionality

Cosminexus Component Transaction Monitor provides the gateway functionality that enables direct invocation of EJB applications from OTM clients and ORB clients. The term *ORB clients* generically refer to the following three types of clients:

- TPBroker clients
- CORBA clients compliant with CORBA 2.1 other than TPBroker clients
- CORBA clients compatible with TMS-4V/SP/Object Access

9.2.1 Clients

The following table lists the types of clients that can directly invoke EJB applications.

Table 9–2: Clients supported by the gateway functionality

Client	Java	C++	COBOL
OTM	Y#1	Y	N
TPBrokerV3	Y	Y	--
TPBrokerV5	Y	Y	N
CORBA clients compliant with CORBA 2.1	Y#2	Y#2	Y#2
TMS-4V/SP/Object Access	--	--	Y

Legend:

Y: Can be connected

N: Cannot be connected

--: Not supported

#1

EJB applications cannot be invoked from TPBroker included in OTM V3 or Cosminexus, or from Java applications that use TPBroker V5.

#2

Theoretically, clients can be connected if they are compliant with the CORBA 2.1 specifications, but Cosminexus Component Transaction Monitor does not guarantee connections of these clients. Verification of connectivity is required before use.

9.2.2 Data types

The gateway functionality can send and receive data types that can be specified in IDL.

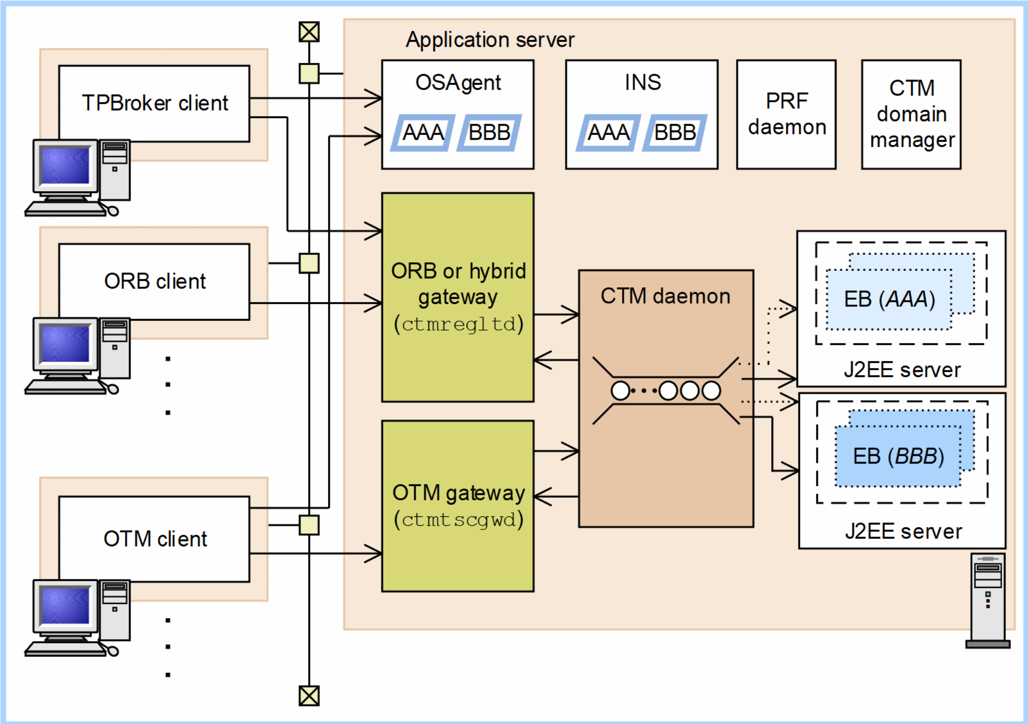
The `in` attribute is the only argument that can be specified in the EJB remote interface. You can use only Java arrays and classes for which the `Serializable` interface is implemented as structure types that can be used for arguments and return values. Java-provided objects cannot be handled as communication data.

When you issue requests from OTM, take account of the handling of the `string` and `char` types. Note that IDL code generation commands provided by CTM generate IDL code that can be used in OTM. For details about the handling of the `string` and `char` types, see *9.3.2 Handling characters and strings in OTM*. For details about IDL, see the manual *TPBroker Object Transaction Monitor Programmers Guide*.

9.2.3 Process configuration

ORB clients issue requests via the ORB gateway or hybrid gateway. OTM clients issue requests via the OTM gateway. The TPBroker client and OTM clients can also issue requests by using OSAgent. The following figure shows an overview of process configuration.

Figure 9–1: Overview of process configuration



9.2.4 Starting a gateway

The following table lists gateways, clients that can connect to the gateways, commands for starting the gateways, and specifiable command options.

Table 9–3: Gateways, connectable clients, start commands, and specifiable options

Gateway type		Client	Command	Option		
				-CTMAgent1	-CTMIDLConnect	-CTMTSCGwStart
CTM regulator	EJB regulator	EJB	`ctmstart`	N	N	N
	Hybrid gateway	EJB, ORB	`ctmstart`	Δ#	Δ#	N
ORB gateway		ORB	`ctmstartgw`	Δ	Δ	N
OTM gateway		OTM	`ctmstart`	N	N	Y

Legend:

Y: This option can be specified.

Δ: Either or both of these options must be specified.

N: This option cannot be specified.

#

Specify these options in the file specified for the `-CTMRegOption` option.

To receive requests from EJB and ORB clients collectively with the same gateway, start a CTM regulator (hybrid gateway). To receive requests from EJB and ORB clients separately with different gateways, start a CTM regulator (EJB regulator) and ORB gateway.

For details about the commands and options, see the *uCosminexus Application Server Command Reference Guide*.

9.2.5 Reference resolver and lookup name specification

This section describes the reference resolver and lookup name specification.

(1) Reference resolver for the client to invoke an EJB application

The following table shows the reference resolvers that can be used to invoke an EJB application for each client type.

Table 9–4: EJB application reference resolvers

Client		Reference resolver		
		CORBA Naming Service (INS)	OSAgent	IOR string
EJB		Y	N	N
OTM		N	Y	N
ORB	TPBroker	N	Y	Y
	Other than TPBroker	N	N	Y

Legend:

Y: Supported

N: Not supported

(2) Specifying the lookup name

For OTM Client

For an OTM client, specify the TSC acceptor name as the argument for the constructor that generates a TSC user proxy. For the TSC acceptor name, specify the lookup (registration) name of the EJB application. The TSC acceptor name cannot be omitted. For the connection mode, select a connection via a TSC regulator.

Specification examples where the lookup name of the EJB application is `AAA/a1`:

- **C++**

```
Converter_TSCprxy(TSCClient_ptr _tsc_client, "AAA/a1");
```

- **Java**

```
Converter_TSCprxy(TSCClient _tsc_client, new String("AAA/a1"));
```

For details about the procedure for creating an application, see *9.3.1 Procedure for creating an OTM application*.

For TPBroker Client

If you specify the `-CTMAgent 1` option in the `ctmstartgw` or `ctmregltd` command, the command registers a CORBA reference in OSAgent by using the lookup name of the EJB application as an object name.

Therefore, on the TPBroker client, resolve the reference by specifying the lookup name of the EJB application as the argument for the `_bind()` method.

Specification examples for specifying the default lookup name of the `converter` sample program:

- **C++**

```
Converter::_bind("HITACHI_EJB/SERVERS/MyServer/EJB/converter/MyConverte
r");
```

- **Java**

```
ConverterHelper::_bind(org.omg.CORBA.ORB orb,
      new String("HITACHI_EJB/SERVERS/MyServer/EJB/converter/MyConverte
r"));
```

For details about the procedure for creating an application, see *9.4 Implementation procedures for EJB invocation from ORB clients*.

9.3 Implementation procedures for EJB invocation from OTM applications

This section describes the procedure for creating an application by using the `converter` sample program of Cosminexus Component Container.

9.3.1 Procedure for creating an OTM application

1. Create an EJB application (server application).

   Creation of an EJB application (server application) requires the following Java programs:

   - `Converter.java`: EJBObject inherited class
   - `ConverterEJB.java`: SessionBean implementation class
   - `ConverterHome.java`: EJBHome inherited class

   When an Enterprise Bean is compiled by using the `compileBean.bat` sample file, the following files are generated:

   - `Converter.class`
   - `ConverterEJB.class`
   - `ConverterHome.class`
   - `converter.jar`

   For details, see the *uCosminexus Application Server Application Development Guide*.

2. Generate IDL files.

   Use the IDL generation command provided by CTM to generate IDL files from a `.class` file generated in step 1:

   ```
   % ctmjava2idl -g -o Converter.idl Converter.class
   ```

   - `Converter.id`
   - `TSCjava.idl`

   Perform this step in a Cosminexus environment. Because `java2idl` of TPBroker V5 is required, set the `bin` directory of Cosminexus TPBroker for the `PATH` environment variable.

3. Generate stubs.

   With the IDL files generated in step 2, generate stubs by using the stub generation command provided by CTM.

   Perform this step in a client development environment. Note that the OTM environment for executing client applications and its underlying TPBroker environment must have been set up.

   For C++ applications

   `% ctmidl2cpp -gen_included_files Converter.idl`

   When the preceding command is executed, the following files, which are required for creating a client application, are generated:

   · `Converter_c.hh`
   · `Converter_c.cc`
   · `Converter_s.hh`
   · `Converter_s.cc`
   · `Convtrter_TSC_c.hh`

· `Converter_TSC_c.cc`

For Java applications

`% ctmidl2j -gen_included_files Converter.idl`

When the preceding command is executed, the following files, which are required for creating a client application, are generated:

· `Converter.java`

· `ConverterHelper.java`

· `ConverterOperations.java`

· `ConveterPOA.java` (None#)

· `_Converter_Stub.java (_st_Converter.java#)`

· `_Converter_Tie.java (_tie_Converter.java#)`

· `Converter_TSCprxy.java`

· Classes for other structure-type data

File output only if the TPBroker version is V3

4. Perform compilation and linkage.

   Following the OTM application development procedure, perform compilation and linkage by using the stub files generated in step 3.

9.3.2 Handling characters and strings in OTM

The following table lists the data types with which character strings can be handled in OTM.

Table 9–5: Data types supported by OTM

No.	Definition in the remote interface	Data type used on the client
1	string	`::TSC::TSCWStringValue`
2	char	`::TSC::TSCWChar`

(1) ::TSC::TSCWStringValue

In IDL, `::TSC::TSCWStringValue` is defined as follows:

```
module TSC {
  typedef sequence<octet> TSCWString;
  struct TSCWStringValue {
    TSCWString value;
  };
};
```

The following sections show the functions provided to use data of the `::TSC::TSCWStringValue` type as actual strings of the `wchar_t*` type.

(a) For the C++ interface

- Include file

  `#include "Converter_TSC_c.hh"` (stub header)

- Provided functions

  `void TSCsetWString( ::TSC::TSCWString&, const ::CORBA::WChar* )`

  Description:

  This function sets a wide string for `::TSC::TSCWString`.

  I/O variables:

  `::TSC::TSCWString& :out`: Variable of the `::TSC::TSCWString` type to be set

  `const ::CORBA::WChar*:in`: Wide string to be set

  `::CORBA::WChar* TSCgetWString( ::TSC::TSCWString )`

  Description:

  This function retrieves the wide string that is set in `::TSC::TSCWString`.

  I/O variables:

  `::TSC::TSCWString :in`: Object that holds the wide string

  Return value:

  Wide string retrieved from `::TSC::TSCWString`

  Note:

  If you use this interface with OTM V3, use `delete[]` to release the return value.

- Sample code

```
CORBA::WChar* wstr_data = new CORBA::WChar[5];
wstr_data[0] = L 'さ' ;
wstr_data[1] = L 'ん' ;
wstr_data[2] = L 'ぷ' ;
wstr_data[3] = L 'る' ;
wstr_data[4] = 0;
::TSC::TSCWStringValue tsc_wstr_value_data;

// Set a wide string
TSCsetWString( tsc_wstr_value_data.value, wstr_data );
// Release the area that is no longer used
delete[] wstr_data;

// Retrieve the wide string
wstr_data = TSCgetWString( tsc_wstr_data.value );
// Release the area from which the string was retrieved
delete[] wstr_data;
```

(b) For the Java interface

No functions are provided for the Java interface.

- Sample code

```
String wstr_data = new String(" さんぷる ");
System.out.println(wstr_data);
TSC.TSCWStringValue tsc_wstr_value_data = new TSC.TSCWStringValue();

// Set a wide string
tsc_wstr_value_data.value = wstr_data.getBytes("UTF-16");
```

```
// Retrieve the wide string
wstr_data = new String( tsc_wstr_value_data.value );
```

(2) ::TSC::TSCWChar

In IDL, `::TSC::TSCWChar` is defined as follows:

```
module TSC {
  typedef octet TSCWChar[3];
};
```

The following sections show the functions provided to use data of the `::TSC::TSCWChar` type as actual strings of the `wchar_t*` type.

(a) For the C++ interface

- Include file

  `#include "Converter_TSC_c.hh"` (stub header)

- Provided functions

  `void TSCsetWChar( ::TSC::TSCWChar , ::CORBA::WChar )`

  Description:

  This function sets a wide string for `::TSC::TSCWChar`.

  I/O variables:

  `::TSC::TSCWChar :out`: Area to be set

  `const ::CORBA::WChar :in:` Wide string to be set

  Return value:

  None

  `::CORBA::WChar TSCgetWChar( ::TSC::TSCWChar )`

  Description:

  This function retrieves the wide string that is set in `::TSC::TSCWChar`.

  I/O variables:

  `const ::TSC::TSCWChar :in`:Object that holds a wide string

  Return value:

  Wide string retrieved from `::TSC::TSCWChar`

- Sample code

```
::TSC::TSCWChar tsc_wchar_data;
// Set a wide string
TSCsetWChar( tsc_wchar_data, L' あ ' );

// Retrieve the wide string
CORBA::WChar wchar_data = TSCgetWChar( tsc_wchar_data );
```

(b) For the Java interface

No functions are provided for the Java interface.

- Sample code

```
TSC.TSCWCharHolder tsc_wchar_data = new TSC.TSCWCharHolder();
tsc_wchar_data.value = new byte[3];
char wch_data = ' あ ';

// Set a wide string
tsc_wchar_data.value[0] = (byte)2;
tsc_wchar_data.value[1] = (byte)((wch_data >> 8) & 0xff);
tsc_wchar_data.value[2] = (byte)( wch_data & 0xff);

// Retrieve the wide string
wch_data = (char)(((char)tsc_wchar_data.value[1] & 0xff) << 8 |
((char)tsc_wchar_data.value[2] & 0xff));
```

9.3.3 How to reference exceptions on OTM clients

OTM clients cannot directly catch exception classes thrown by EJB applications. Therefore, exception classes are converted on a TSC user proxy (stub) and OTM clients catch the conversion results.

The following describes exceptions that have been converted on a TSC user proxy.

(1) List of exceptions received by OTM clients

The following table shows the correspondence between exceptions that are generated by EJB applications and exceptions that are received by OTM clients.

Table 9–6: List of exceptions received by OTM clients

No.	Exception generated by EJB applications	Exception received by OTM clients
1	Java user-defined exception (one inheriting `java.lang.Exception`)	User-defined exception (referencing `exception` in the IDL definition)#
2	Java user-defined exception (one inheriting `java.rmi.RemoteException`)	`TSC::java::rmi::RemoteEx` exception
3	Java system-defined exception (one inheriting `java.lang.Exception`)	`TSCUnknown` exception (exception defined in the relevant Java system)
4	Java runtime exception (one inheriting `java.lang.RuntimeException`)	`TSC::java::rmi::RemoteEx` exception
5	Java system-defined error (one inheriting `java.lang.Error`)	`TSC::java::rmi::RemoteEx` exception
6	`java.rmi.RemoteException`	`TSC::java::rmi::RemoteEx` exception
7	CORBA system exception (one inheriting `org.omg.CORBA.SystemException`)	`TSCUnknown` exception (exception defined in the relevant TSC system)
8	CORBA user exception (one inheriting `org.omg.CORBA.UserException`)	User-defined exception (referencing `exception` in the IDL definition)

#

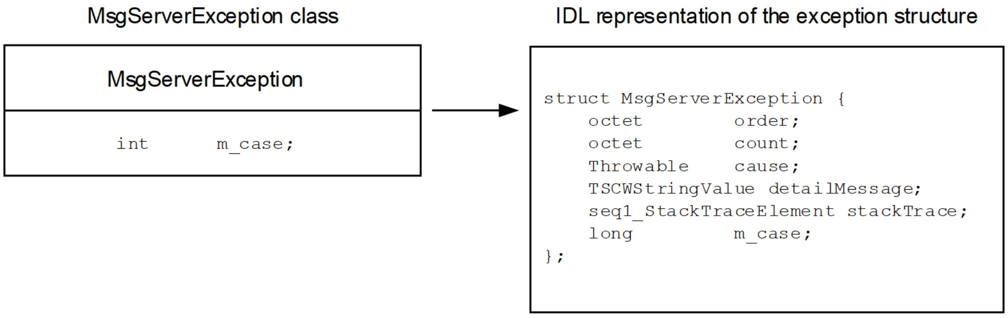
Exceptions with names in the *xx...xx*`Exception` format correspond to exceptions with names in the *xx...xx*`Ex` format. For exceptions whose names do not end with `Exception`, the corresponding exceptions have the same names with a suffix of `Ex`. The value of the `value` member of each exception is an exception structure.

(2) About exception structures

(a) Overview of exception structures

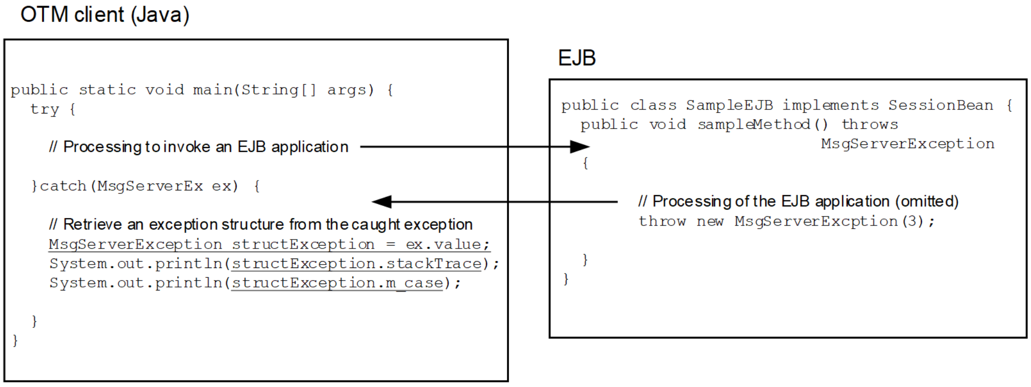
Restrictions placed on an OTM client include inabilities to interpret data that has a recursive structure and to use `valuetype`. Therefore, an exception thrown by an EJB application is converted into an exception that has a structure as a member as shown in the following figure. This structure is called an exception structure.

Figure 9–2: Exception class and exception structure



An exception is converted into an exception structure, and then the OTM client catches an exception that has the exception structure as a member. When a user throws a `MsgServerException` exception from an EJB application, the OTM client catches the exception as a `MsgServerEx` exception. The `value` member of the `MsgServerEx` exception is the `MsgServerException` exception structure.

Figure 9–3: Retrieving an exception structure from an exception class



(b) Member of an exception structure

The following table describes the members contained in an exception structure.

Table 9–7: Members of an exception structure

No.	Structure member	Contents
1	order	OTM application developers do not need to use these items. These items are required for the OTM client to receive the structure as an exception.
2	count	
3	cause	OTM application developers do not need to use these items. On the OTM client, do not throw an exception from the EJB application by setting any value in `cause`.
4	detailMessage	Detail message specified in the super class
5	stackTrace	Stack trace up to the time when an exception is generated on the EJB application side

No.	Structure member	Contents
6	Member variables	Member variables set in a Java user-defined exception. For `RemoteException`, no member variables exist.

(c) Information items of a stack trace

The `stackTrace` member of an exception structure is an array for the `StackTraceElement` structure. The `StackTraceElement` structure contains a single stack of stack traces.

The following table shows the information set in each member of the `StackTraceElement` structure.

Table 9–8: Members of the StackTraceElement structure

No.	Field name	Contents	Example
1	LineNumber	Source line number	29
2	declaringClass	Class name	jp.co.Hitachi.soft.test.ejb.testMsgSyncServiceDelEJB
3	FileName	File name	testMsgSyncServiceDelEJB.java
4	MethodName	Method name	InvokeBinary

(3) Example of referencing exception information on an OTM client

The following describes how an OTM client can reference exceptions generated by an EJB application, by using sample code as examples.

In the examples, the following Java user-defined exception is used.

```
Package jp.co.Hitachi.soft.test;
public class MsgServerException extends Exception
{
    public MsgServerException()
    {
      m_case=1;
    }
    private int m_case;
}
```

(a) Example of referencing exception information with C++

The following shows an example of referencing exception information with C++ code:

```
#define ERR_FORMAT "EC=%d,DC=%d,PC=%d,CS=%d,MC1=%d,MC2=%d,MC3=%d,MC4=%d\n"
// Function that returns the length of the specified string
int my_wstringlen(CORBA::WChar* arg){
    int i;
    for(i=0;arg[i] != 0;i++);
    return i;
}
// Function that displays the specified string
void my_print_wstring(CORBA::WChar* arg){
    for(int i=0;i < my_wstringlen(arg); i++){
      printf("%c", arg[i]);
    }
```

```
}

int main(int argc, char** argv){
  try {
      // Processing to invoke the EJB application (omitted)
  }catch(jp::co::Hitachi::soft::test::MsgServerEx &e){
   // EJB application throws a Java user-defined exception (MsgServerExcepti
on)
    jp::co::Hitachi::soft::test::CSCMsgServerException& ex_val = e.value;

    // Output a detail message
    printf("detailMessage:");
    ::TSC::TSCWStringValue& detailMessage = ex_val.detailMessage;
    CORBA::WChar* w_detail_msg = new CORBA::WChar[detailMessage.value.length
()];
    w_detail_msg = TSCgetWString(detailMessage.value );
    my_print_wstring(w_detail_msg);
    printf("\n");

    // Output the m_case member variable
    printf("m_case:%d", ex_val.m_case);
    // Output a stack trace
    ::TSC::java::lang::seq1_StackTraceElement& stackTrace = ex_val.stackTrac
e;
    CORBA::ULong len = stackTrace.value.length();
    :TSC::java::lang::StackTraceElement stackTraceElement;
    for ( int i = 0 ; i < len ; i ++ ) {
        stackTraceElement = stackTrace.value [ i ] ;
        ::TSC::TSCWStringValue& className = stackTraceElement.declaringClas
s;
        ::TSC::TSCWStringValue& methodName= stackTraceElement.methodName;
        ::TSC::TSCWStringValue& fileName = stackTraceElement.fileName;
        CORBA::Long lineNumber= stackTraceElement.lineNumber;

        CORBA::WChar* w_class_name = new CORBA::WChar[className.value.length
()];
        w_class_name = TSCgetWString( className.value );

      CORBA::WChar* w_method_name = new CORBA::WChar[methodName.value.length
()];
      w_method_name = TSCgetWString( methodName.value );

      CORBA::WChar* w_file_name = new CORBA::WChar[fileName.value.length()];
      w_file_name = TSCgetWString( fileName.value );

      printf("at ");
      my_print_wstring(w_class_name);
      printf(".");
      my_print_wstring(w_method_name);
      printf("(");
      my_print_wstring(w_file_name);
      printf(":%d)\n",lineNumber);
    }
}catch(TSC::java::rmi::RemoteEx &e){
    // EJB application throws a Java runtime exception
    // EJB application throws a Java system-defined error
    // EJB application throws a java.rmi.RemoteException
```

```
    TSC::java::rmi::RemoteException& ex_val = e.value;

    // Output a detail message
    printf("detailMessage:");
    ::TSC::TSCWStringValue& detailMessage = ex_val.detailMessage;
    CORBA::WChar* w_detail_msg = new CORBA::WChar[detailMessage.value.length
()];
    w_detail_msg = TSCgetWString(detailMessage.value );
    my_print_wstring(w_detail_msg);
    printf("\n");

    // Output a stack trace
    ::TSC::java::lang::seq1_StackTraceElement& stackTrace = ex_val.stackTrac
e;
    CORBA::ULong len = stackTrace.value.length();
    :TSC::java::lang::StackTraceElement stackTraceElement;
    for ( int i = 0 ; i < len ; i ++ ) {
        stackTraceElement = stackTrace.value [ i ] ;
        ::TSC::TSCWStringValue& className = stackTraceElement.declaringClas
s;
        ::TSC::TSCWStringValue& methodName= stackTraceElement.methodName;
        ::TSC::TSCWStringValue& fileName = stackTraceElement.fileName;
        CORBA::Long lineNumber= stackTraceElement.lineNumber;

        CORBA::WChar* w_class_name = new CORBA::WChar[className.value.length
()];
        w_class_name = TSCgetWString( className.value );
      CORBA::WChar* w_method_name = new CORBA::WChar[methodName.value.length
()];
      w_method_name = TSCgetWString( methodName.value );

      CORBA::WChar* w_file_name = new CORBA::WChar[fileName.value.length()];
      w_file_name = TSCgetWString( fileName.value );

      printf("at ");
      my_print_wstring(w_class_name);
      printf(".");
      my_print_wstring(w_method_name);
      printf("(");
      my_print_wstring(w_file_name);
      printf(":%d)\n",lineNumber);
    }
} catch(TSCSystemException& se) {
    // EJB application throws a Java system-defined error
    // EJB application throws a CORBA system exception
    // Invocation of the EJB application fails
    printf(ERR_FORMAT,
            se.getErrorCode(), se.getDetailCode(),
            se.getPlaceCode(), se.getCompletionStatus(),
            se.getMaintenanceCode1(), se.getMaintenanceCode2(),
            se.getMaintenanceCode3(), se.getMaintenanceCode4());
} catch(UserExcept& se) {
    // EJB application throws a CORBA user exception
    printf("UserExcept\n");
}
```

(b) Example of referencing exception information with Java

The following shows an example of referencing exception information with Java code.

```
// Function that performs byte[]-to-char[] conversion to obtain String
private static String myString(byte[] barray) {
    char[] carry = new char[barray.length/2];
    for ( int i=0;i<carry.length;i++) {
        carry[i] = (char)((( barray[i*2]& 0xff) << 8 ) | barray[(i*2)+1]& 0x
ff) ;
    }
    return new String(carry);
}

public static void main(String[] args) {
  try {
      // Processing to invoke an EJB application (omitted)
  }catch(jp.co.Hitachi.soft.test.MsgServerEx ex) {

     // EJB application throws a Java user-defined exception (MsgServerExcep
tion)
    jp.co.Hitachi.soft.csc.msg.message.reception.MsgServerException testExce
ption = ex.value;

    // Output a detail message
    System.out.println ("detailMessage:"+ myString(testException.detailMessa
ge);

    // Output the m_case member variable
    System.out.println ("m_case:%d"+ testException .m_case);

    // Output a stack trace
    TSC.java.lang.StackTraceElement[] stackElements= testException.stackTrac
e.value;
    java.lang.StringBuffer stb = new java.lang.StringBuffer();

    for (int i=0;i<stackElements.length;i++) {
        stb = stb.append("at ");
        stb = stb.append(myString(stackElements[i].declaringClass.value));
        stb = stb.append(".");
        stb = stb.append(myString(stackElements[i].methodName.value));
        stb = stb.append("(");
        stb = stb.append(myString(stackElements[i].fileName.value));
        stb = stb.append(":");
        stb = stb.append(stackElements[i].lineNumber);
        stb = stb.append(")");
        stb = stb.append("\n");
    }
        System.out.println(stb.toString());
}catch(TSC.java.rmi.RemoteEx ex) {
    // EJB application throws a Java runtime exception
    // EJB application throws a Java system-defined error
    // EJB application throws a java.rmi.RemoteException
    TSC.java.rmi.RemoteException testException = ex.value;

    // Output a detail message
    System.out.println ("detailMessage:"+ myString(testException.detailMessa
ge);
```

```
    // Output a stack trace
    TSC.java.lang.StackTraceElement[] stackElements= testException.stackTrac
e.value;
    java.lang.StringBuffer stb = new java.lang.StringBuffer();

    for (int i=0;i<stackElements.length;i++) {
        stb = stb.append("at ");
        stb = stb.append(myString(stackElements[i].declaringClass.value));
        stb = stb.append(".");
        stb = stb.append(myString(stackElements[i].methodName.value));
        stb = stb.append("(");
        stb = stb.append(myString(stackElements[i].fileName.value));
        stb = stb.append(":");
        stb = stb.append(stackElements[i].lineNumber);
        stb = stb.append(")");
        stb = stb.append("\n");
    }
        System.out.println(stb.toString());
} catch(TSCSystemException tsc_se) {
    // EJB application throws a Java system-defined exception
    // EJB application throws a CORBA system exception
    // Invocation of the EJB application fails
    System.out.println(tsc_se);
} catch(UserExcept tsc_se) {
    // EJB application throws a CORBA user exception
    System.out.println("catch" + tsc_se.value);
}
```

9.4 Implementation procedures for EJB invocation from ORB clients

This section describes the implementation procedures for EJB invocation from ORB clients.

9.4.1 Procedure for creating a TPBroker V5 application

This section describes the procedure for creating an application by using TPBroker V5.

For details about how to create an application for ORB clients other than TPBroker V5, see *9.4.2 Procedure for creating an application for ORB clients other than TPBroker V5*.

1. Create an EJB application (server application).

   Creation of an EJB application (server application) requires the following Java programs:

   - `Converter.java`: EJBObject inherited class
   - `ConverterEJB.java`: SessionBean implementation class
   - `ConverterHome.java`: EJBHome inherited class

   When an Enterprise Bean is compiled by using the `compileBean.bat` sample file, the following files are generated:

   - `Converter.class`
   - `ConverterEJB.class`
   - `ConverterHome.class`
   - `converter.jar`

   For details, see the *uCosminexus Application Server Application Development Guide*.

2. Generate IDL files.

   If you are creating a C++ application, use the following command provided by TPBroker to generate IDL files from a `.class` file generated in step 1:

   ```
   % java2idl Converter.class > Converter.idl
   ```

3. Generate stubs.

   Generate stubs from the IDL files generated in step 2 or from a `.class` file.

   Perform this step in a client development environment.

   For C++ applications

   ```
   % idl2cpp -namespace Converter.idl
   ```

   When the preceding command is executed, the following files, which are required for creating a client application, are generated:

   · `Converter_c.cc`

   · `Converter_c.hh`

   · `Converter_s.cc`

   · `Converter_s.hh`

   For Java applications

   ```
   % java2iiop Converter.class
   ```

When the preceding command is executed, the following files, which are required for creating a client application, are generated:

· `Converter.java`

· `ConverterHelper.java`

· `ConverterHolder.java`

· `ConverterOperations.java`

· `ConveterPOA.java`

· `Converter_Stub.java`

· `Converter_Tie.java`

· Classes for other structure-type data

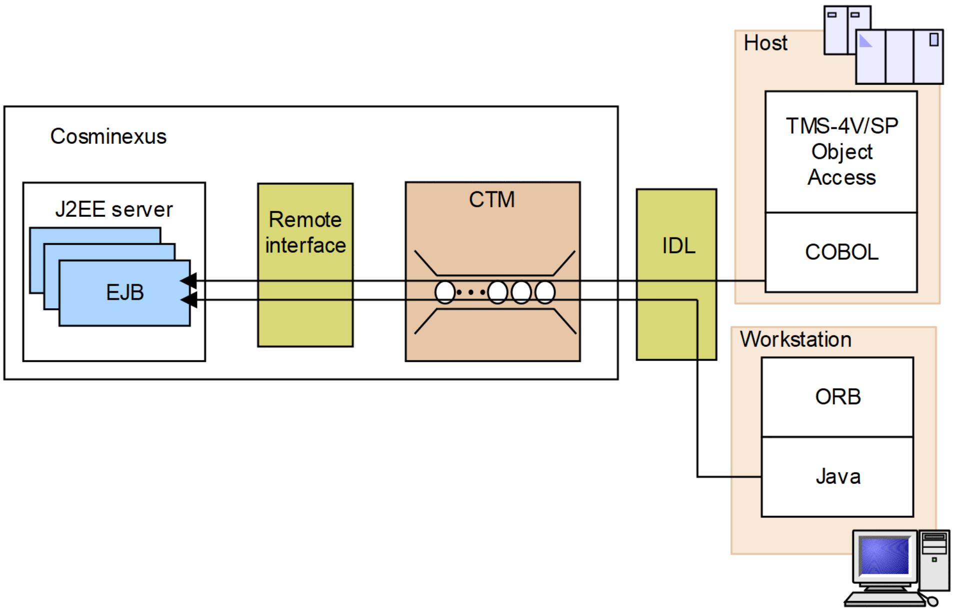
4. Perform compilation and linkage.

   Following the TPBroker application development procedure, perform compilation and linkage by using the stub files generated in step 3.

9.4.2 Procedure for creating an application for ORB clients other than TPBroker V5

The following figure shows a method whereby EJBs of an EJB application (Cosminexus) are invoked via CORBA communication from an ORB client other than TPBroker V5 (CORBA client compliant with CORBA 2.1; TMS-4V/SP Object Access; or TPBroker V3).

Figure 9–4: Invoking EJBs from an ORB client



With this method, only the following data types and data definitions can be used in IDL:

`long`, `unsigned long`, `string`, `sequence<octet>`, `struct`

(1) Interface design procedure

The communication interface on the Application Server side must be defined as a remote interface of EJB. The communication interface on the ORB client must be defined as an IDL definition of CORBA.

The following shows an overview of the procedure for designing the remote interface and IDL definition:

1. Create the IDL definition on which the interface will be based.
2. Create a remote interface of EJB according to the IDL definition.
3. Correct the order of the structure members in the IDL definition based on the information output by the `ctmjava2idl` command.
4. Change the IDL definitions for the `string`, `sequence<octet>`, and `struct` members.

The following describes each step of the preceding procedure.

(a) Creating an IDL definition on which the interface will be based

Note the following when creating an IDL definition on which the interface will be based:

- The `out` and `inout` arguments cannot be used. Use a return value as the data to be returned. If multiple pieces of data must be returned, put them into a `struct`.
- Do not use user exceptions.
- For `struct`, you can use only the `long`, `unsigned long`, `string`, and `sequence<octet>` members.
- Do not use `#pragma`.

(b) Creating a remote interface

After you have created a base IDL definition, create a remote interface of EJB based on the IDL definition.

At this time, make sure that the corresponding members between the IDL definition and remote interface have the same names.

Note that *modules* in IDL correspond to *packages* in Java.

The following table shows the mapping of supported data types.

Table 9–9: Mapping of data types from IDL to the remote interface

Data Type	IDL	Remote interface
Integer	`long`	`int`
	`unsigned long`#1	
String	`string`#3	`byte[]`#2
Binary	`sequence<octet>`#3	`byte[]`
Structure	`struct`	`class` (`Serializable` implemented)

#1

Java does not provide a data type that corresponds to `unsigned`. Therefore, `unsigned long` in IDL is handled as `long`.

#2

`String` in Java corresponds to `Wstring` in IDL. Therefore, in the remote interface, use `byte[]` instead of `String`.

#3

In IDL, maximum lengths can be set for `string` and `sequence<octet>`. In Java, however, values of those types are handled as variable-length data.

Coding example in IDL:

```
struct AAA {
  long longData;
  string<4> strData;
  unsigned long ulongData;
  sequence<octet> octseqData;
};
```

Coding example in the remote interface:

```
public class AAA
  implements java.io.Serializable {
  public int longData;
  public byte[] strData;
  public int ulongData;
  public byte[] octSeqData;
};
```

(c) Correcting the order of structure members in IDL based on the information output by the ctmjava2idl command

To enable CORBA communication with EJB, the members of `struct` in IDL must be sorted according to the specifications of `Serializable` in Java. Use the `ctmjava2idl` command to check the correct order of the members of `struct` in IDL.

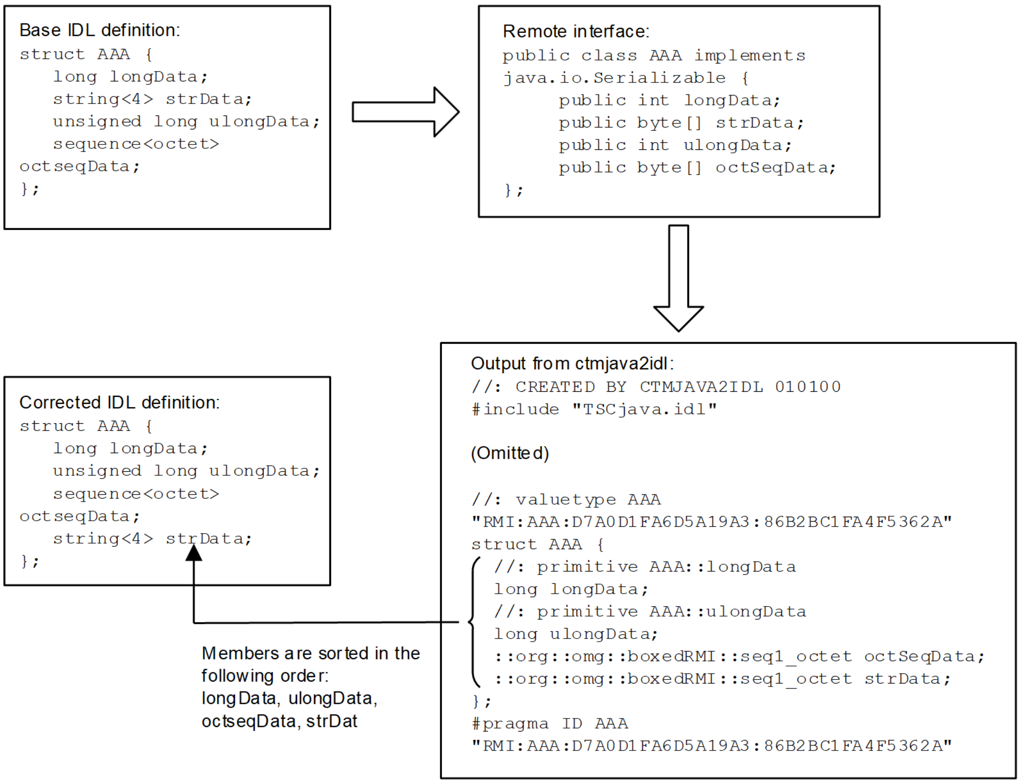
The `ctmjava2idl` command converts a remote interface of EJB into an IDL definition.

The following shows an example procedure in a case where the `AAA` structure shown in the example for *Table 9-9* is used:

1. Compile `AAA.java`, which is the Java source program for `AAA`.
   `% javac AAA.java`
2. Using the generated class file (`AAA.class`) as the input, execute the `ctmjava2idl` command:
   `% ctmjava2idl AAA.class`
3. Correct the base IDL definition by referring to the IDL definition that has been output to the standard output.

The following figure shows an example of changing the order of structure members in IDL based on the information output by the `ctmjava2idl` command.

Figure 9–5: Example of changing the order of structure members in IDL based on the information output by the ctmjava2idl command



Reference note

The rules on the order of `struct` members are as follows:

The order of members is determined based on the data type and member name in the remote interface.

1. Members of type `int` come before members of type `byte[]`.
2. Members of type `int` are sorted by name in character code order.
3. Members of type `byte[]` are sorted by name in character code order.
4. Sort the members of type `int` and the members type `byte[]` separately based on the order of `struct` members in the IDL definition output by the `ctmjava2idl` command.

(d) Changing the IDL definitions of string, sequence<octet>, and struct

In EJB invocation from an ORB client, the data of types `string`, `sequence<octet>`, and `struct` cannot be directly handled. The data of these types must be changed to structures in a specific format. The data of types `long` and `unsigned long` can be directly handled.

Format of send data on the ORB client side:

The format of the data to be sent from the ORB client to EJB must be changed to the following format.

Note that the following is an example applicable to the `in` arguments `StringValue`, `OctSeqValue`, and `AAARequest` of the `invoke` method in *Figure 9-6 Conversion example of an IDL definition for EJB*.

```
struct user-defined-structure-name {
  long VALUE_TAG;
  string REPOSITORY_ID;
```

```
  data-body;
}
```

For the `string` and `sequence<octet>` members of `struct`, do not use the preceding structure. Instead, prefix the following structure to each data item.

Note that the following is an example applicable to the `tag1` and `tag2` members of `AAARequest` in *Figure 9-6 Conversion example of an IDL definition for EJB*.

```
struct ValueTag {
  long VALUE_TAG;
  string REPOSITORY_ID;
}
```

For program implementations on the ORB client, values must be set for `VALUE_TAG` and `REPOSITORY_ID`.

Format of receive data on the ORB client side:

For `struct` itself as the return value, change its format in the same way as for send data.

Note that the following is an example applicable to `AAAReply` in *Figure 9-6 Conversion example of an IDL definition for EJB*.

```
struct user-defined-structure-name {
  long VALUE_TAG;
  string REPOSITORY_ID;
  data-body;
}
```

For the first occurrences of `string` and `sequence<octet>` members among all members of `struct`, prefix the following structure to their data.

Note that the following is an example applicable to the `tag1` member of `AAAReply` in *Figure 9-6 Conversion example of an IDL definition for EJB*.

```
struct ValueTag {
  long VALUE_TAG;
  string REPOSITORY_ID;
}
```

For the second and subsequent occurrences of the `string` and `sequence<octet>` members, prefix the following structure to their data.

Note that the following is an example applicable to the `tag2` member of `AAAReply` in *Figure 9-6 Conversion example of an IDL definition for EJB*. In this case, `ReplyTag` is used instead of `ValueTag` because an occurrence of the `sequence<octet>` member exists before the `tag2` member.

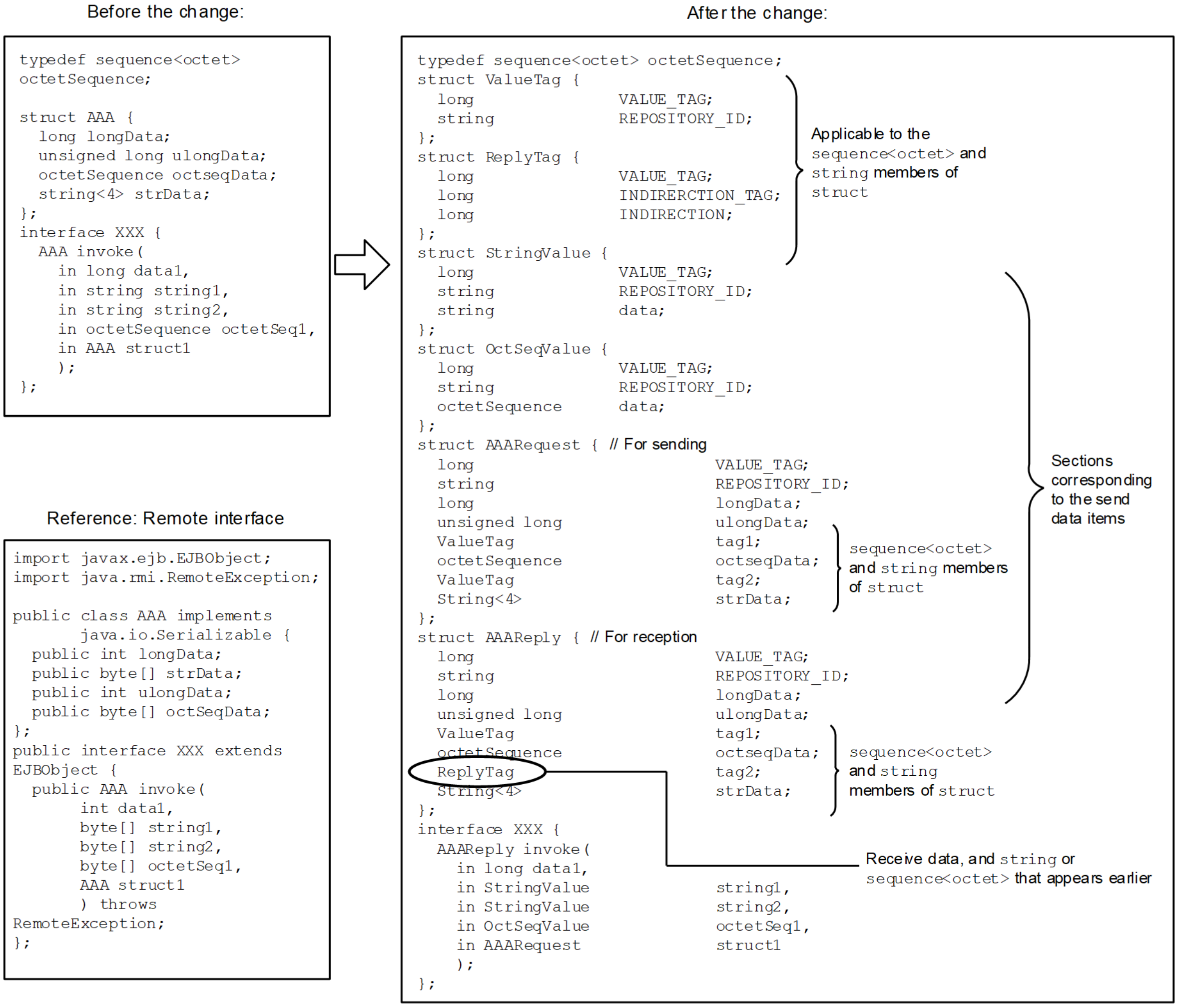
```
struct ReplyTag {
  long VALUE_TAG;
  long INDIRECTION_TAG;
  long INDIRECTION;
}
```

For receive data, ignore the values of `VALUE_TAG`, `REPOSITORY_ID`, `INDIRECTION_TAG`, and `INDIRECTION`.

Example of changing an IDL definition:

The following figure shows an example of changing an IDL definition (including the corresponding remote interface).

Figure 9–6: Conversion example of an IDL definition for EJB



(2) Program implementations related to communication processing

This section describes the processing to be added as program implementations.

(a) VALUE_TAG and REPOSITORY_ID settings in the send data on the ORB client

For `string`, `sequence<octet>`, and `struct`, if you send data from the ORB client to EJB, beforehand, set values also for `VALUE_TAG` and `REPOSITORY_ID` in addition to the data body.

VALUE_TAG

Always set the fixed value `0x7fffff02`.

REPOSITORY_ID

For `string` and `sequence<octet>`:

Set the fixed value `"RMI:[B:0000000000000000"`.

For `struct`:

Because the repository ID differs depending on the name and structure of the `struct` member, obtain the repository ID from the output results of the `ctmjava2idl` command. In the output results, under the `#pragma ID` *class-name* line, the `"RMI:`*xxxxx*`"` portion is the repository ID. For example, in the following figure, the repository ID of `AAA` is `"RMI:AAA:D7A0D1FA6D5A19A3:86B2BC1FA4F5362A"`.

Figure 9–7: Example of the repository ID

```
Output from ctmjava2idl:
//: CREATED BY CTMJAVA2IDL 010100
#include "TSCjava.idl"

(Omitted)

//: valuetype AAA
"RMI:AAA:D7A0D1FA6D5A19A3:86B2BC1FA4F5362A"
struct AAA {
  //: primitive AAA::longData
  long longData;
  //: primitive AAA::ulongData
  long ulongData;
  ::org::omg::boxedRMI::seq1_octet octSeqData;
  ::org::omg::boxedRMI::seq1_octet strData;
};
#pragma ID AAA
"RMI:AAA:D7A0D1FA6D5A19A3:86B2BC1FA4F5362A"
```

If you specify the length of the `REPOSITORY_ID` section in the COBOL map file, specify the following value:

For `string` or `sequence<octet>`:

23 or more characters

(Numeric value equal to or larger than the length of the fixed value `"RMI:[B:0000000000000000"`)

For `struct`:

Numeric value equal to or larger than the number of characters in the repository ID obtained from the output results of the `ctmjava2idl` command

For the receive data on the ORB client side, ignore all information (including `VALUE_TAG`) except the data body.

The following figure shows an example of specifying the `VALUE_TAG` and `REPOSITORY_ID` settings applicable to `XXX::invoke()` in *Figure 9-6 Conversion example of an IDL definition for EJB*.

Figure 9–8: Example of specifying the VALUE_TAG and REPOSITORY_ID settings (COBOL)

```
 DATA DIVISION.

  03 DATA1                      PIC S9(9)COMP.
  03 STRING1.
   04 VVLUE-TAG1                PIC X(4).
   04 REPOSITORY-ID1            PIC X(23).
   04 DATA-LEN                  PIC 9(9) COMP.
   04 DATA                      PIC X(512).
  03 STRING2.
   04 VVLUE-TAG2                PIC X(4).
   04 REPOSITORY-ID2            PIC X(23).
   04 DATA-LEN                  PIC 9(9) COMP.
   04 DATA                      PIC X(512).
  03 OCTETSEQ1.
   04 VVLUE-TAG3                PIC X(4).
   04 REPOSITORY-ID3            PIC X(23).
   04 DATA-LEN                  PIC 9(9) COMP.
   04 DATA                      PIC X(512).
  03 AAAREQUEST.
   04 VVLUE-TAG4                PIC X(4).
   04 REPOSITORY-ID4            PIC X(41).
   04 LONGDATA                  PIC S9(9) COMP.
   04 ULONGDATA                 PIC 9(9) COMP.
   04 FILLER. . . .
    05 VVLUE-TAG5               PIC X(4).
    05 REPOSITORY-ID5           PIC X(23).
   04 OCTSEQ-DATA-LEN           PIC 9(9) COMP.
   04 OCTSEQ-DATA               PIC X(512).
   04 FILLER.
    05 VVLUE-TAG6               PIC X(4).
    05 REPOSITORY-ID6           PIC X(23).
   04 STR-DATA-LEN              PIC 9(9) COMP.
   04 STR-DATA                  PIC X(4).

  77 REPO-ID-AAA                PIC X(66) VALUE 'RMI:AAA:D7A0D1FA6D5A19A3:
                                              - '86B2BC1FA4F5362A'.
  77 REPO-C                     PIC X(23) VALUE 'RMI:[B:0000000000000000'.
  77 VTAG-C                     PIC X(4) VALUE X'7FFFFF02'.

 PROCEDURE DIVISION.

      MOVE VTAG-C               TO VVLUE-TAG1
      MOVE REPO-C               TO REPOSITORY-ID1
      MOVE VTAG-C               TO VVLUE-TAG2
      MOVE REPO-C               TO REPOSITORY-ID2
      MOVE VTAG-C               TO VVLUE-TAG3
      MOVE REPO-C               TO REPOSITORY-ID3
      MOVE VTAG-C               TO VVLUE-TAG4
      MOVE REPO-ID-AAA          TO REPOSITORY-ID4
      MOVE VTAG-C               TO VVLUE-TAG5
      MOVE REPO-C               TO REPOSITORY-ID5
      MOVE VTAG-C               TO VVLUE-TAG6
      MOVE REPO-C               TO REPOSITORY-ID
```

Data division

Operation argument definition

Procedure division

Processing to specify the VALUE_TAG and REPOSITORY_ID settings

Figure 9–9: Example of specifying the VALUE_TAG and REPOSITORY_ID settings (Java)

```
/* Obtain the EJB reference */
org.omg.CORBA.Object object = orb.string_to_object("IOR:・・・");
XXX exRecive = XXXHelper.narrow(object);

/* Set ValueTag */
final int vtg = 0x7FFFFF02;
final String rid="RMI:[B:0000000000000000";
ValueTag wVl = new ValueTag(vtg,rid);

/* Declaration of send data */
final int wldata=100;
final int wuldata=200;

/* Creation of send data (string) */
StringValue wsvdata = new StringValue();
wsvdata.VALUE_TAG = 0x7FFFFF02;
wsvdata.REPOSITORY_ID="RMI:[B:0000000000000000";
wsvdata.data= new String("data");

/* Creation of send data (octet sequence) */
byte[] wbdata=new byte[3];
wbdata[0]=0x01;
wbdata[1]=0x02;
wbdata[2]=0x03;
OctSeqValue wovdata = new OctSeqValue();
wovdata.VALUE_TAG = 0x7FFFFF02;
wovdata.REPOSITORY_ID="RMI:[B:0000000000000000";
wovdata.data=wbdata;

/* Creation of send data (AAARequest) */
AAARequest requestData = new AAARequest();
requestData.VALUE_TAG = 0x7FFFFF02;
requestData.REPOSITORY_ID= "RMI:AAA:D7A0D1FA6D5A19A3:86B2BC1FA4F5362A";
requestData.longData=wldata;
requestData.ulongData=wuldata;
requestData.tag1=wVl;
requestData.octseqData=wbdata;
requestData.tag2=wVl;
requestData.strData= new String("test");

/* EJB invocation */
AAAReply replyData = exRecive.invoke( wldata,
                                wsvdata,
                                wsvdata,
                                wovdata,
                                requestData);
```

Figure 9–10: Example of specifying the VALUE_TAG and REPOSITORY_ID settings (C++)

```
/* Obtain the EJB reference */
XXX_var exrcv = XXX::_bind("HITACHI_EJB/…");

ValueTag wvl;
wvl.VALUE_TAG=0x7FFFFF02;
wvl.REPOSITORY_ID="RMI:[B:0000000000000000";

/* Declaration of send data */
int wldata=100;
int wuldata=200;

/* Creation of send data (string) */
StringValue wsvdata ;
wsvdata.VALUE_TAG = 0x7FFFFF02;
wsvdata.REPOSITORY_ID="RMI:[B:0000000000000000";
wsvdata.data= "data";

/* Creation of send data (octet sequence) */
OctSeqValue wovdata;
wovdata.VALUE_TAG = 0x7FFFFF02;
wovdata.REPOSITORY_ID="RMI:[B:0000000000000000";

char *str="sendData";
unsigned char* buf = new unsigned char[strlen(str)+1];
strcpy((char *)buf,str);
const octetSequence wk(strlen(str)+1,strlen(str)+1,buf,1);
wovdata.data=wk;

/* Creation of send data (AAARequest) */
AAARequest requestData ;
requestData.VALUE_TAG = 0x7FFFFF02;
requestData.REPOSITORY_ID= "RMI:AAA:D7A0D1FA6D5A19A3:86B2BC1FA4F5362A";
requestData.longData=wldata;
requestData.ulongData=wuldata;
requestData.tag1=wvl;
requestData.octseqData=wk;
requestData.tag2=wvl;
requestData.strData="test";

/* EJB invocation */
AAAReply_var replyData = exrcv->invoke( wldata,
                                  wsvdata,
                                  wsvdata,
                                  wovdata,
                                  requestData);
```

(b) Handling of IDL string data in EJB

On the EJB side, strings being sent or received are handled as data of type `byte[]`. Therefore, perform conversion from or to `String` in Java if necessary.

If received `byte[]` data is directly converted into `String` in Java, the code `\0` is suffixed. Therefore, if you retrieve the string length, the length one character longer than the actual string length will be returned. If necessary, delete the code `\0` at the end of the string.

Conversely, when you set a string in `byte[]`, make sure that the program explicitly suffixes the code `\0` to the string. If data not terminated by `\0` is returned as an IDL string, an error might occur on the receiving side.

If the maximum string length is specified in IDL, because the length of the termination code `\0` is not included in the maximum length, secure the `byte[]` size that is one larger than the maximum length.

(c) Null in EJB

Do not use a null object for `class/byte[]` in the return value returned from Java.

(3) Generating a reference to a stringified object

This section describes the procedure for obtaining an IOR string file for connection from an ORB client to an ORB gateway.

1. Start an ORB gateway by executing the `ctmstartgw` command with the `-CTMIDLConnect` option specified.
2. Deploy and then start the J2EE application that is to be invoked from the ORB client.
3. Execute the `ctmgetior` command to obtain an IOR string file.

If you re-generate a file that contains IOR strings, note the following:

If you do not change the EJB remote interface, the name registered in the CORBA Naming Service, the server IP address, or the receive port number of the ORB gateway:

You do not need to re-generate the file.

If you start the ORB gateway by executing the `ctmstartgw` command with `-CTMIDLConnect 0` specified, although you cannot obtain IOR strings by using the `ctmgetior` command, you can shorten the times required to start the application and ORB gateway.

If you change the EJB remote interface, the name registered in the CORBA Naming Service, the server IP address, or the receive port number of the ORB gateway:

Re-perform the procedure for generating a reference to a stringified object from step 1.

9.5 Troubleshooting when a CORBA system exception occurs

This section describes the troubleshooting that you can perform when a CORBA system exception occurs.

9.5.1 Minor code in a CORBA system exception

An exception that occurs on the J2EE server, CTM daemon, or ORB gateway is converted into a CORBA system exception, which is then reported to the ORB client. At this time, a minor code in the range from 1213473792 to 1213474047 (from 0x48542400 to 0x485424FF) is set.

If a minor code is set in a CORBA system exception, determine the cause of the error and take appropriate action by referring to the following table.

Table 9–10: Minor code set by CTM

Minor code	Cause of the error	Action
1213473792	An error occurred in CTM.	Collect troubleshooting information and then contact the maintenance personnel, providing them with the collected information and the information about the situation in which the error occurred.
From 1213473793 to 1213474042	An error occurred in CTM.	Take action by referring to *9.5.2 Troubleshooting in the event of an error in CTM*.
1213474046	An error occurred in CTM.	Collect troubleshooting information and then contact the maintenance personnel, providing them with the collected information and the information about the situation in which the error occurred.
1213474047	A `RemoteException` was thrown by the J2EE server.	Determine the reason why the J2EE server threw a `RemoteException`.

9.5.2 Troubleshooting in the event of an error in CTM

If a value in the range from 1213473793 to 1213474042 is set as the minor code in a CORBA system exception, you can determine the cause of the error that occurred in CTM from the exception log file output to the CTMSPOOL directory.

(1) Format of the exception log file

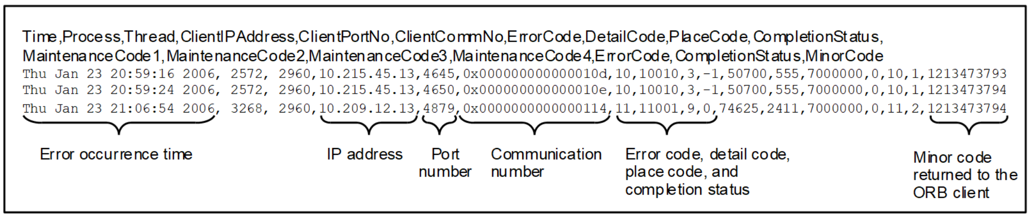
The exception log will store information obtained each time an error occurs. Each record that is stored is a comma-separated list consisting of 17 fields.

Field	Item obtained	Explanation
1	Time	Error occurrence time
2	Process	Process ID
3	Thread	Thread ID
4	ClientIPAddress	IP address of the ORB client
5	ClientPortNo	Port number of the ORB client
6	ClientCommNo	Communication number

Field	Item obtained	Explanation
7	ErrorCode	Error code
8	DetailCode	Detail code
9	PlaceCode	Place code
10	CompletionStatus	Completion status
11	MaintenanceCode1	Maintenance information
12	MaintenanceCode2	Maintenance information
13	MaintenanceCode3	Maintenance information
14	MaintenanceCode2	Maintenance information
15	ErrorCode	Error code returned to the ORB client
16	CompletionStatus	Completion status returned to the ORB client
17	MinorCode	Minor code returned to the ORB client

The following figure shows an example of data output to the exception log.

Figure 9–11: Example of data output to the exception log



(2) Troubleshooting procedure

1. In the exception log file, check whether minor codes received by the ORB client have been output to field 17.

   The location of the exception log file is as follows:

   In Windows:

   CTMSPOOL-directory`\ejb\`*CTMID-name-of-connection-destination*`\expt\`

   In UNIX:

   CTMSPOOL-directory`/ejb/`*CTMID-name-of-connection-destination*`/expt/`

2. If minor codes have been output, find the one on the line where the error occurrence time (field 1) is the same as the time when the error occurred. Then, confirm that the IP address (field 4) and port number (field 5) are the same as those for the ORB client.

3. On the line you found, check the error code (field 7), detail code (field 8), place code (field 9), and completion status (field 10). Then, by referring to *19.4 Details of KFCT messages* in the manual *uCosminexus Application Server Messages*, take appropriate action based on the information you checked.

4. On the line you found, check the communication number (field 6). Then, use the PRF trace to find the last point at which the request was processed to narrow down the location where the problem occurred.

Appendixes

A. Main Updates in the Functionality of Each Version

This section describes the updates in the main functionality for versions of Application Server earlier than 11-00 and the purpose of updates. For details on the main updates in the functionality of 11-00, see *1.4 Main updates in the functionality of Application Server 11-00*.

The description is as follows:

- This section gives an overview and describes the main updates in the functionality of each version of Application Server. For details on the functionality, you check the description in the *Reference location* column corresponding to the *Reference manual* column. The *Reference manual* and *Reference location* columns describe the main locations in the manuals of *11-00* used for this functionality.
- *uCosminexus Application Server* is omitted from the manual names mentioned in the *Reference* column.

A.1 Main updates in the functionality of 09-87

(1) Supporting standard and existing functionality

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

Table A–1: Changes made for supporting standard and existing functionality

Item	Overview of changes	Reference manual	Reference location
Java SE 11 support	The Java SE 11 functionality can now be used.	*Maintenance and Migration Guide*	*Chapter 9*

A.2 Main updates in the functionality of 09-80

(1) Supporting standard and existing functionality

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

Table A–2: Changes made for supporting standard and existing functionality

Item	Overview of changes	Reference manual	Reference location
Lambda expression support in the JAX-RS function	Lambda expressions can now be used with the package specified for the servlet initialization parameter in the `web.xml` file and with the classes included in the subpackages of that package.	*Web Service Development Guide*	*11.2*
Java SE 9 support	The Java SE 9 functionality can now be used.	*Maintenance and Migration Guide*	*Chapter 9*

(2) Maintaining and improving reliability

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

Table A–3: Changes made for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference location
Apache 2.4 support for the web server	Apache 2.4 is supported as the base component of the web server.	*HTTP Server User Guide*	*Chapter 6, Appendix G*
Use of elliptic curve cryptography in SSL communication	Elliptic curve cryptography can now be used in SSL communication.	*HTTP Server User Guide*	*Chapter 5, Appendix G*
Change of the SSL library	The SSL library that provides SSL functions was changed to OpenSSL.	*HTTP Server User Guide*	*Chapter 5, Appendix G*

A.3 Main updates in the functionality of 09-70

(1) Supporting standard and existing functionality

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

Table A–4: Changes made for supporting standard and existing functionality

Item	Overview of changes	Reference manual	Reference location
Addition of JSP compilation methods in the management portal	The compilation methods compliant with JDK 1.7 specifications and with JDK 7 specifications were additionally supported for servlets generated from JSP files on the J2EE server.	*Management Portal User Guide*	*10.8.4*
		Definition Reference Guide	*4.11.2*
Metaspace support in JDK 8	The options for the Permanent area used for Java VM startup were changed to the options for the Metaspace area.	*System Setup and Operation Guide*	*Appendix A.2*
		Management Portal User Guide	*10.8.7*
		Definition Reference Guide	*5.2.1, 5.2.2, 8.2.3*
SHA-2 support for user authentication in the integrated user management	The following hash algorithms for user authentication in the integrated user management were additionally supported: SHA-224, SHA-256, SHA-384, SHA-512	*Security Management Guide*	*5.3.1, 5.3.9, 5.10.7, 11.4.3, 12.4.3, 12.5.3, 13.2, 14.2.2*
Addition of automatic start, restart, and stop in Red Hat Enterprise Linux Server 7	Automatic start, restart, and stop were supported by Management Server and Administration Agent that operate on Red Hat Enterprise Linux Server 7.	*Operation, Monitoring, and Linkage Guide*	*2.6.3, 2.6.4, 2.6.5*
		Command Reference Guide	*7.2*

(2) Maintaining and improving operation performance

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

Table A–5: Changes made for maintaining and improving operation performance

Item	Overview of changes	Reference manual	Reference location
Upgrade to version 9.7	A procedure for changing the options for the Permanent area used for Java VM startup to the options for the Metaspace area was added.	*Maintenance and Migration Guide*	*10.3.1*, *10.3.2*, *10.3.4*

(3) Other purposes

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

Table A–6: Changes made for other purposes

Item	Overview of changes	Reference manual	Reference location
Collection-target data for the snapshot log	Java VM event log data and Management Server thread dumps were added to the collection targets for the snapshot log.	*Maintenance and Migration Guide*	*Appendix A.2*

A.4 Main updates in the functionality of 09-60

(1) Supporting standard and existing functionality

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

Table A–7: Changes made for supporting standard and existing functionality

Item	Overview of changes	Reference manual	Reference location
G1 GC support	G1 GC can now be selected.	*System Design Guide*	*7.15*
		Definition Reference Guide	*14.5*
Support for the object-pointer compression function	The object-pointer compression function can now be used.	*Maintenance and Migration Guide*	*9.18*

(2) Maintaining and improving reliability

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

Table A–8: Changes made for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference location
Addition of the finalize-retention resolution function	A function that resolves a problem that delays execution of many finalize processes was added to prevent occurrence of a delay in releasing OS resources.	*Maintenance and Migration Guide*	*9.16*

(3) Other purposes

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

Table A–9: Changes made for other purposes

Item	Overview of changes	Reference manual	Reference location
Addition of asynchronous output of log files	Log files can now be output asynchronously.	*Definition Reference Guide*	*14.2*

A.5 Main updates in the functionality of 09-50

(1) Improvement in development productivity

The following table describes the items that are changed for improving the development productivity.

Table A–10: Changes made for improving development productivity

Item	Overview of changes	Reference manual	Reference location
Simplification of Eclipse setup	An Eclipse environment can now be set up from the GUI.	*Application Development Guide*	*1.1.5*, *2.4*
Aid for debugging with user-extended performance analysis traces	A user-extended performance analysis trace configuration file can now be created in the development environment.	*Application Development Guide*	*1.1.3*, *6.4*

(2) Simplifying implementation and setup

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

Table A–11: Changes made for simplifying implementation and setup

Item	Overview of changes	Reference manual	Reference location
Addition of system configurations in a virtualization environment	Additional tier types (`http-tier`, `j2ee-tier`, and `ctm-tier`) can now be used in a virtualization environment. As a result, systems can now be built in the following configurations: • Configuration in which a web server and J2EE server are deployed on different hosts • Configuration in which the front-end (servlet and JSP) and back-end (EJB) are separately deployed • Configuration that uses CTM	*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 the standard and existing functionality.

Table A–12: Changes made for supporting standard and existing functionality

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

Item	Overview of changes	Reference manual	Reference location
Relaxation of the naming rules for Portable Global JNDI names	Characters that can be used in Portable Global JNDI names were added.	*Common Container Functionality Guide*	*2.4.3*
Support for the Servlet 3.0 specifications	HTTP Cookie names and URL path parameter names can now be changed in Servlet 2.5 or earlier as they can be changed in Servlet 3.0.	*Web Container Functionality Guide*	*2.7*
Expansion of applications that can link with Bean Validation	Bean Validation can now be used with the CDI or a user application to perform validation.	*Common Container Functionality Guide*	*Chapter 9*
JavaMail support	An email sending and receiving function that uses an API compliant with JavaMail 1.4 can now be used.	*Common Container Functionality Guide*	*Chapter 7*
OSs supported by the `javacore` command	The `javacore` command can now be used to obtain thread dumps in Windows.	*Command Reference Guide*	*javacore (Acquiring the thread dump/in Windows)*

(4) Maintaining and improving reliability

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

Table A–13: Changes made for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference location
Prevention of depletion of the code cache area	Depletion of the code cache area can now be prevented by checking the size of the area used by the system and changing the threshold value before depletion occurs.	*System Design Guide*	*7.2.6*
		Maintenance and Migration Guide	*5.7.2*, *5.7.3*
		Definition Reference Guide	*14.1*, *14.2*, *14.4*
Efficient use of the Explicit Memory Management functionality	The following functions were added that can control the objects that move to the Explicit heap so that the time required for automatic release is reduced to efficiently apply the Explicit Memory Management functionality: • Function that controls object movement to Explicit memory blocks • Function that specifies classes to which the Explicit Memory Management functionality is not applied • Function that outputs the object release rate information as Explicit heap information	*System Design Guide*	*7.14.6*
		This manual	*7.2.2*, *7.6.5*, *7.10*, *7.13.1*, *7.13.3*
		Maintenance and Migration Guide	*5.5*
Expansion of the output range of class-wise statistical information	An extended thread dump containing class-wise statistical information can now include reference relationships based on static fields.	*Maintenance and Migration Guide*	*9.6*

(5) Maintaining and improving the operation performance

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

Table A–14: Changes made for maintaining and improving operation performance

Item	Overview of changes	Reference manual	Reference location
Support for the EADs session failover functionality	The EADs session failover functionality, which enables session failover in collaboration with EADs, is supported.	This manual	*Chapter 5*
Operations using WAR files	A WAR application that consists of only WAR files can now be deployed on the J2EE server.	*Web Container Functionality Guide*	*2.2.1*
		Common Container Functionality Guide	*15.9*
		Command Reference Guide	*cjimportwar (Import a WAR application)*
Synchronous start and stop of the management functionality	An option that synchronously starts and stops the management functionality (Management Server and Administration Agent) was added.	*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 setup Management Server)*
Forced release of Explicit memory blocks by the Explicit Memory Management functionality	The `javagc` command can now be used to release Explicit memory blocks at any time.	This manual	*7.6.1*, *7.9*
		Command Reference Guide	*javagc (forcibly perform GC)*

(6) Other purposes

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

Table A–15: Changes made for other purposes

Item	Overview of changes	Reference manual	Reference location
Acquisition of definition information	The `snapshotlog` command (collect snapshot logs) can now be used to collect definition files only.	*Maintenance and Migration Guide*	*2.3*

Item	Overview of changes	Reference manual	Reference location
		Command Reference Guide	*snapshotlog (collect snapshot logs)*
Output of log data for the `cjenvsetup` command	The information about setup operations (using the `cjenvsetup` command) performed by the Component Container administrator is now output to the message log.	*System Setup and Operation Guide*	*4.1.4*
		Maintenance and Migration Guide	*4.20*
		Command Reference Guide	*cjenvsetup (set up Component Container Administrator)*
BIG-IP v11 support	BIG-IP v11 was added as a type of load balancer that can be used.	*System Setup and Operation Guide*	*4.7.2*
		Virtual System Setup and Operation Guide	*2.1*
Output of the CPU time to the event log for the Explicit Memory Management functionality	The CPU time required to release Explicit memory blocks is now output to the event log for the Explicit Memory Management functionality.	*Maintenance and Migration Guide*	*5.11.3*
Extension of the user-extended performance analysis trace functionality	The following changes were made to the user-extended performance analysis trace functionality: • Although the trace target could be specified by only method, it can now also be specified by package or class. • The range of applicable event IDs was expanded. • The maximum number of lines that can be contained in the user-extended performance analysis trace configuration file was increased. • The user-extended performance analysis trace configuration file can now be used to specify the trace collection level.	*Maintenance and Migration Guide*	*7.5.2, 7.5.3, 8.23.1*
Improvement of information analysis in cases where asynchronous Session Bean invocations are used	The requests of invocation source and destination can now be matched by using the root application information of the PRF trace.	*EJB Container Functionality Guide*	*2.17.3*

A.6 Main updates in the functionality of 09-00

(1) Simplifying implementation and setup

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

Table A–16: Changes made for simplifying implementation and setup

Item	Overview of changes	Reference manual	Reference location
Changing the operation unit used for setup and operation in virtual environment	The operation unit used for setup and operation in the virtual environment has been changed from a virtual server to a virtual server group. You can now perform batch registration of multiple virtual servers to management unit by using a file, which defines the information of virtual server group.	*Virtual System Setup and Operation Guide*	*1.1.2*
Releasing restrictions on environment setup by using setup wizard	The restrictions on the environment that you can set up by using Setup Wizard have been released. Now, you can unset up even an environment that is set up with other functionality and set up the environment by using Setup Wizard.	*System Setup and Operation Guide*	*2.2.7*
Simplifying deletion procedure of the setup environment	The deletion procedure has been simplied by adding a functionality (`mngunsetup` command) that deletes the system environment, which is set up by using Management Server.	*System Setup and Operation Guide*	*4.1.37*
		Management Portal User Guide	*3.6, 5.4*
		Command Reference Guide	*mngunsetup (deleting setup environment of Management Server)*

(2) Supporting standard and existing functionality

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

Table A–17: Changes made for supporting standard and existing functionality

Item	Overview of changes	Reference manual	Reference location
Supporting Servlet 3.0	Servlet 3.0 is now supported.	*Web Container Functionality Guide*	*Chapter 7*
Supporting EJB 3.1	EJB 3.1 is now supported.	*EJB Container Functionality Guide*	*Chapter 2*
Supporting JSF 2.1	JSF 2.1 is now supported.	*Web Container Functionality Guide*	*Chapter 3*
Supporting JSTL 1.2	JSTL 1.2 is now supported.	*Web Container Functionality Guide*	*Chapter 3*
Supporting CDI 1.0	CDI 1.0 is now supported.	*Common Container Functionality Guide*	*Chapter 8*
Using the Portable Global JNDI name	You can now perform the lookup of objects by using the Portable Global JNDI name.	*Common Container Functionality Guide*	*2.4*
Supporting JAX-WS 2.2	JAX-WS 2.2 is now supported.	*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,*

Item	Overview of changes	Reference manual	Reference location
			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*
Supported JAX-RS 1.1	JAX-RS 1.1 is now supported.	*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 A–18: Changes made for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference location
Using TLSv1.2 in SSL/TLS communication	You can now perform SSL/TLS communication in the security protocols including TLSv1.2 by using RSA BSAFE SSL-J.	--	--

Legend:
--: The function is deleted in 09-70.

(4) Maintaining and improving the operation performance

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

Table A–19: Changes made for maintaining and improving operation performance

Item	Overview of changes	Reference manual	Reference location
Monitoring total pending queues in the entire web container	You can now output the pending queues in the entire web container to statistics and monitoring the pending queues.	*Operation, Monitoring, and Linkage Guide*	*Chapter 3*
Outputting performance analysis trace of an application (user extension trace)	You can now output the performance analysis trace used for analyzing the processing performance of applications	*Maintenance and Migration Guide*	*Chapter 7*

Item	Overview of changes	Reference manual	Reference location
	developed by a user, without making changes to the applications.		
Operations in a virtual environment that use user script	You can now execute scripts created by a user (user script) on the virtual server at any time.	*Virtual System Setup and Operation Guide*	*7.8*
Improving the management portal	Changes have been made to display the messages that describe procedures, on the following management portal windows: • Deploying the Preference information window • Startup window for the web server, J2EE server, and SFO server • Package start, package restart and startup window of the web server cluster, and J2EE server cluster	*Management Portal User Guide*	*10.10.1*, *11.9.2*, *11.10.2*, *11.10.4*, *11.10.6*, *11.11.2*, *11.12.2*, *11.12.4*, *11.12.6*
Adding restart functionality of operation management functionality	You can now set automatic restart with the operation management functionality (Management Server and Administration Agent). You can also continue an operation even if the operation management functionality fails. Also the method of setting automatic start has been changed.	*Operation, Monitoring, and Linkage Guide*	*2.4.1*, *2.4.2*, *2.6.3*, *2.6.4*
		Command Reference Guide	*mngautorun (setting up and unsetting up automatic start and automatic restart)*

(5) Other purposes

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

Table A–20: Changes made for other purposes

Item	Overview of changes	Reference manual	Reference location
Changing the file switching unit at the time of log output	You can now execute the date-wise switching of files at the output destination when you output the log.	*Maintenance and Migration Guide*	*3.2.1*
Changing the name of the web server	A name of the web server included in Application Server has been changed to HTTP Server	*HTTP Server User Guide*	--
Supporting direct connection that uses API (SOAP architecture) of BIG-IP	A direct connection that uses API (SOAP architecture) of BIG-IP (load balancer) is now supported. Also the method of setting the connection environment of a load balancer when using a direct connection that uses API has been changed.	*System Setup and Operation Guide*	*4.7.3*, *Appendix J*
		Virtual System Setup and Operation Guide	*2.1*, *Appendix C*
		Security Management Guide	*8.2*, *8.4*, *8.5*, *8.6*, *18.2.1*, *18.2.2*, *18.2.3*

Legend

--: See the entire manual

A.7 Main updates in the functionality of 08-70

(1) Simplifying implementation and setup

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

Table A–21: Changes made for simplifying implementation and setup

Item	Overview of changes	Reference manual	Reference location
Improving the management portal	You can now specify the property that defines resource adaptor properties (setting contents of the Connector property file) and connection test of the properties on the Management Portal screen. Also, enabled the uploading of the J2EE application (ear files and zip files) to Management Server, on the Management Portal screen.	*First Step Guide*	*3.5*
		Management Portal User Guide	--
Adding the implicit import functionality of the `import` attribute in the `page/tag` directive	You can now use the implicit import functionality of the `import` attribute in the `page/tag` directive.	*Web Container Functionality Guide*	*2.3.7*
Supporting the automation of environment settings for JP1 products in a virtual environment	The environment settings of JP1 products can be automatically specified now, for a virtual server, when specifying the settings of Application Server on the virtual server.	*Virtual System Setup and Operation Guide*	*7.7.2*
Improving the integrated user management functionality	You can now use the JDBC driver of database products to connect to the database, when using the database in a user information repository. A database connection with the JDBC driver of Cosminexus DABroker Library is now unsupported. Enabled the settings related to the integrated user management functionality in the Easy Setup Definition file and on the Management Portal screen. In case of Active Directory, the double byte characters such as Japanese characters with DN are now supported.	*Security Management Guide*	*Chapter 5, 14.2.2*
		Management Portal User Guide	*3.5, 10.8.1*
Expanding setting items of HTTP Server	Enabled the direct settings of the directive (settings of `httpsd.conf`) that defines the operation environment of HTTP Server in the Easy Setup definition file and on the Management Portal screen.	*System Setup and Operation Guide*	*4.1.21*
		Management Portal User Guide	*10.9.1*
		Definition Reference Guide	*4.10*

Legend:

--: Reference the entire manual

(2) Supporting the standard and existing functionality

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

Table A–22: Changes made for supporting the standard and existing functionality

Item	Overview of changes	Reference manual	Reference location
Adding specification items of `ejb-jar.xml`	You can now specify the settings of the class level interceptor and method level interceptor in `ejb-jar.xml`.	*EJB Container Functionality Guide*	*2.15*

Item	Overview of changes	Reference manual	Reference location
Supporting the parallel copy GC	You can now select the parallel copy GC.	*Definition Reference Guide*	*14.5*
Supporting global transactions of `Inbound` resource adapters based on the Connector 1.5 specifications	Enabled the usage of `Transacted Delivery` in resource adapters based on the Connector 1.5 specifications. Now EIS that invokes Message-driven Beans can also participate in global transactions.	*Common Container Functionality Guide*	*3.16.3*
Supporting MHP of a 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 an FTP inbound adapter with 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 A–23: 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, you can now select a mode that does not acquire the lock of the database storing the global session information. Also, enabled the definition of request for a reference without updating the database.	This manual	*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*	*2.5.4*
		Definition Reference Guide	*14.2*
Adding a functionality for reduction in the memory size of the Explicit heap used in HTTP sessions	Added a functionality to inhibit the memory usage of the Explicit heap used in HTTP sessions.	This manual	*7.11*

(4) Maintaining and improving operation performance

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

Table A–24: Changes made for maintaining and improving operation performance

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 a JP1 product, at the time of the JP1 integration.	*Virtual System Setup and Operation Guide*	*1.2.2, Chapter 3, 4, 5 and 6, 7.9*

(5) Other purposes

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

Table A–25: Changes made for other purposes

Item	Overview of changes	Reference manual	Referenc e location
Supporting a direct connection with the load balancer with API (REST architecture)	Supported a direct connection with API (REST architecture) as a method of connecting to the load balancer. Also added ACOS (AX2500) as a type of the 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	*4.2.4*
Supporting a timeout when collecting the snapshot log and improving the collection target	Enabled the end (timeout) processing in the time specified for the collection of snapshot logs. Changed the data collected as primary submitted documents.	*Maintenance and Migration Guide*	*Appendix A*

A.8 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 A–26: Changes made for simplifying implementation and setup

Item	Overview of changes	Reference manual	Referenc e location
Configuring a virtual environment supporting various hypervisors	Enabled the configuration of the Application Server virtual servers that are implemented by using various hypervisors. Also, supported the environment including a mix of multiple hypervisors.	*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 the standard and existing functionality.

Table A–27: Changes made for supporting standard and existing functionality

Item	Overview of changes	Reference manual	Referenc e location
Invocation from OpenTP1 supporting the integration of transactions	Enabled the integration of transactions when invoking Message-driven Beans operating on Application Server from OpenTP1.	*Common Container Functionality Guide*	*Chapter 4*
JavaMail	Enabled the usage 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 7*

(3) Maintenance and improvement of reliability

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

Table A–28: Changes made for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference location
Improving the JavaVM troubleshooting functionality	Enabled the usage of the following functionality as the JavaVM troubleshooting functionality: • Enabled the change in the operations when an `OutOfMemoryError` occurs. • Enabled the settings of the upper limit of the C heap allocated volume when compiling JIT. • Enabled the settings of the upper limit of the number of threads. • Extended the output items of the extended `verbosegc` information.	*Maintenance and Migration Guide*	*Chapter 4*, *5*, *9*

(4) Maintaining and improving operation performance

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

Table A–29: Changes made for maintaining and improving operation performance

Item	Overview of changes	Reference manual	Reference location
Supporting JP1/ITRM	Supported JP1/ITRM that is a product to centrally manage the IT resources.	*Virtual System Setup and Operation Guide*	*1.3*, *2.1*

(5) Other purposes

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

Table A–30: Changes made for other purposes

Item	Overview of changes	Reference manual	Reference location
Supporting Microsoft IIS 7.0 and Microsoft IS 7.5	Supported Microsoft IIS 7.0 and Microsoft IIS 7.5 as a Web server.	--	--
Supporting HiRDB Version 9 and SQL Server 2008	Supported the following products as a database: • HiRDB Server Version 9 • HiRDB/Developer's Kit Version 9 • HiRDB/Run Time Version 9 • SQL Server 2008 Also, supported SQL Server JDBC Driver as SQL Server 2008 compliant JDBC driver.	*Common Container Functionality Guide*	*Chapter 3*

Legend:

--: Does not support

A.9 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 A–31: Changes made for simplifying implementation and setup

Item	Overview of changes	Reference manual	Reference location
Change in the tags for which you must specify `web.xml` of the Web service provider machine	The specification of the `listener` tag, `servlet` tag, and `servlet-mapping` tag is now changed from `required` to `optional` in `web.xml`, on the Web Service provider machine.	*Definition Reference Guide*	*2.2.3*
Using the network resources of the logical server	Added a functionality to access the network resources and the 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 a sample program	The procedure to execute a sample program is simplified by providing a part of the sample program in the EAR format.	*First Step Guide*	*3.5*
		System Setup and Operation Guide	*Appendix L*
Improving the operations of the Management Portal screen	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 wizard	Enabled the display of the Easy Setup definition file and Connector property file used in setup, on the screen when Setup Wizard is completed.	*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 hypervisors.#	*Virtual System Setup and Operation Guide*	*Chapter 3*, *5*

\#

For setting with the 08-50 mode, see *Appendix D Settings when using the Virtual server manager of the 08-50 mode* in the *uCosminexus Application Server Virtual System Setup and Operation Guide*.

(2) Supporting standard and existing functionality

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

Table A–32: Changes made for supporting standard and existing functionality

Item	Overview of changes	Reference manual	Reference location
Supporting the invocation from OpenTP1	Enabled the invocation of the Message-driven Beans operating on Application Server from OpenTP1.	*Common Container Functionality Guide*	*Chapter 4*
Supporting JMS	Enabled the usage of the Cosminexus JMS provider functionality that is compliant to the JMS1.1 specifications.	*Common Container Functionality Guide*	*Chapter 6*
Supporting Java SE 6	Enabled the usage of the Java SE 6 functionality.	*Maintenance and Migration Guide*	*5.5*, *5.8.1*
Support for using Generics	Enabled the usage of Generics in EJB.	*EJB Container Functionality Guide*	*4.2.18*

(3) Maintaining and improving reliability

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

Table A–33: Changes made for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference location
Improving the usability of the Explicit Memory Management functionality	Enabled the easy usage of the Explicit Memory Management functionality with the automatic allocation setup file.	*System Design Guide*	*7.2*, *7.7.3*, *7.11.4*, *7.12.1*
		Expansion Guide	*Chapter 7*
Disabling the database session failover functionality in URI	When using the database session failover functionality, the request that is not the target of the functionality can now be specified in URI.	This manual	*5.6.1*
Fault monitoring in a virtual environment	Enabled the detection of the fault that occurred, by monitoring the virtual server in a virtual system.	*Virtual System Setup and Operation Guide*	*Appendix D*

(4) Maintaining and improving operation performance

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

Table A–34: Changes made for maintaining and improving operation performance

Item	Overview of changes	Reference manual	Reference location
Omitting the management user account	Enabled the omission of the user login ID and password in the 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 (Starting/Stopping/Setup of Management Server)*, *mngsvrutil(Operations management command of Management Server)*, *8.3*, *cmx_admin_passwd(Settings of the Management user account of Management Server)*
Operations in a virtual environment	Added a procedure to operate batch start and batch stop, scale in and scale out for the multiple servers in a virtual system. #	*Virtual System Setup and Operation Guide*	*Chapter 4*, *6*

#

When you set up in 08-50 mode, see *Appendix D Settings when using the virtual server manager of 08-50 mode* in the *uCosminexus Application Server Virtual System Setup and Operation Guide*.

(5) Other purposes

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

Table A–35: Changes made for other purposes

Item	Overview of changes	Reference manual	Reference location
Functionality to count the unnecessary objects in the Tenured area	Enabled to specify only the unnecessary objects within the Tenured area.	*Maintenance and Migration Guide*	*9.8*

Item	Overview of changes	Reference manual	Referenc e location
Functionality to output the source object list of the Tenured area increase factor	Modified to output the information of the objects that are the source of unnecessary objects specified by using the functionality to count the unnecessary objects in a `Tenured` area.		*9.9*
Functionality to analyze the class wise statistical information	Enabled to output the class wise statistical information in the CSV format.		*9.10*
Cluster node switching according to the automatic restart over number detection of the logical server	Enabled the node switching at the time when the logical server is in an abnormal stop status (when an error is detected if the frequency of automatic restart is over or frequency of automatic restart is set to 0) for a cluster configuration that is monitored for switching Management Server.	*Operation, Monitoring, and Linkage Guide*	*18.4.3*, *18.5.3*, *16.2.2*, *16.3.3*, *16.3.4*
Node switching system for the host unit management model	Enabled the node switching for the host unit management model in the system operations integrated with cluster software.		*Chapter 16*
Supporting ACOS (AX2000 and BS320)	Added ACOS (AX2000 and 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 J*, *Appendix J.2*
		Definition Reference Guide	*4.2.4*, *4.3.2*, *4.3.4*, *4.3.5*, *4.3.6*, *4.7.1*
Adding the transaction property that can be specified in Stateful Session Bean (SessionSynchronization) when managing a transaction in CMT	When managing a transaction in CMT, enabled to specify `Supports`, `NotSupported`, and `Never` as a transaction property in Stateful Session Bean (SessionSynchronization).	*EJB Container Functionality Guide*	*2.7.3*
Terminating Administration Agent when `OutOfMemoryError` occurs	When `OutOfMemoryError` occurs in Java VM, enabled to terminate Administration Agent.	*Maintenance and Migration Guide*	*2.5.5*
Asynchronous parallel processing of threads	Enabled to implement asynchronous timer processing and asynchronous processing of threads by using TimerManager and WorkManager.	This manual	--

A.10 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 A–36: Changes made for improving development productivity

Item	Overview of changes	Reference manual	Reference location
Simplifying migration from other Application Server products	Enabled to use the following functionality for the smooth migration from other Application Server products: • Enabled to judge the upper limit of an HTTP session with an exception. • Enabled to prevent 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.	*Web Container Functionality Guide*	*2.3*, *2.7.5*
Providing `cosminexus.xml`	By coding the attributes unique to Cosminexus Application Server in `cosminexus.xml`, enabled to start a J2EE application without setting up the property, once the J2EE application is imported to the J2EE server.	*Common Container Functionality Guide*	*13.3*

(2) Supporting standard functionality

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

Table A–37: Changes made for supporting standard functionality

Item	Overview of changes	Reference manual	Reference location
Supporting Servlet 2.5	Supported Servlet 2.5.	*Web Container Functionality Guide*	*2.2*, *2.5.4*, *2.6*, *Chapter 7*
Supporting JSP 2.1	Supported JSP 2.1.	*Web Container Functionality Guide*	*2.3.1*, *2.3.3*, *2.5*, *2.6*, *Chapter 7*
JSP debug	Enabled to perform JSP debugging in a development environment using MyEclipse.#	*Web Container Functionality Guide*	*2.4*
Saving the tag library in the library JAR and mapping TLD	When the tag library is saved in the library JAR, enabled to search the TLD files in the library JAR by the Web container when the Web application is running, and then map theTLD files automatically.	*Web Container Functionality Guide*	*2.3.4*
Omitting `application.xml`	Enabled to omit `application.xml`in J2EE applications.	*Common Container Functionality Guide*	*13.4*
Combined use of the annotation and the DD	Enabled to use annotations together with a DD, and update the contents specified in the annotation in the DD.	*Common Container Functionality Guide*	*14.5*
Compliance of annotations with the Java EE 5 standard (default interceptor)	Enabled to save the default interceptor in the library JAR. Also, enabled to perform DIfrom the default interceptor.	*Common Container Functionality Guide*	*13.4*
Reference resolution of @Resource	Enabled to perform the reference resolution of a resource with @Resource.	*Common Container Functionality Guide*	*14.4*
Supporting JPA	Supported the JPA specifications.	*Common Container Functionality Guide*	*Chapter 5*

#

In 09-00 or later, you can use the JSP debug functionality in a development environment with WTP.

(3) Maintenance and improvement of reliability

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

Table A–38: Changes made for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference location
Persistence of the session information	Enabled to save and inherit the session information of an HTTP session in the database.	This manual	*Chapter 5*, *Chapter 6*
Suppression of Full GC	Occurrence of Full GC can now be suppressed by placing objects that can trigger Full GC outside the Java heap.	This manual	*Chapter 7*
Client performance monitor	Enabled to check and analyze the time consumed in a client processing.	--	--

Legend:

--: The function is deleted in 09-00.

(4) Maintaining and improving operation performance

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

Table A–39: Changes made for maintaining and improving operation performance

Item	Overview of changes	Reference manual	Reference location
Improving the operation performance of applications in the management portal	For the application and resource operations, enabled to perform mutual operations of the server management commands and the management portal.	*Management Portal User Guide*	*1.1.3*

(5) Other purposes

The following table describes the items changed for other purposes.

Table A–40: Changes made for other purposes

Item	Overview of changes	Reference manual	Reference location
Deleting the disabled HTTP cookie	Enabled to delete the disabled HTTP cookie.	*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	Enabled to specify the timeout period for a connection failure detection timeout.	*Common Container Functionality Guide*	*3.15.1*
Supporting Oracle11g	Enabled the usage of Oracle11g as a database.	*Common Container Functionality Guide*	*Chapter 3*
Scheduling batch processing	Enabled to schedule the execution of batch applications with CTM.	This manual	*Chapter 4*
Batch processing log	Enabled to specify 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).	*Definition Reference Guide*	*3.2.5*

Item	Overview of changes	Reference manual	Referenc e location
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 the 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*
Selecting whether to perform GC before outputting statistics	Whether to perform GC can now be selected before outputting class-wise statistical information.	*Maintenance and Migration Guide*	*9.7*
Functionality for the output of the age distribution information of the Survivor area	Enabled to output the age distribution information of Java objects of the Survivor area in JavaVM log file.	*Maintenance and Migration Guide*	*9.11*
Functionality for eliminating the finalize stagnation	Enabled to eliminate the stagnation of the finalize processing of JavaVM by monitoring the status of the processing.	--	--
Changing the maximum heap size of the server management commands	The maximum heap size used by the server management commands has been changed.	*Definition Reference Guide*	*5.2.1*, *5.2.2*
Supporting when a display name that is not recommended is specified	Enabled to output a message when a display name that is not recommended in J2EE applications is specified	*Messages*	*KDJE4237 4-W*

Legend

--: The functionality has been deleted in version 09-00.

B. 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

Symbols

A

B

C

D

E

F

Q

R

S

T

U

V

W

Y