HITACHI
Inspire the Next

uCosminexus Application Server

Web Container Functionality Guide

3020-3-Y05-10(E)

■ Relevant program products

See the manual *uCosminexus Application Server Overview*.

■ Export restrictions

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

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

■ Trademarks

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

AX2000 is a product name of A10 Networks, Inc.

BIG-IP, 3-DNS, iControl Services Manager, FirePass, and F5 are either trademarks or registered trademarks of F5 Networks, Inc.

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

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

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

gzip is a freeware of American FSF (Free Software Foundation).

IIOP is a network protocol name for communication between ORB (Object Request Broker) according to OMG specifications.

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

MyEclipse is a trademark of Genuitec Corporation in the United States.

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

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

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

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

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

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

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

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

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

Other product and company names mentioned in this document may be either trademarks or registered trademarks of their respective companies.

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

Throughout this document Hitachi has attempted to distinguish trademarks from descriptive terms by writing the name with the capitalization used by the manufacturer, or by writing the name with initial capital letters. Hitachi cannot attest to the accuracy of this information. Use of a trademark in this document should not be regarded as affecting the validity of the trademark.

■ Microsoft product screen shots

Microsoft product screen shots reprinted with permission from Microsoft Corporation.

■ Microsoft product name abbreviations

This manual uses the following abbreviations for Microsoft product names:

Abbreviation			Full name or meaning
Active Directory			Microsoft(R) Active Directory(R)
Microsoft IIS	Microsoft IIS 7.0		Microsoft(R) Internet Information Services 7.0
	Microsoft IIS 7.5		Microsoft(R) Internet Information Services 7.5
Windows	Windows Server 2008	Windows Server 2008 x86	Microsoft(R) Windows Server(R) 2008 Standard 32-bit
			Microsoft(R) Windows Server(R) 2008 Enterprise 32-bit

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

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

■ Restrictions

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

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

Printed in Japan.

■ Issued

Aug. 2013: 3020-3-Y05-10(E)

■ Copyright

Summary of amendments

The following table lists changes in the manual 3020-3-Y05-10(E) for uCosminexus Application Server 09-50, uCosminexus Application Server(64) 09-50, uCosminexus Client 09-50, uCosminexus Developer 09-50, uCosminexus Service Architect 09-50, uCosminexus Service Platform 09-50, and uCosminexus Service Platform(64) 09-50 and product changes related to the manual:

Additions and Changes	Change Location
The functions for deploying WAR applications as J2EE applications were added.	*2.2.1*
The functions for changing the HTTP Cookie name and the URL parameter name in the version prior to Servlet 2.5, by conforming to the standard specifications of Servlet 3.0 were added.	*2.7.3*, *2.7.6*, *2.7.7*, *2.7.8*, *2.7.9*
The method dependent restrictions in the case of reloading an application in which the functions of Servlet 3.0 specifications are used, were deleted.	*6.2.3*
The description on notes was moved from release notes.	*2.24*, *3.3.1*, *5.2.1*, *6.2.1*, *6.2.2*, *6.2.6*

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

For this version (3020-3-Y05-10(E)), some of the content in the previous version (3020-3-Y05(E)) has been moved to the *uCosminexus Application Server Common Container Functionality Guide* and the structure of the contents have changed. The following table shows the correspondence with the previous version.

Old (3020-3-Y05(E))	New (3020-3-Y05-10(E))
Bean Validation functions and Bean Validation operations	Moved to the *uCosminexus Application Server Common Container Functionality Guide*.
The sequence of using Bean Validation from JSF	
Using the log for debug (development check log)	

Preface

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

■ Non-supported functionality

Some functionality described in this manual are not supported. The non-supported functionality include:

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

■ Non-supported compatibility functionality

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

- Basic mode
- Check of JSP source compliance (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 purpose of functionality of Application Server and the manuals corresponding to each functionality. This chapter also describes the functionality that was changed in this version.

1.1 Classification of functionality

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

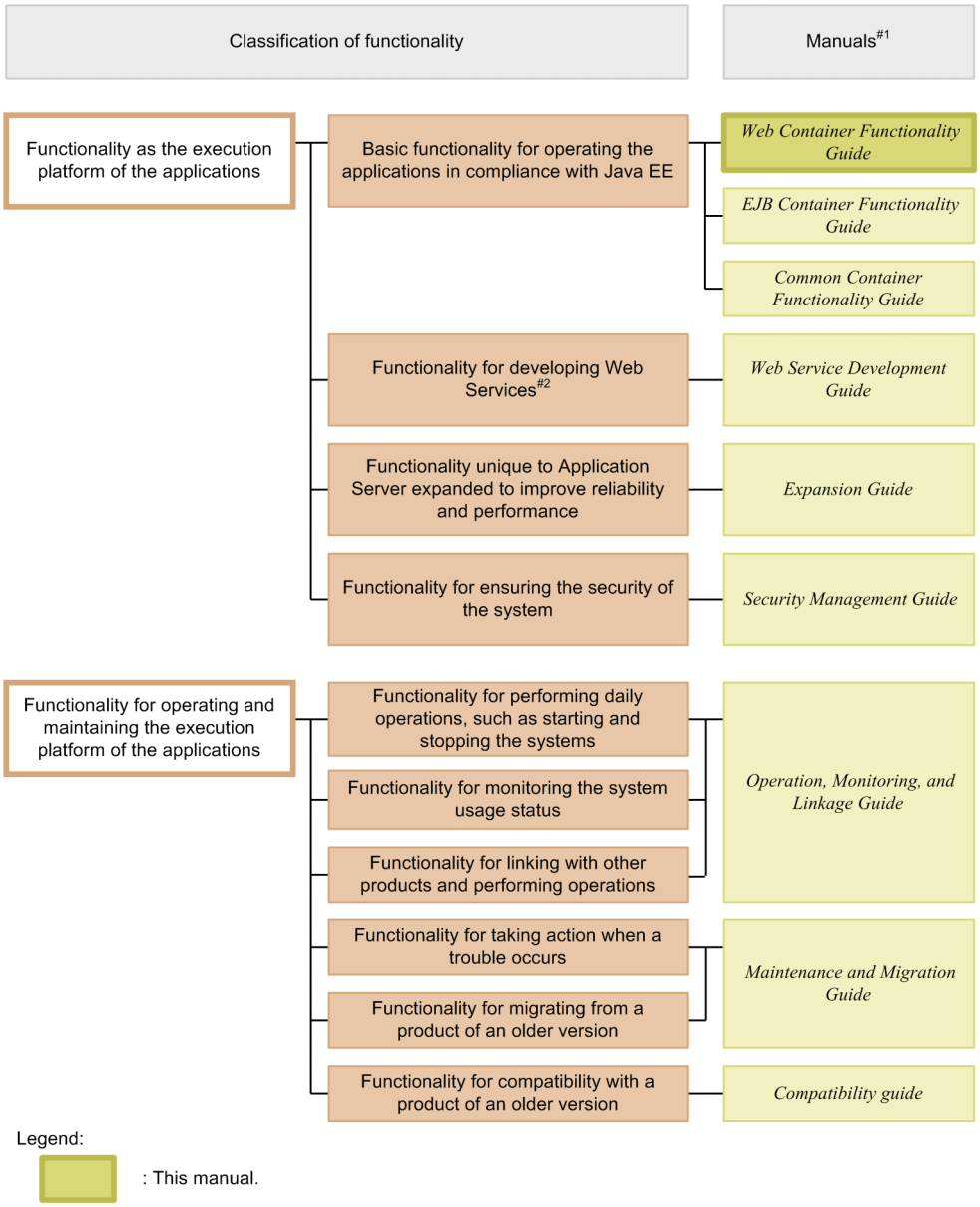
The following are the broad classifications of the Application Server functionality:

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

The above-functionality can be further classified according to the positioning and the intended usage of the functionality. 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 the functionality (functionality guides).

Figure 1－1: Classification of Application Server functionality and the set of manuals corresponding to functionality (functionality guides)



#1

uCosminexus Application Server has been omitted from the manual names.

#2

You can execute SOAP Web Services and RESTful Web Services with Application Server. See the following manuals other than the *uCosminexus Application Server Web Service Development Guide*, depending on your purpose of execution.

When developing and executing SOAP applications

- *uCosminexus Application Server SOAP Application Development Guide*

When ensuring the security of SOAP Web Services or SOAP applications

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

For details about the XML process

- *uCosminexus Application Server XML Processor User Guide*

The following subsections describe the classifications of the functionality as well as the manuals corresponding to the functionality.

1.1.1 Functionality that serves as an execution platform for the applications

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

Before you perform the system building or application development, you must first determine whether you want to use functionality that serves as the execution platform for the applications.

The following are the classification-wise descriptions of functionality that serve as the application execution platform:

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

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

Application Server provides a Java EE 6-compliant J2EE server. The J2EE server provides functionality that is compliant with the standard specifications and is independent of Application Server.

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

The following is an overview of each classification:

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

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

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

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

(2) Functionality for developing Web Services

This classification includes the functionality that can be used as the execution environment and development environment for Web Services.

Application Server provides the following engines:

- The JAX-WS engine that performs binding of SOAP messages in accordance with the JAX-WS specifications
- The JAX-RS engine that performs the binding of RESTful HTTP messages in accordance with the JAX-RS specifications

(3) Application Server independent functionality extended for improving reliability and performance (extended functionality)

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

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

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

The functionalities that ensure system security with a focus on Application Server fall into this category. This includes functionalities such as the authentication functionality to prevent access by unauthorized users and encryption functionality to prevent information leakage in communication paths.

1.1.2 Functionality for operating and maintaining the application execution platform

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

The following are the classification-wise descriptions of functionality used for operating and maintaining the application execution platform:

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

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

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

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

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

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

(4) Functionality for troubleshooting (maintenance functionality)

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

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

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

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

This includes functionality used for the compatibility with older versions of Application Server. For compatibility functionality, Hitachi encourages migrating to the corresponding recommended functionality.

1.1.3 Functionality and corresponding manuals

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

The following table describes the classifications of functionality and the manuals corresponding to each functionality:

Table 1－1: Classifications of functionality and corresponding manuals having functionality description

Category	Functionality	Manuals[#1]
Basic development functionality	Web container	*Web Container Functionality Guide*[#2]

Category	Functionality	Manuals[#1]
Basic development functionality	Using JSF and JSTL	*Web Container Functionality Guide*[#2]
	Web server linkage	
	In-process HTTP server	
	Servlet and JSP implementation	
	EJB container	*EJB Container Functionality Guide*
	EJB client	
	Precautions during Enterprise Bean implementation	
	Naming management	*Common Container Functionality Guide*
	Resource connections and transaction management	
	Invoking Application Server from OpenTP1 (TPI inbound integrated function)	
	JPA usage on Application Server	
	Cosminexus JPA provider	
	Cosminexus JMS provider	
	Usage of JavaMail	
	Using CDI with Application Server	
	Using Bean Validation with Application Server	
	Application attribute management	
	Using annotations	
	Formatting and deploying J2EE applications	
	Container extension library	
Extended functionality	Executing applications using the batch server	*Expansion Guide*
	Scheduling and load balancing requests using CTM	
	Scheduling the batch applications	
	Inheriting the session information between the J2EE servers (Session failover functionality)	
	Database session failover functionality	
	EADs session failover functionality	
	Controlling full garbage collection using the explicit heap functionality	
	Output of the application user log	
	Asynchronous parallel processing of threads	
Security management functionality	Authentication using integrated user management	*Security Management Guide*
	Authentication using application settings	
	Using TLSv1.2 for the SSL/TLS communication	
	Controlling with the operation management functionality of load balancers that use API based direct connections.	

Category	Functionality	Manuals[#1]
Operation functionality	Starting and stopping the system	*Operation, Monitoring, and Linkage Guide*
	Managing J2EE applications	
Watch functionality	Monitoring the operation information (Statistics collection functionality)	
	Monitoring resource depletion	
	Audit log output functionality	
	Database audit trail integration functionality	
	Output of operation information using the management commands	
	Automatic execution of processing by using management event notification and management action	
	Collecting statistical information of CTM	
	Output of the console log	
Linkage functionality	Operating a JP1 integrated system	
	Centralized monitoring of system (Linking with JP1/IM)	
	Automatic running of system by jobs (Linking with JP1/AJS)	
	Collection and consolidated management of audit log (Linking with JP1/Audit Management - Manager)	
	Linking with cluster software	
	1-to-1 node switching system (linking with cluster software)	
	Mutual node switching system (linking with cluster software)	
	Node switching system (integrating with cluster software) for host unit management models.	
Maintenance functionality	Troubleshooting related functionality	*Maintenance and Migration Guide*
	Performance analysis where performance analysis trace is used	
	Hitachi-specific JavaVM (hereafter also abbreviated as JavaVM) functionality	
Migration functionality	Migrating from an older version of Application Server	
	Migrating to a recommended functionality	
Compatibility functionality	Basic mode	*Compatibility Guide*
	Servlet engine mode	
	Compatibility functionality of the basic development functionality	
	Compatibility functionality of the expansion functionality	

#1 *uCosminexus Application Server* has been omitted from the manual name

#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 the types of systems in which the functionality for executing the Web applications can be used. The functionality-wise support for the following items are described here:

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

The functionality for executing Web applications includes the Java EE standard functionality and is the functionality independently extended on Application Server. When you choose the functionality, also confirm the compliance with Java EE standards, as and when required.

1.2.1 Web Container Functionality

The table below describes the functionality of a Web container. Select the functionality according to the purpose of the system. For details on the functionality, see the reference sections.

Table 1－2: Web Container functionality and corresponding purpose of the system

Functionality	Purpose of the system					Compliance with Java EE standards		Refer-ence
	Reliability	Performance	Operation and maintenance	Extendibility	Others	Standard	Extended	
Web application execution functionality	--	--	--	--	--	Y	Y	*2.2*
JSP execution functionality	--	--	--	--	--	Y	Y	*2.3*
JSP debug functionality	--	--	--	--	Y	Y	Y	*2.4*
JSP pre-compilation functionality and maintaining compilation results	--	Y	--	--	--	--	Y	*2.5*
Functionality for setting up the default character encoding	--	--	Y	--	--	Y	Y	*2.6*

Functionality	Purpose of the system					Compliance with Java EE standards		Refer-ence
	Reliability	Performance	Operation and maintenance	Extendibility	Others	Standard	Extended	
Session management functionality	Y	--	--	Y	--	Y	Y	*2.7*
Event listener of an application	--	--	--	--	Y	Y	--	*2.8*
Filtering requests and responses	--	--	Y	--	--	Y	Y	*2.9*
HTTP response compression functionality	--	Y	--	--	--	--	Y	*2.10*
Integrating with an EJB container	--	--	--	--	--	Y	Y	*2.11*
Connecting to the database	--	--	--	Y	--	Y	Y	*2.12*
Creating threads by a Web container	--	Y	--	--	--	--	Y	*2.13*
Using user thread	--	--	--	--	Y	--	Y	*2.14*
Controlling the number of concurrently executing threads	--	Y	--	--	--	--	Y	*2.15* *2.16* *2.17* *2.18* *2.19*
Error page customization	--	--	--	--	Y	--	Y	*2.20*
Caching the static contents	--	Y	--	--	--	--	Y	*2.21*
URI decode functionality	--	--	Y	--	--	Y	--	*2.22*
Version setup functionality of Web applications	--	--	Y	--	--	Y	Y	*2.23*

Legend:

Y: Supported

--: Not supported

Note:

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

1.2.2 JSF and JSTL functionality

The following table describes the JSF and JSTL functionality. Select the functionality that suits the purpose of the system. For details on the functionality, see the *Reference* column.

Table 1-3: JSF and JSTL functionality and the corresponding purpose of the system

Functionality	Purpose of the system					Compliance with Java EE standards		Reference
	Reliability	Performance	Operation and maintenance	Extendibility	Other	Standard	Extended	
JSF (includes functionality that uses Bean Validation)	--	--	--	--	Y#	Y	--	*Chapter 3*
JSTL	--	--	--	--	Y#	Y	--	

Legend:

Y: Supported

--: Not supported

#

These functionalities improve the ease of application development.

1.2.3 Web server integration functionality

The following table describes the functionality of Web server integration. Select the functionality according to the purpose of the system. For details on the functionality, see the reference sections:

Table 1-4: Web server integration functionality and corresponding purpose of the system

Functionality	Purpose of the system					Compliance with Java EE standards		Reference
	Reliability	Performance	Operation and maintenance	Extendibility	Others	Standard	Extended	
Distributing requests with the Web server (redirector)	Y	Y	--	Y	--	--	Y	*4.2* *4.3* *4.4* *4.5*
Communication timeout (Web server integration)	Y	Y	--	--	--	--	Y	*4.6*
Specifying the IP address (Web server integration)	--	--	--	--	Y	--	Y	*4.7*
Error page customization (Web server integration)	--	--	--	--	Y	--	Y	*4.8*
Viewing the top page by specifying the domain name	--	--	--	--	Y	Y	--	*4.9*
Notification of gateway information to a Web container	--	--	--	--	Y	--	Y	*4.10*

Legend:

Y: Supported

--: Not supported

Note:

The functionality for which `Y` is specified in both the 'Standard' and 'Extended' columns of 'Compliance with Java EE standards' column indicates that the functionality unique to Application Server has been extended beyond the Java EE standard

functionality. The functionality for which Y is specified only in the Extended column indicates functionality unique to Application Server.

1.2.4 In-process HTTP server functionality

The following table describes the in-process HTTP server functionality. Select the functionality according to the purpose of the system. For details on the functionality, see the reference section.

Table 1-5: In-process HTTP server functionality and corresponding purpose of the system

Functionality	Purpose of the system					Compliance with Java EE standards		Reference
	Reliability	Performance	Operation and maintenance	Extendibility	Others	Standard	Extended	
Controlling the number of connections from the Web client	--	Y	--	--	--	--	Y	*5.3*
Controlling the number of request processing threads	--	Y	--	--	--	--	Y	*5.4*
Controlling the flow of requests by controlling the number of concurrent connections from the Web client	--	Y	--	--	--	--	Y	*5.5*
Controlling the flow of requests by controlling the number of concurrently executed threads	--	Y	--	--	--	--	Y	*5.6*
Request distribution with the redirector	--	--	--	--	Y	--	Y	*5.7*
Controlling the communication with the Web client by using Persistent Connection	--	Y	--	--	--	--	Y	*5.8*
Communication timeout (in-process HTTP server)	Y	Y	--	--	--	--	Y	*5.9*
Specifying the IP address (in-process HTTP server)	--	--	--	--	Y	--	Y	*5.10*
Controlling access by limiting the hosts that are allowed access	Y	--	--	--	--	--	Y	*5.11*
Controlling access by limiting the request data size	Y	--	--	--	--	--	Y	*5.12*
Controlling access by limiting the HTTP-enabled methods	Y	--	--	--	--	--	Y	*5.13*
Customizing responses to the Web client using HTTP responses	--	--	--	--	Y	--	Y	*5.14*

Functionality	Purpose of the system					Compliance with Java EE standards		Refer-ence
	Reliability	Performance	Operation and maintenance	Extendibility	Others	Standard	Extended	
Error page customization (in-process HTTP server)	--	--	--	--	Y	--	Y	*5.15*
Notification of gateway information to a Web container	--	--	--	--	Y	--	Y	*5.16*
Output of log and trace	--	--	--	--	Y	--	Y	*5.17*

Legend:

Y: Supported

--: Not supported

Note:

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

1.3 Explanation of the functionality described in this manual

This section describes the meaning of the classifications used when describing the functionality in this manual, and also provides an example of the tables used to describe each classification.

1.3.1 Meaning of classifications

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

- Explanation

  This is the explanation about the functionality. This section explains the purpose, features, and mechanism of the functionality. Read this section when you want an overview of the functionality.
- Implementation

  This section describes the methods such as the coding method and the DD writing method. Read this section when developing applications.
- Settings

  This section describes the required property settings for building the system. Read this section when building a system.
- Operations

  This section describes the operation method. This section describes the operating procedures and the execution examples of commands to be used. Read this section when operating a system.
- Notes

  This section describes the general precautions for using the functionality. Make sure that you read the notes.

1.3.2 Examples of tables describing the classification

The following table lists the classification for the description of functionality. The title of the table is "Organization of this Chapter" or " Organization of this Section".

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

Example table describing the classification for the description of functionality

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

Category	Title	Reference
Explanation	What is the *XX* functionality	*X.1*
Implementation	Implementation of applications	*X.2*
	Definitions in the DD and `cosminexus.xml`#	*X.3*
Settings	Settings in the execution environment	*X.4*
Operations	Operations using the *XX* functionality	*X.5*
Notes	Notes on using the *XX* functionality	*X.6*

#

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

Tip

Property settings for applications that do not include cosminexus.xml

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

You specify the application settings in the execution environment using the server management commands and the property files. For details on the application settings using the server management commands and the property files, see *3.5.2 Procedure for setting the properties of a J2EE applications* in the *uCosminexus Application Server Application Setup Guide*.

The tags specified in the property file correspond to the DD or `cosminexus.xml`. For details on the DD or `cosminexus.xml` and the corresponding property file tags, see *2. Cosminexus Application Property File (cosminexus.xml)* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

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

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

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

The contents described in this section are as follows:

- This section gives an overview and describes the main changes in the functionality of Application Server 09-50. For details on the functionality, check the description provided in the corresponding manuals. The `Reference manual` column and `Reference` column indicates the location of the description of a particular functionality.
- *uCosminexus Application Server* is omitted from the manual names mentioned in the Reference manual column.

(1) Improving development productivity

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

Table 1-6: Changes made for improving the development productivity

Item	Overview of changes	Reference manual	Reference
Simplifying the Eclipse setup	The Eclipse environment can now be set up by using a GUI.	*Application Development Guide*	*1.1.5*, *2.4*
Supporting debug by using the user expanded performance analysis trace	The user expanded performance analysis trace setup file can now be created in the development environment.	*Application Development Guide*	*1.1.3*, *6.5*

(2) Simplifying implementation and setup

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

Table 1-7: Changes made for simplifying implementation and setup

Item	Overview of changes	Reference manual	Reference
Expanding the system configuration patterns in the virtual environment	The types of tiers (http-tier, j2ee-tier and ctm-tier) that can be used in the virtual environment have increased. The following system configuration patterns can now be configured in accordance with this: • A pattern of configuring the Web server and J2EE server on separate hosts • A pattern of configuring by separating the front end (servlet, JSP) and backend (EJB) • A pattern of using CTM	*Virtual System Setup and Operation Guide*	*1.1.2*

(3) Supporting standard and existing functionality

The following table describes the items that were changed to support standard and existing functionality:

Table 1-8: Changes made for supporting standard and existing functionality

Item	Overview of changes	Reference manual	Reference
Supporting the JDBC 4.0 specifications	DB Connector now supports the HiRDB Type4 JDBC Driver with JDBC 4.0 specifications and the JDBC Driver of SQL Server.	*Common Container Functionality Guide*	*3.6.3*
Simplifying the naming conventions in the Portable Global JNDI name	The characters that can be used in the Portable Global JNDI name have been added.	*Common Container Functionality Guide*	*2.4.3*

Item	Overview of changes	Reference manual	Reference
Supporting the Servlet 3.0 specifications	You can now use the changed HTTP Cookie name and the URL path parameter name of Servlet 3.0 even in versions earlier than Servlet 2.5.	This manual	*2.7*
Extending the applicability of the applications that can be integrated with Bean Validation	The CDI or user applications can also be verified now by using Bean Validation.	*Common Container Functionality Guide*	*Chapter 10*
Supporting JavaMail	The mail send and receive function that uses JavaMail 1.4 compliant APIs can now be used.	*Common Container Functionality Guide*	*Chapter 8*
Expanding the applicability of OSs on which the `javacore` command can be used	The thread dump of Windows can now be acquired by using the `javacore` command.	*Command Reference Guide*	*javacore (acquiring the thread dump/when using Windows)*

(4) Maintaining and improving reliability

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

Table 1-9: Changes for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference
Avoiding exhaustion of the code cache area	By confirming the size of the code cache area used in the system, and changing the threshold value before the area becomes exhausted, the area exhaustion can now be avoided.	*System Design Guide*	*7.1.2*
		Maintenance and Migration Guide	*5.7.2*, *5.7.3*
		Server Definition Reference Guide	*16.1*, *16.2*, *16.4*
Supporting the effective application of the Explicit management functionality	A functionality that can control the objects to be moved to the Explicit heap is now added as the functionality for effective application of the explicit management function by curtailing the auto-release processing time. • Functionality for controlling the moving of objects to the Explicit memory block • Functionality for specifying the application exclusion class for the explicit management functionality • Output of the object release rate information to the Explicit heap information	*System Design Guide*	*7.13.6*
		Expansion Guide	*8.2.2*, *8.6.5*, *8.10*, *8.13.1*, *8.13.3*
		Maintenance and Migration Guide	*5.5*
Expanding the output range of the class wise statistical information	The reference relation based on the static field can now be output in the expanded thread dump that includes the class wise statistical information.	*Maintenance and Migration Guide*	*9.6*

(5) Maintaining and improving operation performance

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

Table 1-10: Changes for maintaining and improving operation performance

Item	Overview of changes	Reference manual	Reference
Supporting the EADs session failover functionality	The EADs session failover functionality that implements the session failover functionality by integrating with EADs is now supported.	*Expansion Guide*	*Chapter 5*, *Chapter 7*
Operations by WAR	The WAR applications that are configured only by the WAR files can now be deployed on the J2EE server.	This manual	*2.2.1*

Item	Overview of changes	Reference manual	Reference
Operations by WAR	The WAR applications that are configured only by the WAR files can now be deployed on the J2EE server.	*Common Container Functionality Guide*	*13.9*
		Command Reference Guide	*cjimportwar (importing WAR applications)*
Starting and stopping by the concurrent execution of the management functionality	The startup and stopping of the management functionality (Management Server and Administration Agent) has been added to the option for concurrent execution.	*Operation, Monitoring, and Linkage Guide*	*2.6.1*, *2.6.2*, *2.6.3*, *2.6.4*
		Command Reference Guide	*adminagentctl (starting and stopping Administration Agent)*, *mngautorun (setting/cancelling the setting of automatic start and automatic restart)*, *mngsvrctl (starting, stopping and setting up Management Server)*
Forcibly releasing the Explicit memory block in the Explicit management functionality	The Explicit memory block can now be released at any time, by using the `javagc` command.	*Expansion Guide*	*8.6.1*, *8.9*
		Command Reference Guide	*javagc (forcibly generating the garbage collection)*

(6) Other purposes

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

Table 1－11: Changes due to other purposes

Item	Overview of changes	Reference manual	Reference
Acquiring the definition information	Only the definition files can now be collected by using the `snapshot` (collecting the snapshot log) command.	*Maintenance and Migration Guide*	*2.3*
		Command Reference Guide	*snapshotlog (collecting the snapshot log)*
Performing the log output of the `cjenvsetup` command	The execution information of the Component Container administrator setup (`cjenvsetup` command) can now be output in the message log.	*uCosminexus BPM/ESB Service Platform System Setup and Operation Guide*	*4.1.4*
		Maintenance and Migration Guide	*4.20*
		Command Reference Guide	*cjenvsetup (setting up*

Item	Overview of changes	Reference manual	Reference
Performing the log output of the `cjenvsetup` command	The execution information of the Component Container administrator setup (`cjenvsetup` command) can now be output in the message log.	*Command Reference Guide*	*the Component Container administrator)*
Supporting BIG-IP v11	BIG-IP v11 has been added to the available types of the load balancers.	*uCosminexus BPM/ESB Service Platform System Setup and Operation Guide*	*4.7.2*
		Virtual System Setup and Operation Guide	*2.1*
Performing the output of the CPU time to the event log of the Explicit management functionality	The CPU time required for releasing the Explicit memory block can now be output to the event log of the Explicit management functionality.	*Maintenance and Migration Guide*	*5.11.3*
Expanding the functionality of the user expanded performance analysis trace	The following functionality have now been added to the user expanded performance analysis trace: • The specification method to be traced can now be added to the specification method in the method unit and can be specified in the package unit or class unit. • The range of the available event IDs has been expanded. • Released the restrictions on the number of lines that can be specified in the user expanded performance analysis trace setup file. • The trace acquisition level can now be specified in the user expanded performance analysis trace setting file.	*Maintenance and Migration Guide*	*7.5.2*, *7.5.3*, *8.28.1*
Improving the information analysis used at the time of asynchronous calling of Session Bean	The requests at the calling source and the requests at the calling destination can now be compared by using the root application information of the PRF trace.	*EJB Container Functionality Guide*	*2.17.3*

2 Web Container

This chapter describes the Web container functionality that acts as the server platform for executing the servlets and JSPs. Use the Web container functionality when you execute a J2EE application with servlets or JSPs.

2.1 Organization of this chapter

Application Server provides functionality as a container (*Web Container*) that includes functionality for executing Web applications.

The following table lists the Web Container functionality provided with Application Server and the reference sections corresponding to the functionality:

Table 2-1: Web container functionality and the reference sections corresponding to each functionality

Functionality	Reference
Web application execution functionality	*2.2*
Functionality for executing JSPs	*2.3*
Functionality for debugging JSPs	*2.4*
JSP pre-compilation functionality and maintaining compilation results	*2.5*
Functionality for setting up the default character encoding	*2.6*
Session management functionality	*2.7*
Event listener of an application	*2.8*
Functionality of filtering the requests and responses	*2.9*
HTTP response compression functionality	*2.10*
Integrating with the EJB container	*2.11*
Connecting to the database	*2.12*
Creating a thread by using the Web container	*2.13*
Using the user thread	*2.14*
Controlling the number of concurrently executing threads	*2.15*
Controlling the number of concurrently executing threads in the Web container	*2.16*
Controlling the number of concurrently executing threads in each Web application	*2.17*
Controlling the number of concurrently executing threads in each URL group	*2.18*
Changing the number of concurrently executing threads dynamically	*2.19*
Customizing the error page	*2.20*
Caching the static contents	*2.21*
URI decode functionality	*2.22*
Version setup functionality of Web applications	*2.23*
Notes on the maximum size of POST data	*2.24*

The functionality of a Web container provided in Application Server includes functionality wherein the functions unique to Application Server are extended beyond the functions defined in J2EE, and also functions provided as functions unique to Application Server. For details on whether the functionality is unique to Application Server, see *1.2 Functionality corresponding to the purpose of the system*.

2.2 Web application execution functionality

You can execute the Web applications with a Web container. A Web application refers to a server program that runs in a Web container. This section describes the functionality of executing Web applications.

The following table describes the organization of this section.

Table 2-2: Organization of this section (Functionality for executing Web applications)

Category	Title	Reference
Description	Deploying and un-deploying Web applications	*2.2.1*
Notes	Notes on deploying and un-deploying Web applications	*2.2.2*

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

A normal Web server sends only fixed HTML files. Operate a Web container and execute the Web applications in the Web container. This execution of applications will allow you to process the data received from the Web client and generate different Web pages according to the results of the process.

A *Web application* is mainly developed by using two types of technologies; servlet and JSP. *Servlet* is a technology that uses a Java program, generates an HTML and processes the information received from the Web client. *JSP* (JavaServer Pages) is a technology that dynamically generates Web windows by embedding tags and java programs in HTML pages on the basis of the servlet technology. With the help of a JSP compiler, JSP is once transformed to a servlet program coded in Java and is then compiled and executed by a Java compiler.

With the Web container of Application Server, you can execute Web applications that conform to Java Servlet 3.0 specifications and JavaServer Pages (JSP) 2.1 specifications. Note that any newly added tag of the JSP 2.2 specifications is ignored. The `web.xml` file of the JSP 2.2 specifications can however be read without any problem. For details on the functionality for executing Web applications, see *Java Servlet Specification v3.0, and JavaServer Pages Specification v2.2*.

To execute the Web applications by using a Web container, you need to use Cosminexus HTTP Server, Microsoft IIS, or an in-process HTTP server as the Web server.

Reference note

The applications that can run with Application Server of a previous version can also run with this vesion.

2.2.1 Deploying and un-deploying web applications

A Web application having a WAR format changes to an executable format, when you deploy it by using any of the following methods:

- Package the Web application in an EAR format and import the packaged application as a J2EE application in an archive format or J2EE application in an exploded archive format.
- Import a single Web application by specifying a WAR file or a WAR directory.

A single Web application that is imported is called a *WAR application*. For details on WAR applications, see *13.9 WAR applications* in the *uCosminexus Application Server Common Container Functionality Guide*.

For details on the format of the executable J2EE applications, see *13.2 Format of executable J2EE applications* in the *uCosminexus Application Server Common Container Functionality Guide*.

When *deploy multiple Web applications* is executed, isolated class loaders are created for each Web application. Consequently, even if classes with the same class name (Login servlet) are used in different Web applications, they are handled independently on separate class loaders.

Note that in the case of un-deploying a Web application deployed as the J2EE application, un-deploy process is performed for each J2EE application. You cannot un-deploy a Web application in each WAR.

2.2.2 Notes when deploying and un-deploying web applications

This subsection describes the points to be considered when deploying and un-deploying a Web application.

(1) Default mapping of servlets or JSPs

The setting that specifies which servlet will be invoked for the URL requested by the client is called *Servlet mapping*. The Java servlet specification requires the servlet mapping to be coded in the DD (`WEB-INF/web.xml`).

On the other hand, the mapping defined by default in a Web container is called *default mapping*. In a Web container, the following mapping is defined by default:

Table 2-3: Default mapping of servlets or JSPs defined in a Web container

URL	Handling
`*.jsp`	Treated as a JSP file.
`*.jspx`	The Web applications compliant with Servlet 2.4 or later specifications are considered as the JSP documents. Note that the Web applications compliant with the Servlet 2.2 and Servlet 2.3 specifications are considered as the static contents.
`/servlet/*`	The classes of the servlet in the JAR file deployed under WEB-INF/classes or WEB-INF/lib are executed. The executed servlet is searched by the servlet name. When the '*' part of the URL is not defined as the servlet name, the servlet class is searched. In the '*' part of the URL, you can specify either the fully qualified class name of the servlet or the servlet name defined in `web.xml`. When the fully qualified class name of the servlet is specified, the specified servlet is executed. When the servlet name is specified, the servlet defined in `web.xml` is executed.

Note that for the servlet, `web.xml` must include mapping definition, and if `web.xml` is omitted, the default mapping of the servlet will not be enabled for the Web applications.

For the default mapping of the servlet, specify either enabled or disabled in the following parameter in the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file. By default, the default mapping of the servlet is disabled.

`webserver.container.servlet.default_mapping.enabled`

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

(2) Default values when tags in the DD (WEB-INF/web.xml) are not set

In a Web container, if the following tags are not set in the DD (`WEB-INF/web.xml`), use the following default values.

Table 2-4: Default values when tags in the DD (WEB-INF/web.xml) are not set

Tag names	Default values when tags are not set

`welcome-file-list`	`<welcome-file-list>`
	` <welcome-file>`
	` index.jsp`
	` </welcome-file>`
	` <welcome-file>`
	` index.html`
	` </welcome-file>`
	` <welcome-file>`
	` index.htm`
	` </welcome-file>`
	`</welcome-file-list>`

`session-timeout`	30
`mime-mapping`	Relationship between extensions and MIME types

When the tags in above table are set in the DD (`WEB-INF/web.xml`) , the settings will be as follows:

- The default values are overwritten with the value set in <welcome-file-list> tag.
- The default values are overwritten with the value set in <session-timeout> tag.
- The default values are overwritten for each extension, by the settings for each extension defined in <mime-mapping> tag.

For details on the relationship between the extensions and the MIME types set by default in the mime-mapping tag of DD (WEB-INF/web.xml), see *Appendix B.1 Correspondence Between Extensions and MIME Types* in the *uCosminexus Application Server Definition Reference Guide*.

(3) Number of instances created for a servlet

Create one servlet instance of the same class in each Web application. Also, create one servlet instance that inherits SingleThreadModel in each Web application in the same way as a general servlet.

However, if the same servlet and JSP are accessed by both default mapping and the mapping described in the DD, the instances are generated in the following manner.

Table 2-5: Generation of instances when the same servlet and JSP are accessed by both default mapping and the mapping described in the DD

URL	Instance
*.jsp	Generated instance is different from the one that was generated when the same servlet and JSP are accessed by the mapping described in DD.
*.jspx	
/servlet/*	• When the fully qualified class name of the servlet is specified: Generated instance is different from the one that was generated when the same servlet and JSP are accessed by the mapping described in DD. • When the servlet name defined in web.xml is specified: Generated instance is same as the one that was generated when the same servlet and JSP are accessed by the mapping described in DD.

Also, when multiple requests for a single servlet that inherits SingleThreadModel arrive concurrently in each Web application, control each thread so that the threads are executed one-by-one without overlapping.

(4) Timing to execute the init method and service method of servlets

The timing to execute the init method and service method of servlets differs according to the default mapping and the servlet mapping.

- When the servlets are accessed by default mapping:
  The init method and service method of servlets are executed as an extension of the filter processing mapped to the corresponding URL.
- When the servlets are accessed by servlet mapping:
  The init method is executed before the filter processing. The service method is executed as an extension of the filter processing mapped to the corresponding URL.

(5) Servlet buffer used for sending a response

In the servlet buffer used for sending a response, only the number of request processing threads is maintained. When you use the setBufferSize method of javax.servlet.ServletResponse to change the buffer size, estimate the memory usage after considering that the memory of buffer size × number of request processing threads is secured.

(6) Parsing the query character string

The query string includes a combination of one or more Name=Value after ? mark of URL.

With Application Server, if several = exist in the `Name=Value` part, the character string before the first = becomes the name and the character string after = becomes the value. For example, if the URL is `http://host/examples?a=b=c`, the string is analyzed as '`Value of name a is b=c`'.

2.3 JSP execution functionality

This section describes the functionality for executing JSP.

With the Web Container, you can convert the JSP that is created according to the JSP syntax provided in the Servlet specifications into a servlet, compile JSP as a Java program, and can load and execute the program in Java VM.

The following table describes the organization of this section.

Table 2-6: Organization of this section (Functionality for executing JSPs)

Category	Title	Reference
Description	Overview of JSP execution functionality	*2.3.1*
	Executing a tag file	*2.3.2*
	Executing JSP EL	*2.3.3*
	Storing the tag library in the J2EE applications	*2.3.4*
	Checking the attribute name of the custom tag	*2.3.5*
	Checking the duplication of the id attribute of the `<jsp:useBean>` tag	*2.3.6*
	Implicitly importing the `import` attribute of the `page/tag` directive	*2.3.7*

Note:

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

2.3.1 Overview of JSP execution functionality

You can code JSP with the standard syntax that is the standard format of the JSP specifications and the XML syntax that is the format of the XML specifications. JSPs coded with the standard syntax are called the *JSP pages* and JSPs coded with the XML syntax are called the *JSP documents*. Hereafter, JSP pages and JSP documents are collectively referred to as a *JSP file*.

(1) JSP configuration

A JSP includes elements (directive, action, and scripting element) and template text. The template text includes *white space*. Normally, the white spaces included in the template text are maintained as are, but from the JSP 2.1 specifications, functionality is added to delete the unwanted white spaces included in the template text of tag files of the JSP pages or standard syntax. For details, see *6.2.7(2) Functionality for deleting unwanted white spaces*.

Reference note

JSP EL has been provided from the JSP 2.0 specifications. JSP EL is a simplified language that allows you to code directly in an action or a template text. For details, see *2.3.3 Executing JSP EL*.

(2) Translation errors

A *translation error* is an error that occurs when the JSP file cannot be converted to a Java file (*JSP translation*) due to syntax errors in the JSP compilation process.

JSP translation is executed at the following times, but translation errors might occur at these times:

- When JSP receives a request
- When the applications are reloaded
- When the applications are started for using the JSP pre-compilation functionality
- When the commands are executed for using the JSP pre-compilation functionality

When a translation error occurs during the JSP compilation on a J2EE server, the message KDJE39145-E is output in the servlet log and the message KDJE39186-E is output in the message log. When a translation error occurs during the processing of a request, the redirector returns the error status code 500.

If a translation error occurs during the JSP pre-compilation using the `cjjspc` command, the messages KDJE39145-E and KDJE39186-E are displayed on the console.

Note that translation errors might also occur due to other causes, such as analysis of the TLD file, JSP validation by the tag library validator, or the duplication of script variables specified in the `TagExtraInfo` class. In these cases, the messages are displayed according to the cause. The following table lists the messages displayed during the translation error, with their respective cause:

Table 2-7: Messages output when translation errors occur due to the causes other than the JSP compilation

Classification of causes	Output messages
Parsing of the TLD file	KDJE39214-E, KDJE39216-E, KDJE39193-E, KDJE39055-E, KDJE39205-E, KDJE39206-E, KDJE39207-E, KDJE39208-E, KDJE39296-E, KDJE39301-E, KDJE39302-E, KDJE39303-E, KDJE39305-E, KDJE39306-E, KDJE39307-E, KDJE39308-E
JSP validation by tag library validator	KDJE39104-E, KDJE39105-E, KDJE39106-E, KDJE39107-E, KDJE39108-E, KDJE39115-E, KDJE39116-E, KDJE39117-E, KDJE39134-E, KDJE39135-E
Duplication of script variables specified in the `TagExtraInfo` class	KDJE39131-E, KDJE39132-E, KDJE39133-E, KDJE39136-E, KDJE39282-E, KDJE39283-E, KDJE39291-E, KDJE39294-E

2.3.2 Executing a tag file

In Application Server, you can execute tag files created as per the JSP syntax provided in JSP 2.0 or later. When you execute a tag file, the following contents are implemented:

- Transforming the tag file into a java class
- Compiling the transformed java class file
- Loading and executing the compiled file in JavaVM

If you use a tag file, you can describe the tag extension (that was conventionally achieved by a custom tag library) with the JSP syntax alone.

2.3.3 Executing JSP EL

In Application Server, you can execute an EL expression created according to the syntax of EL provided in JSP 2.0 or later. If you use JSP EL, you can describe the access to JavaBeans attributes in the JSP files and tag files.

With the EL functionality added in the JSP 2.1 specifications, you can code EL that supports the Enum type of Java SE, specifies values of the JavaBeans attributes, and shows methods. Also, in the JSP 2.1 specifications, in addition to EL provided in the JSP 2.0 specifications, you can code EL provided in the JSF1.1 specifications as the EL of JSP 2.1 specifications.

With the EL functionality added in the JSP 2.2 specifications, you can specify methods that have arguments.

For specifically indicating EL provided in JSP 2.0 specifications, specify *EL of JSP 2.0 specifications*. For specifically indicating EL of JSP 2.1 specifications, specify *EL2.1*. For details on specifically indicating the EL provided in the JSP 2.2 specifications, specify *EL2.2*.

2.3.4 Storing the tag library in the J2EE applications

With Application Server, you can store the tag library into the J2EE application and use the tag library from JSP.

The methods for using the tag library stored in the J2EE application from JSP differ based on whether the JSP is compiled using the J2EE server or using the `cjjspc` command.

When JSP is compiled using the J2EE server

Store the tag library in the library JAR.

When JSP is compiled using the cjjspc command

Store the tag library in the JAR file that is specified in the class path.

Both these methods are explained below.

(1) When JSP is compiled using the J2EE server

By storing the tag library in the library JAR, you can use the tag library from JSP.

(a) Searching the TLD file

If the tag library is stored in a J2EE application as a library JAR, you cannot specify the TLD file with the `<taglib>` element of `web.xml`. In this case, set up the URI specified in the `<uri>` element of the TLD file in the `uri` property of the `taglib` directive of the JSP file. When the Web application starts, the TLD file saved in the library JAR is searched by the Web container. The URI coded in the `<uri>` element of the searched TLD file and the respective TLD file will be mapped.

Note that when the `<uri>` element does not exist, the URI and the respective TLD file is not mapped.

The mapping of the TLD file in the library JAR has the lowest priority level. Therefore, even if the URI of the TLD file in the library JAR is duplicated with another TLD file and `web.xml` in the mapping of the URI and TLD file, this will not affect the operations of the J2EE applications running on the versions prior to 07-60 version. For details on the priority levels in the mapping of the TLD file within library JAR, see *6.2.6(7) Mapping of URI and TLD files specified in the uri attribute of the taglib directive*.

You can use the tag library in the JSP file by specifying URI in the `uri` attribute of the `taglib` directive.

(b) Tag library functionality that can be used when storing the tag library in the library JAR

The tag library functionality that you can use to store the tag library in the library JAR are the custom tag and the listener.

You cannot use a tag file. If you use a tag file stored in the library JAR for a JSP file or for a tag file, the `<tag-file>` element of the TLD file stored in the library JAR will be ignored. Therefore, considering that the tag file does not exist, a translation error occurs during JSP translation.

(2) When compiling JSP using the cjjspc command

By specifying the tag library that exists within the JAR file in the class path with the `-classpath` option of the `cjjspc` command, you can use the tag library deployed outside the WEB-INF/lib directory of the Web application from JSP.

(a) Searching the TLD file

The TLD file included in the JAR file specified in the class path is searched automatically and URI specified in the `<uri>` element of the TLD file and the TLD file are mapped.

The mapping of the TLD file in the JAR file specified in the class path has the lowest priority level. Therefore, even if the URI of the TLD file within a JAR file specified in the class path is duplicated with another TLD file and `web.xml` in the mapping of the URI and TLD file, this does not affect the operations of the J2EE applications running with the versions prior to the 07-60 version. For details on the priority levels for the mapping of the TLD file within the JAR file specified in a class path, see *6.2.6(7) Mapping of URI and TLD files specified in the uri attribute of the taglib directive*.

(b) Tag library functionality that can be used when storing the tag library in the JAR file specified in the class path

The tag library functionality that you can use for storing the tag library in the JAR file specified in the class path are the custom tag and the listener.

You cannot use a tag file. If you use a tag file stored in the JAR file specified in the class path for a JSP file or a tag file, the `<tag-file>` element of the TLD file stored in the JAR file will be ignored. Therefore, considering that the tag file does not exist, a translation error occurs during the JSP translation.

2.3.5 Checking the attribute name of the custom tag

When you check the attribute name of the custom tag, if the upper case and the lower case of the following attribute names do not match, a translation error occurs.

- Attribute names specified in the JSP custom tag
- Attribute names defined in the TLD file or tag file

With Application Server, you can control the occurrence of translation errors because of mismatch in the upper case and the lower case, when checking the attribute name of the custom tag. If you control the occurrence of such translation errors, the definition of the custom tag attribute is searched without case-sensitivity from the attribute names defined in the TLD file or tag file.

The locations for defining the attribute names of the TLD file and tag file are as follows:

- The attribute name defined in the `<name>` element exists in the `<taglib><tag><attribute>` element of the TLD file
- The attribute name defined in the `attribute` directive of the tag file

(1) How to disable the attribute name check of the custom tag

Use one of the following methods to control the occurrence of translation errors due to mismatch in upper case and lower case when checking the attribute name of the custom tag:

- Specify `true` in the parameter `webserver.jsp.translation.customAction.ignoreCaseAttributeName` in the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file.

  For details on the Easy Setup definition file and the parameters to be specified, see *4.6 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.
- Compile JSP by specifying the `-usebeannocheckduplicateid` option in the `cjjspc` command.

  For details on the command, see *cjjspc (JSP pre-compilation)* in the *uCosminexus Application Server Command Reference Guide*.

(2) Notes

If there are case-sensitive attributes, do not control the occurrence of translation errors because of mismatch in the upper case and the lower case. If such translation errors are controlled, the following problems will occur:

- If multiple attributes that are only distinguished by the upper case or the lower case are specified in the JSP custom tag

  The `setter` method of the tag handler is executed for each of the attributes. At this time, even if the same attributes are specified several times, the translation error does not occur. Therefore, the `setter` method might be executed for multiple identical attributes.
- If tag handler corresponding to the attribute name distinguished only by the upper case or the lower case is implemented

  The `setter` method of the attribute described first in the TLD file or tag file is invoked without case-sensitivity. At this time, the translation error does not occur. Therefore, you might not be able to invoke the intended `setter` method.

2.3.6 Checking the duplication of the id attribute of the <jsp:useBean> tag

If the `id` attribute value of the `<jsp:useBean>` tag is duplicated in the JSP specifications, a translation error occurs and the JSP compilation fails.

With Application Server, you can compile JSP by controlling the occurrence of translation errors because of the duplication of the `id` attribute value of the `<jsp:useBean>` tag.

If you control the occurrence of translation errors because of the duplication of the id attribute value of the <jsp:useBean> tag, the JSPs will run without any problem even if the id attribute value of the <jsp:useBean> tag is duplicated. The following JSP file coding example describes this case:

Example for coding a JSP file

```
(Omitted)
...
<% if (Conditional-expression) { %>
<jsp:useBean id="BeanTest" class="test.TestClass1" />
<% } else { %>
<jsp:useBean id="BeanTest" class="test.TestClass2" />
<% } %>
...
(Omitted)
```

By using the scriptlet and coding the <jsp:useBean> tag in the if clause and the else clause respectively, only one <jsp:useBean> tag is executed. Therefore, even if the id attribute value of the <jsp:useBean> tag is duplicated, JSP will operate without a problem.

When you control the occurrence of translation errors because of duplication of the id attribute value of the <jsp:useBean> tag, the KDJE39544-I message is output when the id attribute value of the <jsp:useBean> tag is duplicated.

(1) How to enable the id attribute duplication check of the <jsp:useBean> tag

To control the occurrence of translation errors because of the duplication of the id attribute value of the <jsp:useBean> tag, use one of the following methods:

- Specify true in the parameter webserver.jsp.translation.useBean.noCheckDuplicateId in the <configuration> tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file.

  For details on the Easy Setup definition file and the parameters to be specified, see *4.6 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.
- Compile JSP by specifying the -usebeannocheckduplicateid option in the cjjspc command

  For details on the command, see *cjjspc (JSP pre-compilation)* in the *uCosminexus Application Server Command Reference Guide*.

(2) Notes

The JavaBeans object created in the <jsp:useBean> tag is managed within the scope that is specified in the scope attribute using the id attribute value as the key. Therefore, when the following conditions are fulfilled, the unintended JavaBeans objects might be acquired and the invalid operations might be performed:

- The occurrence of the translation error because of the duplication of the id attribute value in the <jsp:useBean> tag is controlled.
- The id attribute value of the <jsp:useBean> tag is duplicated.
- Different JavaBeans classes are specified in the class attributes of two or more <jsp:useBean> tags where the id attribute value is duplicated

The following points describe the examples for coding JSP files in which the intended objects might not be acquired, when controlling the occurrence of a translation error because of the duplication of the id attribute value in the <jsp:useBean> tag.

(a) Coding example of JSP file when an unintended JavaBeans object is acquired with a single request (Example in which the same id attribute is specified in two or more different conditional expressions)

```
(Omitted)
```

```
...
<% if (Conditional-expression-1) { %>
<jsp: useBean id="BeanTest" class="test.TestClass1" scope="page"/>
<% } %>
<% if (Conditional-expression-1) { %>
<jsp: useBean id="BeanTest" class="test.TestClass2" type="test.TestIF" scope="page"/>
<% } %>
...
(Omitted)
```

The same `id` attribute value (BeanTest) is specified in the conditional expression 1 and the conditional expression 2. If the conditions in both conditional expression 1 and conditional expression 2 are fulfilled, the `test.TestClass1` class object is acquired in the first <jsp:useBean> tag and the second <jsp:useBean> tag, and the `test.TestClass2` class object is not acquired.

The `test.TestClass1` class object is registered for the `id` attribute value 'BeanTest' in the first <jsp:useBean> tag. As per the interpretation, the processing for the `id` attribute value 'BeanTest' is the same as that of the second <jsp:useBean> tag, and therefore, the `test.TestClass1` class object that is already registered in the first <jsp:useBean> tag will be acquired.

(b) Coding example of JSP file when unintended JavaBeans objects are acquired with multiple requests (Example in which the same id attribute value is specified in if and else statements)

```
(Omitted)
...
<% if (Conditional-expression) { %>
<jsp: useBean id="BeanTest" class="test.TestClass1" scope="session"/>
<% } else { %>
<jsp: useBean id="BeanTest" class="test.TestClass2" scope="session"/>
<% } %>
...
(Omitted)
```

The same `id` attribute value (BeanTest) is specified in `if` and `else` statements. If a request is received twice and the conditional expression of the `if` statement is established in the first request and the conditional expression of the `if` statement is not established in the second request, the `test.TestClass1` class object is acquired in both the first <jsp:useBean> tag and the second <jsp:useBean> tag, and the `test.TestClass2` class object is not acquired.

The `test.TestClass1` class object is registered for the `id` attribute value 'BeanTest' in the first <jsp:useBean> tag. As per the interpretation, the processing for the `id` attribute value 'BeanTest' is the same as that of the second <jsp:useBean> tag, and therefore, the `test.TestClass1` class object that is already registered in the first <jsp:useBean> tag will be acquired.

(c) Coding example 2 of JSP file when unintended JavaBeans objects are acquired with multiple requests (Example 1 of invoking JavaBeans object by using the id attribute value, commonly specified in 2 or more <jsp:useBean> tags, in <jsp:getProperty> tag or <jsp:setProperty> tag)

```
(Omitted)
...
<% if (Conditional-expression-1) { %>
<jsp:useBean id="BeanTest" class="test.TestClass1" scope="page"/>
<% } %>
<% if (Conditional-expression-2) { %>
```

```
<jsp:useBean id="BeanTest" class="test.TestClass2" type=" test.TestIF" scope="page"/>
<% } %>
...
<jsp:setProperty name="BeanTest" property="*"/>
...
<jsp: getProperty name="BeanTest" property="value"/>
...
(Omitted)
```

The JavaBeans object created in the <jsp:useBean> tag will be invoked with the processes defined in the <jsp:getProperty> and the <jsp:setProperty> tag.

If the id attribute value is duplicated in 2 or more <jsp:useBean> tags that create different JavaBeans classes, the object specified in the <jsp:useBean> tag that appears at the end is used for the processing in the <jsp:getProperty> or <jsp:setProperty> tags.

For example, the same id attribute value (BeanTest) is specified in the conditional expression 1 and conditional expression 2. Therefore, even if the conditional expression 1 is established, the test.TestClass1 class object registered in first <jsp:useBean> tag is not used for the processing in the <jsp:getProperty> and <jsp:setProperty> tags.

(d) Coding example 3 of JSP file when unintended JavaBeans objects are acquired with multiple requests (Example 2 of invoking JavaBeans object by using the id attribute value, commonly specified in 2 or more <jsp:useBean> tags, in <jsp:getProperty> tag or <jsp:setProperty> tag)

```
(Omitted)
...
<% if (Conditional-expression-1) { %>
<jsp: useBean id="BeanTest" class="test.TestClass1" scope="page"/>
...
<jsp: setProperty name="BeanTest" property="*"/>
...
<jsp: getProperty name="BeanTest" property="value"/>
...
<% } %>
<% if (Conditional-expression-2) { %>
<jsp: useBean id="BeanTest" class="test.TestClass2" type=" test.TestIF" scope="page"/>
...
<jsp: setProperty name="BeanTest" property="*"/>
...
<jsp: getProperty name="BeanTest" property="value"/>
...
<% } %>
...
(Omitted)
```

The JavaBeans object created in the <jsp:useBean> tag will be invoked in the processes defined in the <jsp:getProperty> tag and the <jsp:setProperty> tag.

If the id attribute value is duplicated in two or more <jsp:useBean> tags that create different JavaBeans classes, the object specified in the <jsp:useBean> tag that is displayed at the end is used for the processing in the <jsp:getProperty> tag or the <jsp:setProperty> tag.

For example, the same id attribute value (BeanTest) is specified in the conditional expression 1 and conditional expression 2. Therefore, even if the conditional expression 1 is established, the test.TestClass2 class object

registered in the second <jsp:useBean> tag is used for the processing in the first <jsp:useBean> tag. The test.TestClass1 class object registered in the first <jsp:useBean> tag is not used for the processing in the <jsp:getProperty> tag and the <jsp:setProperty> tag.

2.3.7 Implicitly importing the import attribute of the page/tag directive

According to the JSP specifications, the following classes are imported implicitly while compiling JSP:

- java.lang.*
- javax.servlet.*
- javax.servlet.jsp.*
- javax.servlet.http.*

If you use the functionality for implicitly importing the import attribute of the page/tag directive, you can implicitly import any desired class besides those described above.

(1) How to specify the classes to be imported implicitly

This subsection describes how to specify the functionality for implicitly importing the import attribute of the page/tag directive. You can use the functionality for implicitly importing the import attribute of the page/tag directive when compiling JSP in a J2EE server, or when compiling JSP by the cjjspc command.

Specify the class to be imported implicitly with a fully qualified class name or with the "package name.*". When specifying multiple class names, use a comma (,) to demarcate two class names. When you specify a class name that does not exist or a class name with an invalid class path, the KDJE39143-E message is output during JSP compilation.

- When compiling JSP in a J2EE server

  Specify the class to be imported implicitly in the webserver.jsp.additional.import.list key of the Easy Setup definition file.

  For details on the Easy Setup definition file and the parameters to be specified, see *4.6 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.
- When performing JSP pre-compilation by the cjjspc command

  Specify the class to be imported implicitly in the -addimport option of the cjjspc command.

  For details on the command, see *cjjspc (JSP pre-compilation)* in the *uCosminexus Application Server Command Reference Guide*.

(2) Output order of the import statements

The order of the import statements output in the java file during JSP compilation is as follows:

1. Import statement of classes specified in JSP specifications
   - java.lang.*
   - javax.servlet.*
   - javax.servlet.jsp.*
   - javax.servlet.http.*
2. Import statement of the class specified in the import attribute of the page/tag directive
3. Import statement of the class specified in the functionality for implicitly importing the import attribute of the page/tag directive

(3) Notes

(a) JSP file for which JSP pre-compilation is performed by the cjjspc command

The functionality for implicitly importing the `import` attribute of the `page/tag` directive operates during JSP compilation. When JSP pre-compilation is performed by the `cjjspc` command, the class specified in the `webserver.jsp.additional.import.list` key of the Easy Setup definition file is not imported implicitly to the Web application for which JSP pre-compilation is performed. Therefore, to perform JSP pre-compilation by the `cjjspc` command, specify the `-addimport` option in the `cjjspc` command and then perform JSP pre-compilation for the Web application, when using the functionality for implicitly importing the `import` attribute of the `page/tag` directive.

(b) When multiple Web applications exist in a J2EE server

The class specified in the `webserver.jsp.additional.import.list` key of the Easy Setup definition file is enabled for all Web applications in the J2EE server for which JSP pre-compilation is not performed by the `cjjspc` command. If you want to specify a different class for each Web application, specify the `-addimport` option in the `cjjspc` command, and then perform JSP pre-compilation for the Web applications.

(c) Recompiling after JSP pre-compilation

When all the below conditions are satisfied, the `KDJE39143-E` message is output during recompilation. Therefore, when using a Web application for which JSP pre-compilation is performed by specifying the `-addimport` option in the `cjjspc` command, specify the same class name as the one specified in the `-addimport` option of the `cjjsps` command in the `webserver.jsp.additional.import.list` key of the Easy Setup definition file.

- A class is not defined with a fully-qualified class name in the JSP file.
- JSP pre-compilation is performed by specifying the class name to be imported implicitly in the `-addimport` option of the `cjjspc` command.
- In the `webserver.jsp.additional.import.list` key of the Easy Setup definition file, specify a class name different from the class to be imported implicitly that is specified in the `-addimport` option of the `cjjspc` command. Alternatively, either omit the `webserver.jsp.additional.import.list` key or specify a blank in the `webserver.jsp.additional.import.list` key.
- The Web applications for which JSP pre-compilation is performed in the J2EE server are re-compiled.

(d) When a class name that already belongs to another class is specified

When the class name specified in the functionality for implicitly importing the `import` attribute of the `page/tag` directive duplicates a class name within the package to be imported that is specified in the JSP specifications, or the class name specified in the `import` attribute of the `page/tag` directive, a compilation error occurs during JSP compilation, and the `KDJE39143-E` message is output.

An example when a compilation error occurs during JSP compilation is described below. Note that in the example, the following class names must exist as a prerequisite:

- `packageA.classA`
- `packageB.classA`

■ When import class names from different packages overlap

An example is described below.

For the following specification contents, when an attempt is made to import classes with the same name from multiple packages, an error occurs during JSP compilation.

Type of file	Specification contents
JSP file	`<%@page import="packageA.classA" %>`
Easy Setup definition file	`webserver.jsp.additional.import.list=packageB.classA`

■ When the import source package of the class used in a JSP file cannot be identified

An example is described below.

For the following specification contents, whether the `classA` used in the JSP file is `packageA.classA` or `packageB.classA` cannot be identified and an error occurs during JSP compilation.

Type of file	Specification contents
JSP file	`<%@page import="packageA.* " %>` `<% System.out.println(classA.method1()); %>`
Easy Setup definition file	`webserver.jsp.additional.import.list=packageB.*`

2.4 JSP debug functionality

This section describes the JSP debug functionality.

Using the JSP debug functionality, you can execute the debug tool functionality, such as the breakpoint settings in the JSP file, and the debugging in the post-conversion java source will not be required. Note that you can use the JSP debug functionality with version 2.0 or later JSP files.

The following table describes the organization of this section.

Table 2-8: Organization of this section (JSP debug functionality)

Category	Title	Reference
Description	Mechanism of JSP debug functionality	*2.4.1*
	Procedure of using the JSP debug functionality	*2.4.2*
Settings	Execution environment settings (J2EE server settings)	*2.4.3*
Notes	Precautions for using the JSP debug functionality	*2.4.4*

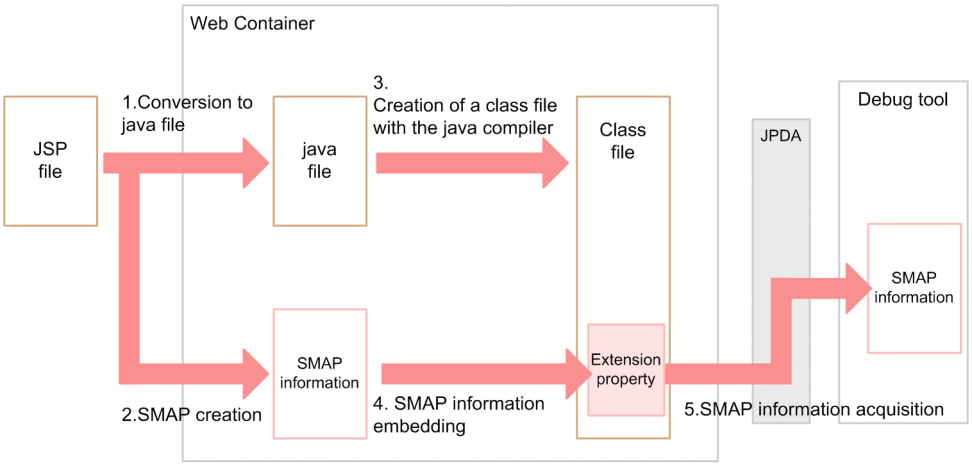
Note:

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

2.4.1 Mechanism of JSP debug functionality

The following figure shows the mechanism of the JSP debug functionality:

Figure 2-1: JSP debug mechanism



The following points describe the flow of data in the figure:

1. Convert the JSP file to the servlet java file.
2. Create SMAP (Source MAP) that describes the mapping of the JSP file lines and the lines of the java file converted from the JSP file.
3. Use java compiler to create the class file from the java file.
4. Embed the SMAP information in the extension attribute (`SourceDebugExtension` attribute) of the created class file.
5. During debug, use JPDA (Java Platform Debugger Architecture) to acquire embedded SMAP from the extension information embedded in the class file in the debug tool.

Note that if an attempt to embed the SMAP information fails, the message KDJE39324-E is output.

Important note

Class names when the JSP debug functionality is used

The names of the classes created during the compilation are different for the case when the JSP debug functionality is used and when the functionality is not used. For details on the class names in the JSP compilation results, see *2.5.7 Class names in JSP compilation results*.

2.4.2 Procedure of using the JSP debug functionality

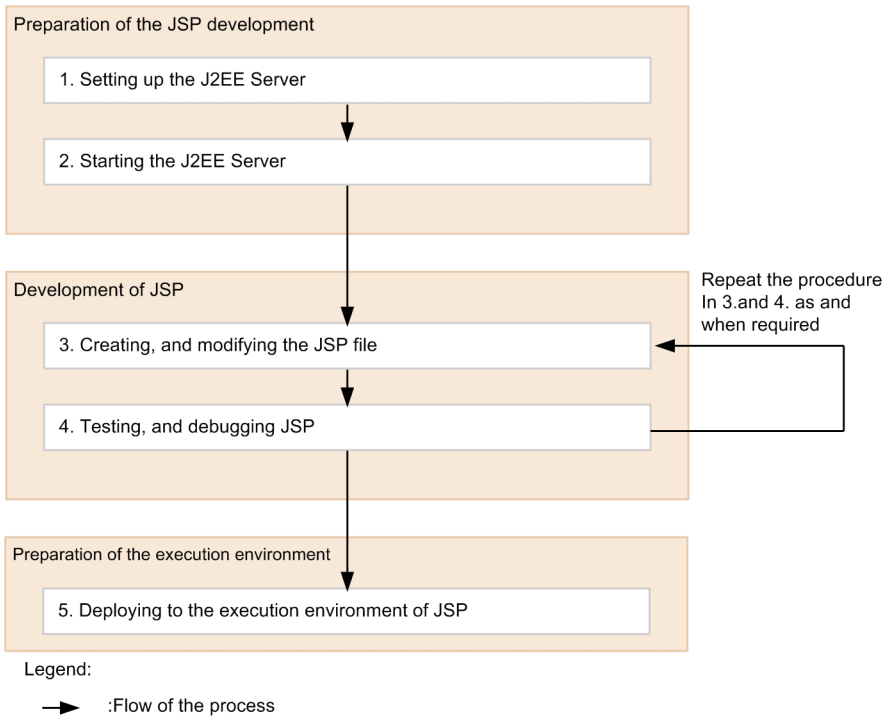
This subsection describes how to use the JSP debug functionality. You can use the JSP debug functionality to compile JSP with the J2EE server or with the `cjjspc` command.

How to use the JSP debug functionality for each case is described below:

(1) When compiling JSP with the J2EE server

The following figure shows the flow of usage procedure of the JSP debug functionality when compiling JSP with the J2EE server:

Figure 2-2: Usage procedure of the JSP debug functionality (when compiling JSP with the J2EE server)



The following points describe the procedure in the above figure:

1. Setting up the J2EE server
   Enable the JSP debug functionality. Also, enable JSP reload. For details on JSP reload, see *13.8.9 Reloading JSP* in the *uCosminexus Application Server Common Container Functionality Guide*.
2. Starting the J2EE server
   Start the J2EE server.
   If the JSP debug functionality is enabled, the KDJE39322-W message is output in the message log when the J2EE server starts.
3. Creating or modifying the JSP file
   Create or modify JSP files.
4. Testing and debugging the JSP file

Use the debug tool supporting JPDA, such as WTP to test and debug. To modify JSPs, return to step 3.

5. Distribution of JSP to the execution environment

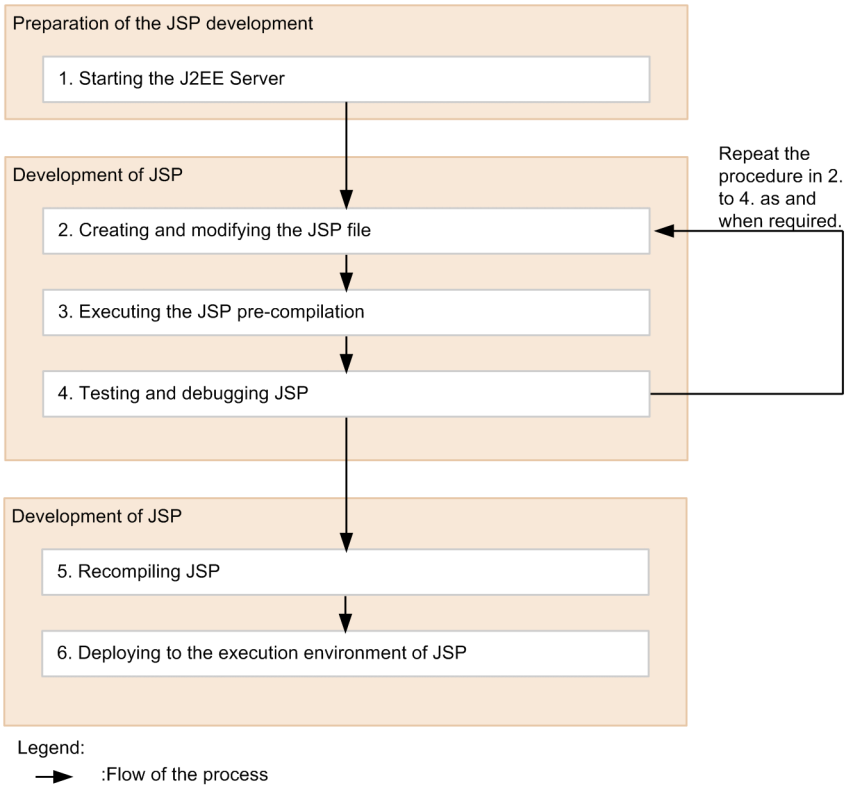
   Export the J2EE applications containing the created JSPs in the development environment, and then import the applications into the execution environment.

(2) When compiling JSP with the cjjspc command

For compiling all JSPs before starting the J2EE applications, execute the JSP pre-compilation functionality with the `cjjspc` command. For details on the JSP pre-compilation functionality, see *2.5 JSP pre-compilation functionality and maintaining compilation results*.

The following figure shows the flow of usage procedure of the JSP debug functionality when JSP pre-compilation is executed with the `cjjspc` command.

Figure 2-3: Usage procedure of the JSP debug functionality (when compiling JSP with the cjjspc command)



The following points describe the procedure in the above figure:

1. Starting the J2EE server

   Enable the JSP debug functionality. Also, enable JSP reload. For details on JSP reload, see *13.8.9 Reloading JSP* in the *uCosminexus Application Server Common Container Functionality Guide*.

   If the JSP debug functionality is enabled, the KDJE39322-W message is output in the message log when the J2EE server starts.

2. Creating or modifying the JSP file

   Create or modify JSP files.

3. Executing the JSP pre-compile

   Use the `cjjspc` command to execute the JSP pre-compilation functionality. Specify the `-debugging` option to execute the command, and compile the JSP file.

   During the execution of the `cjjspc` command, the KDJE53442-W message is output in the console log.

4. Testing and debugging JSP

Use the debug tool supporting JPDA, such as WTP for testing and debugging. To modify JSPs, return to step 2.

5. Re-compiling JSP

   Delete the JSP work directory. Also, use the `cjjspc` command without specifying the `-debugging` option to re-compile the JSP file.

   Note that if the JSP file is not re-compiled, you cannot execute JSP. For details, see *2.4.4 Precautions for using the JSP debug functionality*.

6. Distribution of JSP to the execution environment

   Export the J2EE applications containing the created JSP in the development environment, and then import the applications into the execution environment.

For details on the `cjjspc` command, see *cjjspc (JSP pre-compilation)* in the *uCosminexus Application Server Command Reference Guide*.

2.4.3 Execution environment settings (J2EE server settings)

To use the JSP debug functionality, you must specify the J2EE server settings.

Implement the J2EE server settings in the Easy Setup definition file. Define the JSP debug functionality in `webserver.jsp.debugging.enabled` within the `<configuration>` tag of the logical J2EE server (j2ee-server), in the Easy Setup definition file. In this parameter, you specify whether to use the JSP debug functionality.

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

2.4.4 Precautions for using the JSP debug functionality

This subsection describes the precautions for using the JSP debug functionality.

(1) Deleting the class file created in the JSP debug functionality

The class files created using the JSP debug functionality cannot be used in an environment where the JSP debug functionality is disabled. If you want to distribute the J2EE applications created in the JSP debug functionality-enabled environment to the JSP debug functionality-disabled environment, you must delete the class files created using the JSP debug functionality.

To delete class files:

1. **When the JSP pre-compilation functionality is used to compile JSP for applications in the exploded archive format**

   Delete the JSP work directory of the JSP pre-compilation functionality. For details on the JSP working directory, see *2.5.5(2) Output destination of JSP compilation results*.

2. **When JSP is compiled using any method other than the method of step 1.**

   Use the `cjstopapp` command to stop the J2EE application.

(2) J2EE server settings when the cjjspc command is used to compile JSP

To use the `cjjspc` command for compiling JSPs, enable the JSP debug functionality in the J2EE server executed by JSPs.

If you specify the `-debugging` option for the `cjjspc` command to compile JSPs of the J2EE server applications in which the JSP debug functionality is disabled, the loaded class files will be different. Therefore, the Web container returns the error status code 404 for the HTTP request of JSP.

2.5 JSP pre-compilation functionality and maintaining compilation results

This section describes the JSP pre-compilation functionality and maintenance of compilation results.

Normally, the JSP files in a Web application are compiled on the Web container when the first request arrives for a JSP file. If you use the JSP pre-compilation functionality, you can compile a JSP file before deploying the Web applications. If you compile the JSP files beforehand by the JSP pre-compilation functionality, you can shorten the response time when the first request arrives for a JSP file.

You can also specify whether to maintain the Java source files and the class files that act as JSP compilation results, when the J2EE server is restarted.

The following table describes the organization of this section.

Table 2-9: Organization of this section (JSP pre-compilation functionality and maintaining compilation results)

Category	Title	Reference
Description	Overview of the JSP pre-compilation functionality	*2.5.1*
	Methods for performing JSP pre-compilation	*2.5.2*
	Examples of application of JSP pre-compilation	*2.5.3*
	Processing during execution of JSP pre-compilation	*2.5.4*
	Lifecycle and output destination of JSP compilation results	*2.5.5*
	JSP compilation results when JSP pre-compilation functionality is not used	*2.5.6*
	Class names in JSP compilation results	*2.5.7*
Settings	Execution environment settings (J2EE server settings)	*2.5.8*

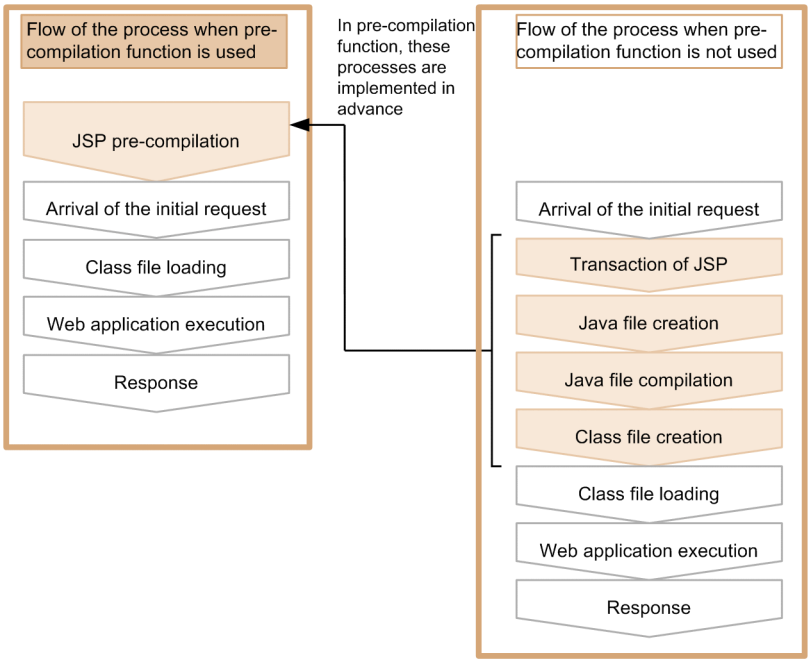
Note:
There is no specific description of *Implementation*, *Operations*, and *Notes* for this functionality.

2.5.1 Overview of the JSP pre-compilation functionality

The JSP pre-compilation functionality is used to compile the JSP files contained in a Web application before they are deployed and then generate the class files.

The following figure shows the processing flow when the JSP pre-compilation functionality is used.

Figure 2-4: Processing flow when the JSP pre-compilation functionality is used



Normally, the JSP files contained in a Web application are compiled when the first request arrives for a JSP file and class files are generated from the JSP files. As a result, the response time during the first request for JSP is delayed. If `<load-on-startup>` is specified in `web.xml`, the compilation time becomes the starting time of the Web application, and therefore, the response time when the first request arrives for JSP can be shortened. Web application, however, will take a long time to start.

When you use the JSP pre-compilation functionality, the response time when the first request arrives in JSP and the starting time of the Web application can be shortened, since the class files are already generated.

2.5.2 Methods for performing JSP pre-compilation

The following are two methods for performing JSP pre-compilation:

- JSP pre-compilation with the `cjspc` command
- JSP pre-compilation when the J2EE application is started with the `cjstartapp` command

This subsection provides an overview of the commands used for JSP pre-compilation and explains the compilation method. Note that the compilation method depends upon stages for pre-compiling the JSP files. For details on the applicable stages for JSP pre-compilation and compilation method, see *2.5.3 Examples of applying JSP pre-compilation*.

(1) Overview of commands

This subsection explains the preconditions for implementing the JSP pre-compilation, the files required during execution of the JSP pre-compilation and the files generated after executing JSP pre-compilation:

Preconditions

To implement JSP pre-compilation, it is assumed that the JSP files to be compiled are stored under the Web application root directory or under the sub directory.

Files required for JSP pre-compilation

The following files are required for implementing JSP pre-compilation:

- JSP files (JSP 1.1, JSP 1.2, JSP 2.0, or JSP 2.1)[#1]
- Tag files compliant with JSP 2.0 specifications or JSP 2.1 specifications

- Files that are statically included from the JSP files and the tag files
- TLD file#2
- `web.xml`#2#3
- Class library necessary for compilation

#1 JSP files implies the following files:

- Files with `.jsp` or `.jspx` extension (`.jspx` is only in the case of JSP 2.0 or later)
- Files specified in `<jsp-file>` tag of `web.xml`
- Files matching with `<jsp-property-group><url-pattern>` tag of `web.xml` (only in the case of JSP 2.0 or later)

#2 Whether the tag files conform to the DTD or XML schema during execution of JSP pre-compilation is verified.

#3 If `web.xml` does not exist, the compilation is executed assuming the Web application version to be version 3.0.

Files generated after JSP pre-compilation JSP compilation results)

The Java source files and the class files generated from the JSP files and the tag files are called *JSP compilation results*. When you implement JSP pre-compilation, the following JSP compilation results are generated in the JSP working directory.

- The Java source files and class files generated from the JSP files
- The Java source files and class files generated from the tag files

Note that during the execution of the JSP pre-compilation, you can specify whether to save the Java source files.

(2) JSP pre-compilation with the cjjspc command

The `cjjspc` command is used to implement JSP pre-compilation. If you implement this command during development of an application, you can compile the JSP files contained in a Web application. JSP pre-compilation with the `cjjspc` command includes the following two methods:

- Pre-compilation in each JSP file
  This method compiles only the specified JSP files among the JSP files included in a Web application.
- Pre-compilation in each Web application
  This method compiles all the JSP files included in each Web application.

You can specify the following contents during execution of the `cjjspc` command:

- Specifying the JSP files that need not be compiled
  You can exclude the pre-compilation of JSP files that need not be compiled by specifying them beforehand. The specification methods are as follows: Specify the JSP file names that need not be compiled one-by-one in the command. Mention all the JSP file names that need not be compiled together in a file and then specify that file in the command.
- Specifying whether to output the list file of execution results
  You can specify whether to output the list file of execution results. The list file of execution results refers to the file in which the execution results of the `cjjspc` command are output. The paths of the JSP files that are compiled successfully, the JSP files that were not be compiled, and the JSP files that are excluded from compilation are output in a list.
- Specifying whether to save the Java source files
  You can specify whether to save the Java source files generated from the JSP.
- Specifying the version of Java language specification during JSP compilation
  You can specify the version of Java language specification during compilation of the Java source file generated by JSP translation.
- Specifying whether to change the name of the JSP working directory
  The JSP working directory refers to the directory that saves the JSP compilation results. You can change the name of the JSP working directory. For details on the JSP working directory, see *2.5.5(2) Output destination of JSP compilation results*.

- Specifying the default character encoding
  You can specify the default character encoding for the JSP file. For an overview of the default character encoding, see *2.6 Functionality for setting up the default character encoding*. Also, for details on the precautions when you set up the default character encoding with the version 07-10 for the Web applications that use the JSP pre-compilation functionality with the version 07-00, see *2.6.8(5) Specifying the character encoding for Web applications that executed JSP pre-compilation with version 07-00*.
- Specifying whether to use the JSP debug functionality
  You can specify whether you want to use the JSP debug functionality. For details on the JSP debug functionality, see *2.4 JSP debug functionality*.
- Specifying the classes to be imported implicitly
  You can specify the class name to be imported implicitly by using the functionality for implicitly importing the `import` attribute of the `page/tag` directive. For details on the functionality for implicitly importing the `import` attribute of the `page/tag` directive, see *2.3.7 Implicitly importing the import attribute of the page/tag directive*.

Note that these settings are specified with command options. For details on the usage of the JSP pre-compilation command (`cjjspc` command), see *cjjspc (JSP pre-compilation)* in the *uCosminexus Application Server Command Reference Guide*.

! Important note

Changing the JavaVM runtime options

If you set the environment variable `CJ_CMD_JVM_ARGS`, you can change the JavaVM runtime options operated by the `cjjspc` command.

By default, `-Xmx512m` (maximum value of Java heap memory area is 512 MB) is specified in the JavaVM runtime option. When you want to compile large-scale Web applications using the `cjjspc` command, the maximum value of the Java heap memory area might exceed and `java.lang.OutOfMemoryError` might occur. Therefore, when compiling large-scale Web applications, you must specify the appropriate Java heap memory area in the environment variable `CJ_CMD_JVM_ARGS` in advance.

Reference note

- When performing the JSP pre-compilation with the `cjjspc` command, if an error occurs during translation of the JSP files or the tag files, error messages are output. An error message is output to the console.
- When the command is executed, the log is output in the standard output or standard error output. To keep the result of log output in a file, redirect the command output to a file.
  The following are the specification examples for keeping the results of log output in a file:
  In Windows

```
> cjjspc -root D: \app\webapp1 1> .\stdout.log 2> .\stderr.log
```

  In UNIX

```
# cjjspc -root /app/webapp1 1> ./stdout.log 2> ./stderr.log
```

(3) JSP pre-compilation when a J2EE application is started by the cjstartapp command

The `cjstartapp` command is used to start a J2EE application. In the `cjstartapp` command, if you specify the option that performs the JSP pre-compilation, the J2EE application is started after the JSP pre-compilation is implemented. The JSP pre-compilation functionality at the time of starting the J2EE application compiles all the JSP files contained in the J2EE application.

You can specify the settings for operation during execution of the `cjstartapp` command in advance. The following contents can be specified:

- Specifying whether to save the Java source files
  You can specify whether to save the Java source files generated from the JSP file.
- Specifying the version of Java language specification during JSP compilation
  You can specify the version of Java language specification during compilation of the Java source file generated by JSP translation.

- Specifying whether to change the name of the JSP working directory
  The JSP working directory refers to the directory that saves the JSP compilation results. You can change the name of the JSP working directory. For details on the JSP working directory, see *2.5.5(2) Output destination of JSP compilation results*.
- Specifying the classes to be imported implicitly
  You can specify the class name to be imported implicitly by using the functionality for implicitly importing the `import` attribute of the `page/tag` directive. For details on the functionality for implicitly importing the `import` attribute of the `page/tag` directive, see *2.3.7 Implicitly importing the import attribute of the page/tag directive*.

Note that you customize the operation settings of the J2EE server to implement these settings. For details on the customization of the operation settings of the J2EE server, see *2.5.8 Execution environment settings (J2EE server settings)*.

Reference note

When performing the JSP pre-compilation by the `cjstartapp` command, if an error occurs during translation of the JSP files or tag files, an error message is output to the Web servlet log or the message log.

Important note

Compilation of Java source generated from JSP

The class file generated using with the `cjjspc` command is used at runtime on the J2EE server. In the `cjjspc` command, a class file same as that of the file generated on the J2EE server will be generated.

Therefore, when compiling Java source that is generated from the JSP file or tag file, you can only specify the version of the Java language specification in the `-source` option or the class path in the `classpath` option. For details on how to specify the versions of the Java language specifications, see *cjjspc (JSP pre-compilation)* in the *uCosminexus Application Server Command Reference Guide*.

2.5.3 Examples of applying JSP pre-compilation

You can use the JSP pre-compilation functionality in the following cases:

- During development of an application
- During system operation

The following table describes the mapping between the applicable stages for JSP pre-compilation and the commands used:

Table 2-10: Mapping between the applicable stages for JSP pre-compilation and the commands used

Applicable stages			Commands used	Reference
Application development	When a Web application is developed		`cjjspc` command	*(1)*
System operation	When J2EE applications are started		`cjstartapp` command	*(2)*
	When J2EE applications are switched	Normal switching	`cjstartapp` command	
		Switching by reloading	`cjjspc` command	
		Switching by redeploying	`cjjspc` command	

The stages in which the JSP pre-compilation is used are explained below. For an overview of the commands used, see *2.5.2 Methods for performing JSP pre-compilation*.

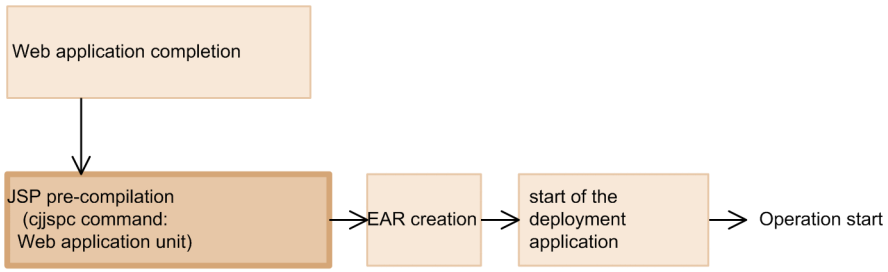
Note that the JSP compilation is executed during the execution of the `cjstartapp` command if you specify the option for performing JSP pre-compilation by the `cjstartapp` command. Because this process is performed in the J2EE server, the memory usage of the J2EE server process increases temporarily.

(1) Usage during development of an application

During the development of a Web application, execute the JSP pre-compilation functionality after the completion of the Web application.

Use the `cjjspc` command for implementing the JSP pre-compilation. The following figure shows an example of application of JSP pre-compilation during development of a Web application:

Figure 2-5: Example of applying the JSP pre-compilation functionality when developing a Web application



Use the JSP pre-compilation functionality to compile all the JSP files included in a completed Web application at the same time. As a result, you can improve the response time of JSP initial request during execution of a Web application.

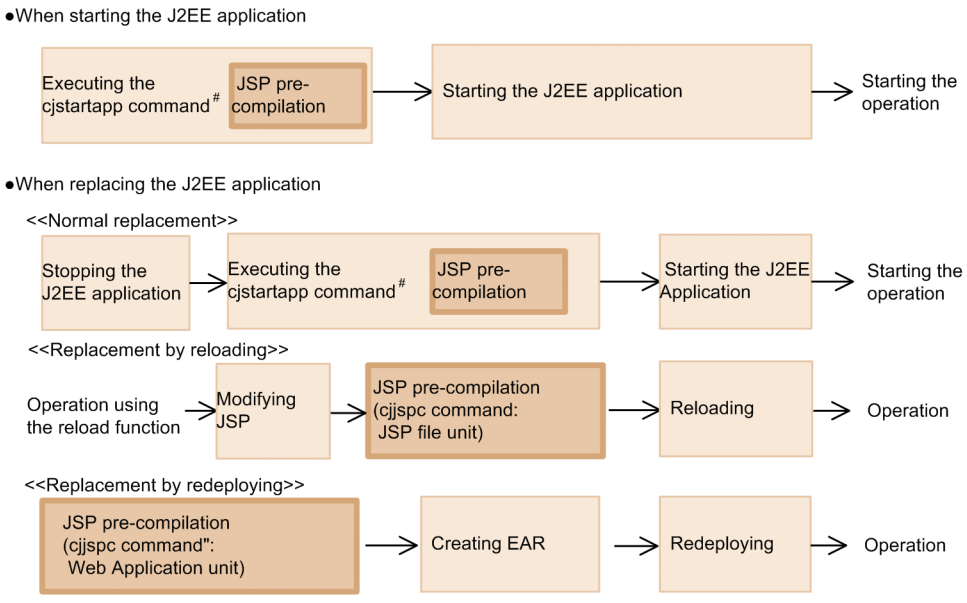
To compile all the JSP files in a Web application collectively, implement the 'Pre-compilation in each Web application method of the `cjjspc` command.

For the `cjjspc` command, see *cjjspc (JSP pre-compilation)* in the *uCosminexus Application Server Command Reference Guide*.

(2) Usage during system operation

To improve the response time of the JSP initial request during system operation, implement the JSP pre-compilation before starting the operation. Implement the JSP pre-compilation during system operation when starting a J2EE application and when switching the J2EE applications. The following figure shows an example of application of JSP pre-compilation during system operation:

Figure 2-6: Applying the JSP pre-compilation functionality during system operations



If you specify the option for JSP pre-compilation, both the JSP pre-compilation and the J2EE Application startup are implemented on executing the cjstartapp command.

The overview of each stage applicable to the JSP pre-compilation functionality is explained below. For details on the compilation methods using the JSP pre-compilation functionality during system operations, see *5.6.3 Substituting and maintaining J2EE applications* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

(a) When J2EE applications are started

When a J2EE application is started, you can use the JSP pre-compilation for an already imported J2EE application. Execute the JSP pre-compilation functionality to be implemented when a J2EE application is started by specifying the option for implementing the JSP pre-compilation in the `cjstartapp` command. If you execute this command, the JSP files included in the J2EE application are complied together before starting the J2EE application.

(b) When J2EE applications are switched

You can use the JSP pre-compilation functionality before switching the J2EE applications. Note that the method of JSP pre-compilation depends upon how the J2EE applications are switched.

- **In the case of normal switching between J2EE applications**

  Normal switching between J2EE applications refers to a method in which the J2EE application is stopped once and then switched to a new application.

  In the case of normal switching between J2EE applications, execute the JSP pre-compilation by specifying the option for implementing the JSP pre-compilation in the `cjstartapp` command, after importing the switched J2EE application. If this command is executed, the JSP files included in the J2EE application are compiled collectively before the switched J2EE application is started.

- **In the case of switching the J2EE applications by reloading**

  Switching the J2EE applications by reloading refers to a method in which a J2EE application is switched to a new J2EE application (an application in expanded format) without stopping.

  In the case you use the reload functionality to operate a J2EE application containing the JSP compilation results, and if the JSP files are modified, you can compile the modified JSP files by the `cjjspc` command.

- **In the case of switching the J2EE applications by redeploying**

  Switching the J2EE application by redeploying refers to a method in which a J2EE application is switched to a new J2EE application (an application in archive format) without stopping.

  The order to switch the J2EE applications by redeploying is as follows:

  1. Implement the JSP pre-compilation with the `cjjspc` command and compile all the JSP files included in the J2EE application collectively.
  2. Create an EAR file containing the JSP compilation results generated in 1.
  3. Redeploy the EAR file created in 2.

For an overview of switching J2EE applications, see *5.6.3 Substituting and maintaining J2EE applications* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*. For reloading, see *13.8 Detecting updates and reloading J2EE applications* in the *uCosminexus Application Server Common Container Functionality Guide*.

2.5.4 Processing during execution of JSP pre-compilation

This subsection explains the checks implemented during execution of JSP pre-compilation functionality and the operation of a J2EE application in which the JSP pre-compilation functionality is executed.

(1) Checks implemented during JSP pre-compilation

When the JSP pre-compilation is executed, validity check of `web.xml` and version check of the JSP compilation results are implemented.

(a) Validity check of web.xml

With the JSP pre-compilation functionality, whether the `web.xml` conforms to the DTD or XML schema is verified before execution of compilation processing. Moreover, the validity of settings for the elements referenced during JSP pre-compilation is also verified to the extent necessary for JSP pre-compilation. If the elements do not conform to the schema, an error occurs during the translation of JSP that generates a Java file from the JSP file.

The following table describes the elements of `web.xml` verified during execution of JSP pre-compile.

Table 2-11: Elements of web.xml verified during execution of JSP pre-compilation

Tag names	Tag description	Servlet version				
		2.2	2.3	2.4	2.5	3.0
<!DOCTYPE>	DOCTYPE declaration	Y	Y	N	N	N
<web-app>	Root tag	Y	Y	Y	Y	Y
<servlet>	Definitions about the servlet	Y	Y	Y	Y	Y
<jsp-file>	JSP file name	Y	Y	Y	Y	Y
<taglib>	Definitions about the tag library	Y	Y	--	--	--
<taglib-uri>	URI of tag library	Y	Y	--	--	--
<taglib-location>	Location of tag library descriptor file (TLD)	Y	Y	--	--	--
<jsp-config>	Definitions about the JSP	--	--	Y	Y	Y
<taglib>	Definitions about the tag library	--	--	Y	Y	Y
<taglib-uri>	URI of tag library	--	--	Y	Y	Y
<taglib-location>	Location of tag library descriptor file (TLD)	--	--	Y	Y	Y
<jsp-property-group>	Settings of JSP that matches with the specified URL pattern	--	--	Y	Y	Y
<url-pattern>	URL pattern of JSP for which installation is applied	--	--	Y	Y	Y
<el-ignored>	Settings indicating whether to ignore EL (Expression Language)	--	--	Y	Y	Y
<scripting-invalid>	Settings indicating whether to disable the scripting element	--	--	Y	Y	Y
<page-encoding>	Page encoding name	--	--	Y	Y	Y
<include-prelude>	File included as the JSP header	--	--	Y	Y	Y
<include-coda>	File included as the JSP footer	--	--	Y	Y	Y
<is-xml>	Settings indicating whether coding is done in XML format	--	--	Y	Y	Y
<deferred-syntax-allowed-as-literal>	Settings indicating whether the presence of #{ string in the part where EL is not used is to be considered as an error	--	--	--	Y	Y
<trim-directive-whitespaces>	Settings indicating whether to output extra spaces from JSP	--	--	--	Y	Y

Legend:

Y: Verified

N: Not verified

--: Unsupported element

(b) Version check of JSP compilation results

When using the JSP pre-compilation functionality, check if the version of the Web application specified in `web.xml` for the J2EE server matches with the version of JSP during JSP compilation. The version check is implemented in the following cases:

- If the implementation of JSP pre-compilation is not specified when application is started (when the application is started with the `cjstartapp` command, without specifying the `-jspc` option)
- If JSP pre-compilation is executed in each Web application with the `cjjspc` command by specifying the files excluded from the compilation
- If JSP pre-compilation is executed in each JSP file using the `cjjspc` command

The class files generated from JSP depend on the version of the Web application specified in `web.xml`. You cannot use the class files in a Web application whose version is different from the version of the Web application during execution of JSP pre-compilation. Consequently, if the version of a Web application is changed, you need to compile all the JSP files.

Note that in the following cases, since all JSP files included in a Web application are compiled, the JSP compilation result is not checked.

- If the JSP pre-compilation is executed in each Web application without specifying the files to be excluded from compilation
- If the JSP pre-compilation used when starting the application is executed

Reference note

When version check of JSP compilation results is implemented, a version information file containing the version information of the JSP files is generated in the JSP working directory of the Web application to be compiled. The version information file is generated at the following location:

Web-application-directory`/WEB-INF/`*JSP-work-directory-name*`/WEB-INF/`*JSP-work-directory-name*`/jsp_compile_info`

(c) TLD file check

You validate whether the TLD file complies with DTD or XML schema during the JSP pre-compilation. The following is the description for TLD file check related to each version of Web applications:

- When the version of Web application is 2.4 or later

  Validation is executed by default. Note that if the TLD file is not in accordance with the schema, an error will occur during the translation of the JSP file.
- When the version of Web application is 2.3 or earlier

  Specify in advance whether validation is to be performed. If you specify settings to validate the TLD file, the TLD file will be validated during the JSP pre-compilation.

  For details on the validation of TLD files, see *2.5.8 Execution environment settings (J2EE server settings)*.

(2) Handling JSP files in application in which the JSP pre-compilation functionality is implemented

This subsection explains the operation of a J2EE application in which the JSP pre-compilation functionality is executed.

(a) Operations when a request is executed and a J2EE application is started

When JSP pre-compilation is being implemented, JSP compilation is not implemented during execution of a request. The class files of JSP created during pre-compilation are loaded and executed.

In such a case, an error occurs if the class files compiled from the JSP files do not exist. The following table describes the behavior of the J2EE server when JSP pre-compilation is implemented and the files do not exist:

Table 2-12: Behavior of the J2EE server when the files do not exist (when JSP pre-compilation is executed)

Non-existent files		Behavior of the J2EE server
JSP files	JSP files	JSP file is not referenced
	Class file	Error 404 is returned
Tag files	Tag files	Tag file is not referenced
	Class file	Error 500 is returned (`java.lang.NoClassDefFoundError` occurs)
Statically included file		Statically included file is not referenced
TLD file		TLD file is not referenced

The J2EE server operates as follows, when the pre-compilation is not implemented and the files do not exist:

Table 2-13: Behavior of the J2EE server when the files do not exist (when JSP pre-compilation is not executed)

Non-existent files		Behavior of the J2EE server
JSP files	JSP files	Error 404 is returned
	Class file	JSP files are compiled
Tag files	Tag files	Error 500 is returned (compilation error)
	Class file	Tag files are compiled
Statically included file		Error 500 is returned (compilation error)
TLD file		

(b) Operation when a J2EE application is started

If you pre-compile the JSP files of a Web application in which `<load-on-startup>` is specified in the JSP files with `web.xml`, JSP compilation is not implemented when a J2EE application is started. The class files generated during JSP pre-compilation are loaded and `jspInit` method is executed. In such a case, loading of JSP files fails, if the class files of JSP, or a class file on which JSP depends does not exist.

Note that if the settings for error notification in servlet and JSP are enabled, the operation fails when starting a Web application. For details on the settings for error notification in servlets and JSPs, see *9.16 Settings for error notification in servlets and JSPs* in the *uCosminexus Application Server Application Setup Guide*.

(3) Notes on JSP pre-compilation functionality

The notes related to the JSP pre-compilation functionality are as follows:

- **Sending requests in which `jsp_precompile` is added**

  Even if you send a request in which the query string `jsp_precompile` or `jsp_precompile=true` was added to an application that executes the JSP pre-compilation, the JSP compilation is not executed.

- **Operation of the same Web application by multiple invocations of the command for JSP pre-compilation**

  You cannot execute JSP pre-compilation in the same Web application by multiple invocations of the `cjjspc` command. Also, you cannot execute compilation in the same Web application with the `cjjspc` command, during execution of JSP pre-compilation when an application is started.

  Note that due to the command locking process, a lock file will be generated in the JSP work directory during the execution of JSP pre-compilation functionality. The lock file is generated in the following location:
  Web-application-directory`/WEB-INF/`*JSP-work-directory-name*`/WEB-INF/`*JSP-work-directory-name*`/ExecutingJspPrecompilation.lock`

- **Migrating to applications that use the JSP compilation results**

When the J2EE application is in the archive format, the compilation results generated by the JSP pre-compilation functionality during the startup of the application will be deleted when the application is stopped.

The following is the procedure for using the JSP compilation results that are generated by the JSP pre-compilation functionality during the startup of the application, even after the application stops. The description is given for each J2EE application format.

In the case of a J2EE application of archive format

1. Execute the JSP pre-compilation when starting the application.
2. Export the application.
3. Replace with applications containing the JSP compilation results by using the `redeploy` functionality.

In the case of a J2EE application of exploded archive format

1. Execute JSP pre-compilation when the `cjjspc` command or application starts.

- **Migrating to applications that do not use the JSP compilation results**

  When you do not use the JSP compilation results generated by JSP pre-compilation, you need to delete each JSP working directory.

  The procedure to be followed when you do not use the JSP compilation results is explained below for each format of J2EE application:

  In the case of a J2EE application of archive format

  1. Export the J2EE application.
  2. Deploy the EAR file.
  3. Delete each JSP working directory under *Web-application-root-directory*`/WEB-INF`.
  4. Create the EAR file.
  5. Switch the J2EE applications using the redeploy functionality.

  In the case of a J2EE application of exploded archive format

  1. Stop the J2EE application.
  2. Delete each JSP working directory under *Web-application-root-directory*`/WEB-INF`.
  3. Start the J2EE application.

- **Modifying files on which the JSP file depends**

  When a tag file, static included file, or TLD file is updated, compile all JSP files that reference the updated files.

- **Updating the exploded archive format J2EE applications that use the JSP pre-compilation functionality**

  Note the followings when you update the J2EE applications in exploded archive formats that use the JSP pre-compilation functionality:

  - To add the JSP file or tag file into the J2EE application

    Compile all the JSP files that reference the added JSP file or tag file. Use the JSP pre-compilation functionality to compile the JSP file.

  - To apply the JSP compilation results updated in the development environment to the execution environment

    Copy the class files exist under the JSP work directory of the J2EE application in the development environment to the JSP work directory of the J2EE application in the execution environment. In this case, copy all the class files updated during the JSP pre-compilation in the development environment.

2.5.5 Lifecycle and output destination of JSP compilation results

JSP is compiled in a Web container, and the Java source files and class files are generated. With a Web container, you can specify whether to maintain the Java source files and class files that serve as JSP compilation results, when a J2EE server is restarted. This subsection describes the settings for maintaining the compilation results of the JSP file.

This subsection explains the lifecycle and the output destination of the JSP compilation results when you use the JSP pre-compilation functionality.

(1) Lifecycle of JSP compilation results

The lifecycle of the JSP compilation results when you use the JSP pre-compilation functionality is explained below:

Generating the compilation results

In the case you use the JSP pre-compilation functionality, compilation results are generated when:

- The `cjjspc` command is executed
- The Web application is started by specifying -jspc option in the `cjstartapp` command

Deleting the compilation results

For J2EE applications in the archive format, the compilation results generated by JSP pre-compilation functionality during the startup of the application will be deleted, when the application stops.

(2) Output destination of JSP compilation results

When you implement the JSP pre-compilation functionality, the JSP working directory is created, and the JSP compilation results are output to the JSP working directory. If, however, the JSP files to be compiled do not exist, the JSP working directory is not created. The following files are output:

1. Java source files# generated from JSP files
2. Class file that compiles the Java source files mentioned in 1.
3. Java source files# generated from tag files
4. Class file that compiles the Java source files mentioned in 3.

#

When implementing the JSP pre-compilation functionality, you can specify whether to save the Java source files.

The default output destination and the configuration of the output destination directory are explained below: For details on the names of the output classes, see *2.5.7 Class names in JSP compilation results*.

(a) Default output destination

When you execute the JSP pre-compilation functionality, the compilation results are output to the JSP working directory. The default JSP working directory is at the following location:

In Windows

Web-application-web-inf-directory`\cosminexus_jsp_work`

In UNIX

Web-application-web-inf-directory`/cosminexus_jsp_work`

If the WEB-INF directory does not exist, the WEB-INF directory and JSP work directory are automatically created when the JSP work directory is created.

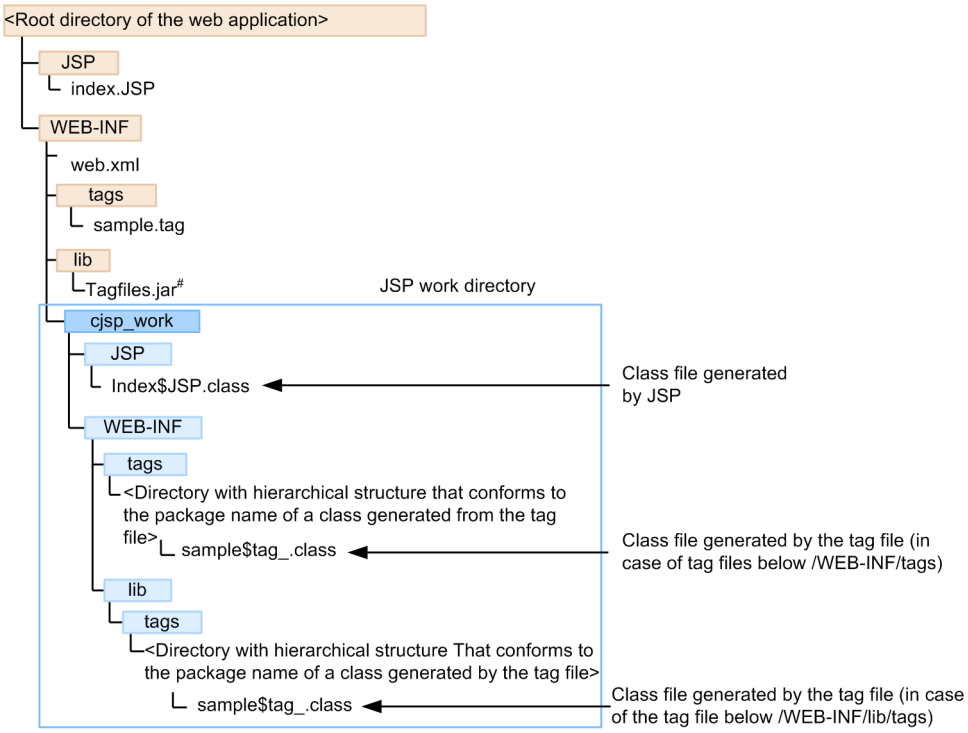
Note that the default value is set for the name of the JSP working directory. However, if necessary, you can change the name. For details on changing the name of the JSP work directory during the JSP pre-compilation by the `cjjspc` command, see *cjjspc (JSP pre-compilation)* in the *uCosminexus Application Server Command Reference Guide*. For details on changing the name of the JSP work directory during the JSP pre-compilation when the J2EE application is started by the `cjstartapp` command, see *2.5.8 Execution environment settings (J2EE server settings)*.

Also, when you change the name of the JSP work directory, you must specify the name of the changed JSP work directory in the parameter `webserver.jsp.precompile.jsp_work_dir` within the `<configuration>` tag of the logical J2EE server (j2ee-server) of the Easy Setup definition file. For details on the Easy Setup definition file and the parameters to be specified, see *4.6 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

(b) Configuration of output destination directory

The JSP compilation results are output to the JSP working directory. The following figure shows the configuration of the output destination directory for JSP compilation results. Note that in the following figure, default directory name is used for the JSP working directory.

Figure 2-7: Configuration of output destination directory for JSP compilation results (When JSP pre-compilation is executed)



Use META-INF/tags/sample/sample.tag as the tag file

The directory configuration is explained below:

- The package name of the classes generated from the tag files is in the following format:

  In the case of a tag file under `WEB-INF/tags`

  `org.apache.jsp.tag.web.`*path-under-*`/WEB-INF/tags-`*directory*

  In the case of a tag file included in the jar file under `WEB-INF/lib`

  `org.apache.jsp.tag.meta.`*string-with-encoded-jar-file-name*`.`*path-under-*`META-INF/tags-`*directory-in-jar-file*

- The path length restrictions for output destination directory of the class files generated from the JSP files and the tag files depend on the path length upper limit in the OS. If the path length exceeds the upper limit in the OS, change the name of the JSP working directory.

  ! Important note

  If you specify the same JSP work directory in multiple J2EE servers, the status of the JSP work directory might become invalid when you deploy an application with the same context root.

2.5.6 JSP Compilation results when JSP pre-compilation functionality is not used

When you do not use the JSP pre-compilation functionality, the JSP files are compiled during initial access to a JSP file. This subsection describes the JSP compilation results and the method of changing the output destination of the compilation results, when the JSP pre-compilation functionality is not used.

(1) Lifecycle of JSP compilation results

The lifecycle of the JSP compilation results when you do not use the JSP pre-compilation functionality is explained below:

Generating the compilation results

In the case you do not compile the JSP files beforehand by the JSP pre-compilation functionality, the JSP compilation results are generated when:

- The JSP is first accessed
- The Web application in which `<load-on-startup>` is specified for JSP in DD (`web.xml`) is started

Deleting the compilation results

The JSP compilation results are deleted when:

- The J2EE application is un-deployed
- The J2EE server is started#
- The J2EE server is stopped#

#

If the setting specifies that the JSP compilation results are not to be maintained, the JSP compilation results are deleted. When the J2EE server is started, the compilation results are deleted to prepare for the forced termination of the server.

(2) Maintaining the compilation results of JSP files

If you do not perform JSP pre-compilation, with a Web container, you can specify whether to maintain the Java source files and class files that serve as JSP compilation results, when a J2EE server is restarted.

Customize the properties of the J2EE server to specify the settings for maintaining the compilation results of JSP files. For details on customizing the settings of the J2EE server operations, see *2.5.8 Execution environment settings (J2EE server settings)*.

! Important note

Points to be noted when a Web application is un-deployed

By default, the setting is specified to maintain the JSP compilation results. Even if you specify the settings to maintain the JSP compilation results, the JSP compilation results are deleted if a Web application is un-deployed. Consequently, the user need not delete the JSP compilation results when a server is restarted. After running a Web container by specifying the settings to maintain the JSP compilation results, if you do not need the JSP compilation results, un-deploy the J2EE application.

Hitachi recommends that you specify the settings to maintain the JSP compilation results.

---

(3) Output destination of JSP compilation results

When you do not implement the JSP pre-compilation functionality, the JSP compilation results are output to the temporary directory for JSP.

The following files are output:

1. Java source files generated from JSP files
2. Class file that compiles the Java source files mentioned in 1.
3. Java source files generated from tag files
4. Class file that compiles the Java source files mentioned in 3.

The default output destination and the configuration of the output destination directory are explained below:

For details on the output class names, see *2.5.7 Class names in JSP compilation results*.

(a) Default output destination

When you do not execute the JSP pre-compilation functionality, the JSP compilation results are output to the directory for each Web application created under the temporary directory for JSP. The default temporary directory for JSP is at the following location:

In Windows

Cosminexus-installation-directory\CC\server\repository*server-name*\web

In UNIX

/opt/Cosminexus/CC/server/repository/*server-name*/web

Note that the default value is set for the temporary directory for JSP. However, if necessary, you can change the default value. For details on changing the temporary directory for JSP, see *2.5.8 Execution environment settings (J2EE server settings)*.

The directory will be created for each Web application under the temporary directory for JSP and the JSP compilation results that exist in the relevant Web application will be output.

Note that the directory for the Web applications has the directory name based on the context root name. If the context root name includes a slash (/), dollar sign ($), percent sign (%), and plus sign (+), the characters will be converted to the following characters:

Characters before conversion	Characters after conversion
/	$2f
$	$24
%	$25
+	$2b

Example:

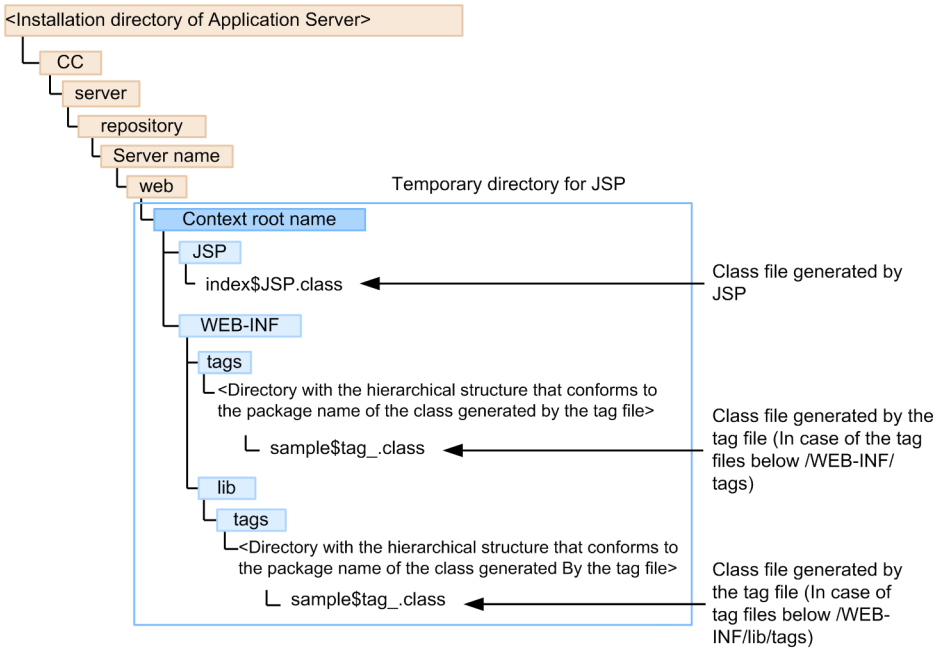
When the temporary directory for JSP is the default directory and the context root name is J2EE_AP1/WEB_AP1_war, the output destination for the JSP compilation results of the relevant Web applications will be as follows:

- **In Windows**
  Cosminexus-installation-directory\CC\server\repository*server-name*\web\J2EE_AP1$2fWEB_AP1_war
- **In UNIX**
  /opt/Cosminexus/CC/server/repository/*server-name*/web/J2EE_AP1$2fWEB_AP1_war

(b) Configuration of output destination directory

The following figure shows the configuration of the output destination directory for JSP compilation results.

Figure 2-8: Configuration of output destination directory for JSP compilation results (When JSP pre-compilation is not executed)



The directory configuration is explained below:

- The package name of the classes generated from the tag files is in the following format:

  In the case of a tag file under `WEB-INF/tags`

  `org.apache.jsp.tag.web.`*path-under-*`/WEB-INF/tags-`*directory*

  In the case of a tag file included in the jar file under `WEB-INF/lib`

  `org.apache.jsp.tag.meta.`*string-with-encoded-jar-file-name*`.`*path-under*`-META-INF/tags-`*directory-in-jar-file*
- The path length restrictions for output destination directory of the class files generated from the JSP files and the tag files depend on the path length upper limit in the OS. If the path length exceeds the upper limit in the OS, change the name of the JSP working directory.

2.5.7 Class names in JSP compilation results

This subsection describes the format and the conversion rules of the class names generated from the JSP file or tag file.

(1) Class name format

The class name format of the classes generated from JSP files or tag files depend on the file type and whether the JSP debug functionality is enabled. The following table lists the class name formats:

Table 2-14: Class name format of classes generated from JSP files or tag files

File type	When the JSP debug functionality is disabled	When the JSP debug functionality is enabled
JSP files	*file-name*	*file-name*`_jsp`
Tag files	*file-name*`_`	*file-name*`_tag`

The rules specified in *(2) Class name conversion rules* are applied to *file-name*.

(2) Class name conversion rules

If the name of the class generated from the JSP file or tag file contains characters that cannot be used as class name, underscore (_), or dollar sign ($), the following conversion rules are applied sequentially:

1. If the first character is the package name and a character that cannot be used at the beginning of the class name, add a dollar sign ($) at the beginning.
2. Convert a period (.) in the middle of the file name into a dollar sign ($).
3. For the characters that cannot be used as the class name, convert the underscore (_) and the characters that cannot be used into a string of 4-digit hexadecimal expression.
   The alphabetical characters used in the hexadecimal expression will be in the lower-case.

The conversion rule in step 1 is applied only to the first character. The conversion rules in step 2 and 3 are applied sequentially from the beginning of the string.

The following table describes the conversion examples for names of classes generated from JSP files or tag files:

Table 2-15: Conversion examples for names of classes generated from JSP files or tag files

File type	File name	JSP debug functionality	Class name
JSP files	index.jsp	--	index$jsp
		Y	index$jsp_jsp
	10test-10.jspx	--	$10test_002d10$jspx
		Y	$10test_002d10$jspx_jsp
	test.tag	--	test$tag
		Y	test$tag_jsp
Tag files	tagfile1.tag	--	tagfile1$tag_
		Y	tagfile1$tag_tag
	Tag_File$10.tagx	--	Tag_005fFile_002410$tagx_
		Y	Tag_005fFile_002410$tagx_tag

Legend:
Y: Valid
--: Invalid

2.5.8 Execution environment settings (J2EE server settings)

To use the JSP pre-compilation functionality or to maintain the compilation results, you must specify the J2EE server settings.

Implement the J2EE server settings in the Easy Setup definition file. Specify the definition for using the JSP pre-compilation functionality or for maintaining the compilation results within the <configuration> tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file.

The following table lists the definitions of the Easy Setup definition file for using the JSP pre-compilation functionality or maintaining the compilation results:

Table 2-16: Definitions of the Easy Setup definition file for using the JSP pre-compilation functionality or maintaining the compilation results

Items	Parameter to be specified	Setting contents
JSP pre-compile	webserver.jsp.additional.import.list	Specify the class name (fully-qualified class name or "package name.*") to be imported implicitly during JSP compilation.

Items	Parameter to be specified	Setting contents
JSP pre-compile	webserver.jsp.keepgenerated	You can specify whether to save the Java source files generated from the JSP file.
	webserver.jsp.compile.backcompat	Specify the version of the Java language specifications used for compiling the Java source files. Note that if the version of the Java source files generated from the JSP file by the Web application is different, specify the version for each Web application and implement JSP pre-compile.
	webserver.jsp.precompile.jsp_work_dir	Specify the directory to output the compilation results when the JSP pre-compilation functionality is implemented.
	webserver.xml.validate	Specify whether to validate the TLD file during the execution of the JSP pre-compilation functionality, when the version of servlets included in the Web application is earlier than the version 2.3. Note that you specify this parameter when the Web application version is earlier than the version 2.3. If the Web application has the version 2.4 or later versions, the validation is implemented by default, so you cannot specify a value in this parameter.
Save the compilation result of the JSP file	webserver.work.clean	Specify whether you want to maintain the compilation results.
	webserver.work.directory	Specify the output destination (temporary directory for JSP) for the compilation results. Specify this parameter to change the default output destination#. For details on the JSP working directory, see *2.5.6(3) Output destination of JSP compilation results*. Note that this is the output destination when you do not execute the JSP pre-compilation functionality.

#

To change the settings for the temporary directory of JSP

After operating the Web container with settings to maintain the JSP compilation results, if you change the settings for the temporary directory of JSP, the old temporary directory of JSP will not be deleted. Therefore, you must delete the temporary directory manually.

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

2.6 Functionality for setting up the default character encoding

This section describes the functionality for setting up the default character encoding.

With Application Server, you can set up the character encoding that is compliant with the Servlet specifications and is unique to Application Server.

The following table describes the organization of this section.

Table 2-17: Organization of this section (Functionality for setting the default character encoding)

Category	Title	Reference
Description	Units for setting the default character encoding	*2.6.1*
	Applicable locations and conditions for default character encoding	*2.6.2*
	Application of character encoding during JSP pre-compilation	*2.6.3*
	Specifiable character encoding	*2.6.4*
Implementation	Implementation of default character encoding (for Servlet specifications)	*2.6.5*
	Definition in the DD	*2.6.6*
Settings	Execution environment settings	*2.6.7*
Notes	Precautions related to default character encoding	*2.6.8*

Note:
There is no specific explanation of *Operations* for this functionality.

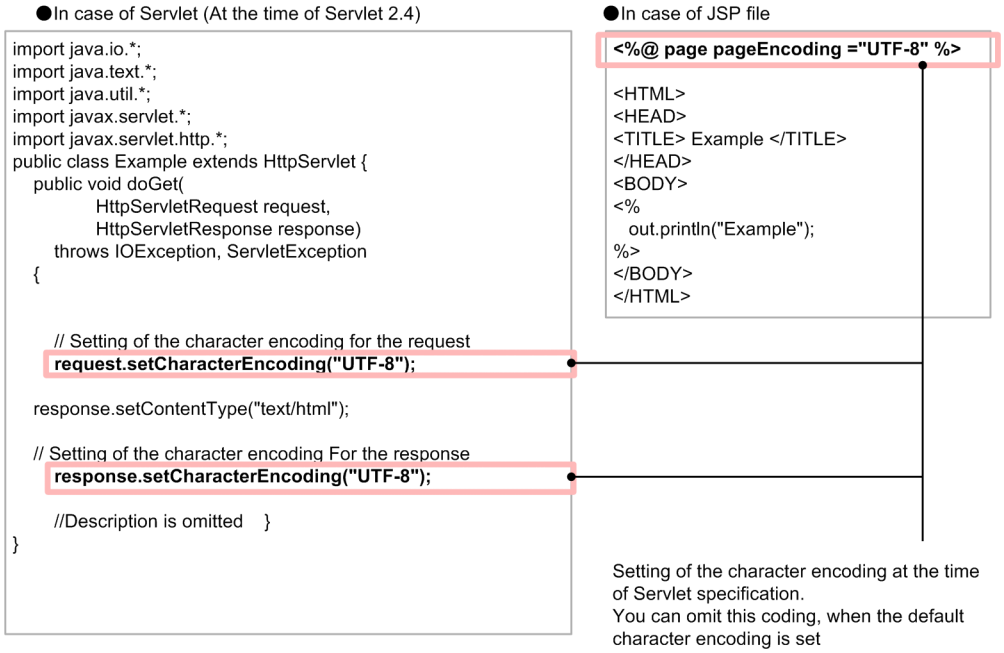
When specifying settings for character encoding used for request decoding and response encoding compliant with the Servlet specifications, you describe the settings for each servlet and JSP file. With Application Server, you can specify default character encoding using the following methods apart from the settings compliant with the Servlet specifications:

- Setting up character encoding to be used in J2EE servers
- Setting up character encoding to be used in Web applications

This enables you to omit the character encoding settings described in the servlet and JSP file during the development of Web applications. Also, you can easily unify the character encoding for the J2EE servers or Web applications.

The following figure shows the character encoding settings that you can omit in the Servlet specifications:

Figure 2-9: Character encoding settings that can be omitted in the Servlet specifications



The following subsection describes the default character encoding settings.

2.6.1 Units for setting the default character encoding

With Application Server, you can specify settings for request decoding, response encoding, and default character encoding used in JSP files of each J2EE server and Web application.

This subsection describes the default character encoding settings. You can also apply the default character encoding during the execution of JSP pre-compile. For details on the default character encoding settings during the JSP pre-compilation, see *2.6.3 Application of character encoding during JSP pre-compilation*.

(1) Settings for each J2EE server

You set up default character encoding for each J2EE server. If you specify settings for default character encoding for each J2EE server, the specified character encoding will be applied to the servlets and JSP files of all the J2EE applications deployed on the J2EE server. This enables you to unify the character encoding for J2EE servers.

In the case of the J2EE servers, the default character encoding will be set up when you customize the J2EE server operation settings. For details on the settings, see *2.6.7 Execution environment settings*.

(2) Settings for each Web application

You set up the default character encoding for each WAR file. When you specify settings for default character encoding for each WAR file, the specified character encoding will be applied to the servlets and JSP files included in the WAR file. This enables you to unify the character encoding for Web applications.

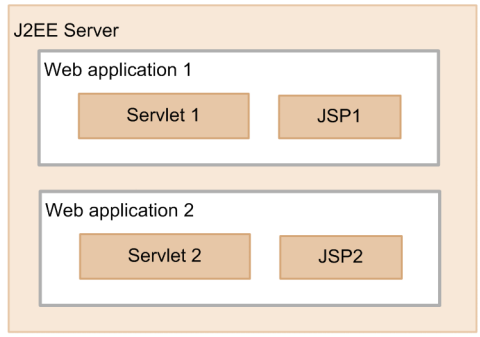
In the case of the Web application settings, the default character encoding will be set up when you define the J2EE application properties. For details on the settings, see *2.6.7 Execution environment settings*.

(3) Operations when character encoding is specified with multiple ranges

Apart from the character encoding settings for each J2EE server or Web application, you can also set up the character encoding defined in the Servlet specifications. For the character encoding settings in the Servlet specifications, the settings are specified for each servlet or JSP file.

The following figure shows the scope of settings:

Figure 2-10: Scope of character encoding settings



Legend:
: Applicability of the character encoding set in the Servlet/JSP file unit (Servlet specifications)
: Applicability of the character encoding set in the Web application unit
: Applicability of the character encoding set in the J2EE server unit

You can also specify the default character encoding with multiple ranges. For multiple setting, the default character encoding is applied in the following sequence:

1. Settings in each servlet or JSP file (Servlet specifications)
2. Settings in each Web application
3. Settings for each J2EE server

For example, as shown in Figure 2-10, the character encoding is set up in servlet 2 and J2EE server. In this case, the default character encoding set in each J2EE server is applied to all the applications in the J2EE server, but the character encoding set up in the servlet is applied only to the servlet 2.

The following table lists the valid settings for each combination of the character encoding settings:

Table 2-18: Combinations of character encoding settings and valid settings

Combination of settings			Valid settings
Settings in the servlets or JSP files (Servlet specifications)#	Settings in each Web application	Settings for each J2EE server	
Y	--	--	Settings in servlets or JSP files
Y	Y	--	
Y	--	Y	
Y	Y	Y	
--	Y	--	Settings in each Web application
--	Y	Y	
--	--	Y	Settings for each J2EE server
--	--	--	Character encoding defined in the Servlet specifications

Legend:
Y: Available
--: Unavailable

Note:
If the character encoding is not specified, the character encoding defined in the Servlet specifications is applied. For details, see *2.6.5(2) Character encoding defined in the Servlet specifications*.

#

In the JSP file or tag file with the XML syntax, if the encoding attribute is not specified in the XML declaration, the default encoding setup functionality is enabled.

2.6.2 Applicable locations and conditions for default character encoding

This subsection describes the applicable locations and the conditions for application of default character encoding specified for each J2EE server or Web application.

(1) Applicable locations

The default character encoding specified for each J2EE server or Web application is applied at the following locations:

- **Request decoding**
  Applied to the default character encoding used for decoding the request body and query.
- **Response encoding**
  Applied to the default character encoding used for encoding the response body and response Content-Type header.
- **JSP files**
  Applied to default character encoding of the JSP files.

(2) Applicable conditions

The default character encoding is applied if the character encoding defined in the Servlet specifications is not specified. For details on how to set up the character encoding defined in the Servlet specifications, see *2.6.5 Implementation of default character encoding (For Servlet specifications)*.

The following conditions are also applicable to the character encoding used for request decoding and response encoding:

(a) Applicable conditions for requests

The following conditions are applicable in the case of character encoding used for request decoding:

- The HTTP request header of the request sent from the client contains the charset parameter and does not contain the Content-Type header.

Moreover, the following conditions are applicable to the request body and query:

■ Request body

In the case of servlet

The character encoding is applied when the request POST data is read using one of the following methods:

- The request POST data is read in `BufferedReader` acquired by using the `getReader` method of `javax.servlet.ServletRequest`.
- The request POST data is read as a request parameter.

When the request POST data is read as a request parameter, the `getParameter` method, `getParameterMap` method, `getParameterName` method, and `getParameterValues` method of `javax.servlet.ServletRequest` will be used.

In the case of JSP files

The character encoding is applied when the request POST data is read using one of the following methods:

- The request POST data is read in `BufferedReader` acquired by using the `getReader` method of the implicit object request.
- The request POST data is read as a request parameter.

When the request POST data is read as a request parameter, the `getParameter` method, `getParameterMap` method, `getParameterName` method, and `getParameterValues` method of the implicit object request will be used or the implicit object param and paramValues in the Expression Language will be used.

■ Query

In the case of servlet

The character encoding is applied when the query is read as a request parameter in the method that uses the `getParameter` method, `getParameterMap` method, `getParameterName` method, and `getParameterValues` method of `javax.servlet.ServletRequest`.

In the case of JSP files

The character encoding is applied when the query is read as a request parameter in one of the following methods:

- The query is read as a request parameter by using the `getParameter` method, `getParameterMap` method, `getParameterName` method, and `getParameterValues` method of the implicit object request.
- The query is read as a request parameter by using the implicit object param and paramValues in the Expression Language.

(b) Applicable conditions for response

The conditions applicable to the character encoding used for response encoding are described separately for the response body and the character encoding name for the response Content-Type header.

■ Response body

In the case of servlet

The character encoding is applied when the response data is created using `PrintWriter` acquired with the `getWriter` method of `javax.servlet.ServletResponse`.

In the case of JSP files

The character encoding is applied when the response is output without acquiring the `ServletOutputStream` object by the `getOutputStream()` method of the implicit object response.#

#

When the `ServletOutputStream` object is acquired, all the output to the JSP body text or the implicit object `out` that does not use the `ServletOutputStream` object become runtime error. Therefore, this output cannot be output as response.

■ Character encoding name for the response Content-Type header

In the case of servlet

The character encoding is applied when MIME type beginning with `text/` is set and charset is not set in the response contents format.

In the case of JSP files

The character encoding is applied in one of the following cases:

- The response contents format is not set.
- MIME type beginning with `text/` is set and charset is not set.

In the case of the static contents

The character encoding is applied when the following conditions are fulfilled:

- The static contents extension is set as a target for default encoding setup functionality.
- The extension is set in the MIME type beginning with `text/`.
- Character encoding is not set for the response in servlet, JSP, or filter before the static contents are output.

Reference note

The contents format indicates the contents MIME type. You can include the character encoding in the contents format. The examples of contents format settings in the servlets and JSP files are as follows:

- In the case of servlet
  The `setContentType` method of `javax.servlet.ServletResponse` is used.
  Example of settings: `response.setContentType("text/html");`
- In the case of JSP files
  The `contentType` attribute of the `Page` directive is set up.
  Example of settings: `<%@ page contentType="text/plain" %>`

The examples of MIME type when the default character encoding settings are applied and when the default character encoding settings are not applied are as follows:

- Example when the settings are applied to the MIME type: `text/plain`, `text/html`
- Example when the settings are not applied to the MIME type: `image/gif`, `text/html;charset=UTF-8`

2.6.3 Application of character encoding during JSP pre-compilation

You can apply the default character encoding of the JSP file during the execution of the JSP pre-compilation functionality. If the default character encoding is applied during the execution of the JSP pre-compilation functionality, the default character encoding settings depend on the JSP pre-compilation method. The JSP pre-compilation methods have the following two types:

- JSP pre-compilation functionality executed during the application development
- JSP pre-compilation functionality executed during the startup of the J2EE application

The default character encoding settings is described for each JSP pre-compilation type. For details on the JSP pre-compilation functionality, see *2.5 JSP pre-compilation functionality and maintaining compilation results*.

For details on the specifiable character encoding, see *2.6.4 Specifiable character encoding* and for the applicable points and conditions of the specified default character encoding, see *2.6.2 Applicable locations and conditions for default character encoding*.

(1) JSP pre-compilation functionality executed during the application development (cjjspc command)

When JSP pre-compilation is executed using the `cjjspc` command, the default character encoding can be applied to JSP files or tag files. In the `cjjspc` command, you specify the default character encoding in the argument of the `cjjspc` command. As a result, the default character encoding specified in the command is applied to the JSP file that is executed when the `cjjspc` command is running. However, if the JSP file and tag file contain the character encoding defined in the Servlet specifications, the default character encoding settings will not be applied.

For details on settings, see *cjjspc (JSP pre-compilation)* in the *uCosminexus Application Server Command Reference Guide*.

! Important note

The default character encoding used for request decoding and response encoding is not applied during the compilation of JSP files, and therefore, you cannot specify the settings in the `cjjspc` command.

---

(2) JSP pre-compilation functionality executed during the startup of the J2EE application (cjstartapp command)

When JSP pre-compilation is executed with the startup of the J2EE applications using the `cjstartapp` command, you can apply the default character encoding to JSP files or tag files. The method of setting up the default character encoding is same for J2EE servers and Web applications. For J2EE servers, the default character encoding is set up for each J2EE server during the J2EE server operation settings. For Web applications, the default character encoding is set up in each WAR file during the Web application development. Execute the `cjstartapp` command to apply the specified default character encoding.

For details on the settings for each J2EE server and Web application, see *2.6.1 Units for setting the default character encoding*. Also, for details on settings, see *2.6.7 Execution environment settings*.

2.6.4 Specifiable character encoding

The characters that you can specify as default character encoding is the character encoding supported in JavaVM. For details on the character encoding supported in JavaVM, see the description related to the supported encoding in the JDK documentation.

The strings you can specify are the character encoding mentioned in the canonical name for `java.nio` API and the canonical name for `java.lang` API and their optional names.

! Important note

If the OS of the development and operation environments of the J2EE applications are different and when the J2EE applications are exported and imported using EAR files containing the runtime information, specify a character encoding that is supported in both the OS of the development environment and the OS of the operation environment. If the character encoding set up in the development environment is not supported in the operation environment, an exception might occur when the application starts.

Furthermore, whether the specified default character encoding is supported in JavaVM or not will be validated. The timing for validation depends on the method used for setting up the default character encoding. The following table describes the validation timing for character encoding:

Table 2-19: Validation timing for character encoding

Validation timing	Operations when unsupported character encoding is specified
When J2EE server starts	A warning message will be output and the J2EE server starting process continues. The set up character encoding is ignored.
When the server management command (`cjsetappprop`) is executed	An error message will be output and the processing of the server management command will be cancelled.
When the `cjjspc` command is executed#	An error message will be output and the processing of the command will terminate.

#

Validation timing when the default character encoding is set up during the JSP pre-compilation with the `cjjspc` command. For details on the application of default character encoding during the JSP pre-compilation, see *2.6.3 Application of character encoding during JSP pre-compilation*.

2.6.5 Implementation of default character encoding (For Servlet specifications)

In the locations where the character encoding settings defined in the Servlet specifications exist, the default character encoding set up on Application Server is disabled.

This subsection describes the character encoding settings defined in the Servlet specifications. Note that the character encoding settings depend on the version of the Servlet specifications.

(1) Character encoding setting method defined in the Servlet specifications

The following table describes the character encoding setting method defined in the Servlet specifications for each Servlet/JSP version:

Table 2-20: Character encoding setting method defined in the Servlet specifications (Servlet 2.5, 3.0/JSP 2.1)

Setting contents	Setting location	Setting method in the Servlet specifications
Character encoding of request	Servlet	`ServletRequest.setCharacterEncoding(java.lang.String` *env*`)`#1
	JSP files	None

Setting contents	Setting location	Setting method in the Servlet specifications
Character encoding of response	Servlet	• `ServletResponse.setCharacterEncoding(java.lang.String` *charset*`)` #1 • `ServletResponse.setContentType(java.lang.String` *type*`)` #1 • `ServletResponse.setLocale(java.util.Locale` *loc*`)` #1
	JSP files	• `contentType` attribute value of the `Page` directive (including charset)#2 • `pageEncoding` attribute of the `Page` directive#3 • `page-encoding` element of `web.xml`#2
Character encoding of JSP file	JSP files	• BOM#3 • `contentType` attribute value of the `Page` directive (including charset)#2 • `pageEncoding` attribute of the `Page` directive or `Tag` directive#3 • `page-encoding` element of `web.xml`#2 • `encoding` attribute of the XML declaration#4

#1
Package is `javax.servlet`.

#2
Method set up in the JSP page.

#3
Method set up in the JSP page or in the standard format tag file.

#4
Method set up in the JSP document or XML tag file.

Table 2-21: Character encoding setting method defined in the Servlet specifications (Servlet 2.4/JSP 2.0)

Setting contents	Setting location	Setting method in the Servlet specifications
Character encoding of request	Servlet	`ServletRequest.setCharacterEncoding(java.lang.String` *env*`)` #1
	JSP files	None
Character encoding of response	Servlet	• `ServletResponse.setCharacterEncoding(java.lang.String` *charset*`)` #1 • `ServletResponse.setContentType(java.lang.String` *type*`)` #1 • `ServletResponse.setLocale(java.util.Locale` *loc*`)` #1
	JSP files	• `contentType` attribute value of the `Page` directive (including charset)#2 • `pageEncoding` attribute of the `Page` directive#3 • `page-encoding` element of `web.xml`#2
Character encoding of JSP file	JSP files	• `contentType` attribute value of the `Page` directive (including charset)#2 • `pageEncoding` attribute of the `Page` directive or `Tag` directive#3 • `page-encoding` element of `web.xml`#2 • `encoding` attribute of the XML declaration#4

#1
Package is `javax.servlet`.

#2
Method set up in the JSP page.

#3
Method set up in the JSP page or in the standard format tag file.

#4

Method set up in the JSP document or XML tag file.

Table 2-22: Character encoding setting method defined in the Servlet specifications (Servlet 2.3/JSP 1.2)

Setting contents	Setting location	Setting method in the Servlet specifications
Character encoding of request	Servlet	`ServletRequest.setCharacterEncoding(java.lang.String` *env*`)`[#1]
	JSP files	None
Character encoding of response	Servlet	• `ServletResponse.setContentType(java.lang.String` *type*`)`[#1] • `ServletResponse.setLocale(java.util.Locale` *loc*`)`[#1]
	JSP files	`contentType` attribute value of the `Page` directive (including charset)
Character encoding of JSP file	JSP files	• `contentType` attribute value of the `Page` directive (including charset)[#2] • `pageEncoding` attribute of the `Page` directive[#2]

#1

Package is `javax.servlet`.

#2

Method set up in the JSP page or JSP document.

(2) Character encoding defined in the Servlet specifications

If the character encoding settings in the Servlet specifications and the default character encoding settings in Application Server do not exist, the character encoding defined in the Servlet specifications is applied.

When the character encoding is not specified, the following character encoding that are defined in the Servlet specifications will be applied:

- For a request

  ISO-8859-1 is applied. The character encoding is specified in the servlets and JSP files by using Servlet API.
- For a response

  The following table lists the character encoding defined in the Servlet specifications for each servlet version:

  Table 2-23: Character encoding defined in the Servlet specifications (Response)

Servlet version	Types	Applied character encoding
Servlet 2.3	Servlet	ISO-8859-1
	JSP page	
	JSP document	
Servlet 2.4 or later	Servlet	ISO-8859-1
	JSP page	
	JSP document	UTF-8

- In the case of JSP files

  The following table lists the character encoding defined in the Servlet specifications for each Servlet version:

  Table 2-24: Character encoding defined in the Servlet specifications (JSP file)

Servlet version	JSP version	Types	Applied character encoding
Servlet 2.3	JSP 1.2	JSP page	ISO-8859-1
		JSP document	

Servlet version	JSP version	Types	Applied character encoding
Servlet 2.4 or later	JSP 2.0, 2.1	JSP page	ISO-8859-1
		Standard format tag file	
		JSP document	UTF-8
		XML tag file	

2.6.6 Definition in the DD

Define the default character encoding for each Web application in `web.xml`.

The following table lists the definition of default character encoding in the DD:

Table 2-25: Definition of default character encoding in the DD

Specified tags	Setting contents
`<http-request>-<encoding>` tag	Specify the character encoding to be used for request body and query decoding.
`<http-response>-<encoding>` tag	Specify the character encoding to be used for the response body encoding.
`<jsp>-<page-encoding>` tag	Specify the character encoding for the JSP file.

Note:

If the character encoding settings in the Servlet specifications exist in the servlets or JSP files in the Web application, the settings in the Servlet specifications are enabled. For details on the priority level of the settings, see *2.6.1(3) Operations when character encoding is specified with multiple ranges*.

2.6.7 Execution environment settings

To specify settings for the default character encoding, you must set up the J2EE server and Web applications.

Reference the Web application settings only when you want to set up or change the properties of Web applications that do not contain `cosminexus.xml`.

(1) Setting up the J2EE server

Implement the J2EE server settings in the Easy Setup definition file. Specify the definition for the default character encoding within the `<configuration>` tag of the logical J2EE server (j2ee-server), in the Easy Setup definition file.

The following table lists the definitions for the default character encoding in the Easy Setup definition file:

Table 2-26: Definitions of default character encoding in the Easy Setup definition file

Parameter to be specified	Setting contents
`webserver.http.request.encoding`	Specify the character encoding to be used for request body and query decoding.
`webserver.http.response.encoding`	Specify the character encoding to be used for the response body encoding.
`webserver.jsp.pageEncoding`	Specify the encoding for the JSP file.
`webserver.static_content.encoding.extension`	Specify the static contents extension for applying the default response character encoding.

Note:

If the settings for each J2EE server are specified with the character encoding settings for each Web application, the settings for each Web application are enabled. For details on the priority level of the settings, see *2.6.1(3) Operations when character encoding is specified with multiple ranges*.

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

(2) Web application setup

Implement the settings for the Web applications in the execution environment using the server management commands and property files. Use the WAR property file for defining the default character encoding.

The tags specified in the WAR property file correspond to the DD. For details on the definition in the DD (`web.xml`), see *2.6.6 Definition in the DD*.

2.6.8 Precautions related to default character encoding

Note the followings regarding the application of default character encoding:

(1) Availability of default character encoding settings for response

The default character encoding settings for response are not enabled in the following cases:

- When the static contents extension is not the extension specified in the `webserver.static_content.encoding.extension` parameter[#1]
- When the default character encoding for response is not set up for the static contents of the error page defined in the Servlet specifications[#2]

#1 Parameter for specifying the static contents extension that applies the default character encoding.

#2 In the case of static contents output after one of the following conditions, the default character encoding settings for response are enabled:

- When the character encoding is not set for response and when the `java.io.PrintWriter` object is acquired by the `getWriter` method of the `javax.servlet.ServletResponse` interface in the servlet, JSP, or filter.
- When the request object is wrapped even though the `setAttribute` method is executed and the request object is wrapped with the request wrapper that does not invoke the `setAttribute` method.

Note that when the HTTP response compression filter is used, the default character encoding settings for response are not enabled.

(2) Character encoding applied to the getCharacterEncoding method

When the character encoding is not specified in the Servlet specifications, the default character encoding specified for each J2EE server or Web application is applied to the following Servlet API methods:

- `getCharacterEncoding` method of `javax.servlet.ServletRequest` (for request)
- `getCharacterEncoding` method of `javax.servlet.ServletResponse` (for response)

However, when the character encoding is changed using the `setCharacterEncoding` method, the same character encoding will be acquired.

For the response, when the `reset` method of `javax.servlet.ServletResponse` is used to initialize the response data, the character encoding set in Application Server can be acquired with the `getCharacterEncoding` method.

For details on the character encoding setting method defined in the Servlet specifications, see *2.6.5 Implementation of default character encoding (For Servlet specifications)*.

(3) Character encoding in the XML declaration

When the Web container automatically generates an XML declaration in the JSP document and XML tag file, the default character encoding applied to the response body encoding is output in the character encoding declaration in the XML declaration.

(4) Application of character encoding to JSP files

The character encoding settings for JSP files is applied during the compilation of the JSP file. Therefore, if the character encoding settings are added or changed when the JSP file is already compiled, the added or changed character encoding will not be applied to the JSP file. To apply the settings, re-compile the JSP file.

(5) Specifying the character encoding for Web applications that executed JSP pre-compilation with version 07-00

For the Web applications that executed JSP pre-compilation with version 07-00, when you set up the default character encoding for response, re-execute the JSP pre-compilation functionality.

The followings are the description for each of the units to set up the default character encoding:

- In the case of settings for each J2EE server
  Execute JSP pre-compilation for the JSP files of all the Web applications running on the J2EE server.
- In the case of settings for each Web application
  Execute JSP pre-compilation for the JSP files included in the Web applications.

For details on the default character encoding settings during the JSP pre-compilation, see *2.6.3 Application of character encoding during JSP pre-compilation*.

(6) Character encoding for default error pages

The character encoding for default error pages is specified in UTF-8, and therefore the default encoding is not applied.

2.7 Session management functionality

This section describes the session management functionality.

The session management is a functionality used for associating the requests and Web clients. This functionality enables you to perform operations in which the same information is inherited from the Web clients across multiple Web pages.

The following table describes the organization of this section.

Table 2-27: Organization of this section (Session management functionality)

Category	Title	Reference
Description	Objects managing the session information	*2.7.1*
	Session ID format	*2.7.2*
	Session management method	*2.7.3*
	Deleting invalid session IDs maintained by the Web client	*2.7.4*
	Setting the upper limit for the number of HttpSession objects	*2.7.5*
	Adding a server ID to the session ID and Cookie	*2.7.6*
Implementation	Definition in cosminexus.xml	*2.7.7*
Settings	Execution environment settings	*2.7.8*
Notes	Precautions related to session management	*2.7.9*

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

The session management functionality is also used in the following cases:

- Identification of clients when using the load balancing functionality by clustering
- Identification of clients who have already logged on to the security management functionality

The Servlet specifications clearly describe the method of using Cookies and the method of using URL rewrite as the session management methods. Sessions are managed using these methods even in the Web container of Application Server. However, in the Servlet specifications, there are some parts for which the management method is not specifically described. Therefore, this subsection describes how the session management methods that are not described in the Servlet specifications are managed on Application Server.

This subsection also describes the following three types of session management functionality that are unique to Application Server:

- Deleting invalid session IDs maintained by the Web client
- Setting the upper limit for HTTPSession objects
- Adding server ID to the session ID and Cookie

Tip

Deleting invalid session IDs maintained by the Web client, setting the upper limit for the HttpSession objects, and adding server ID to the session ID functionality are the prerequisites when using the session failover functionality of Application Server. For the session failover functionality, see *5. Inheriting Session Information Between J2EE Servers* in the *uCosminexus Application Server Expansion Guide*.

2.7.1 Objects managing the session information

This subsection describes the `HttpSession` objects used for managing the session information.

(1) How to manage the HttpSession object

The session information is information managed by the *HttpSession object* defined in the Servlet API.

The management of the session information starts at the following points:

- In a servlet, when the HttpSession object is referenced.
- In a JSP, when a reference to a page occurs (however, this is a default case).

After management of the session information starts, if request is sent from the same browser process to servlets in the same Web application, the HttpSession object of the managed content is passed to the servlet.

However, the HttpSession object instances actually passed to the servlet are different for each request. In other words, the HttpSession objects with the same contents but different instances might be passed in a series of requests belonging to the same session.

Therefore, note the following points in the operations for the HttpSession objects:

- When accessing an HttpSession object, you need to acquire the instances for each request.
- Do not cache the references to the HttpSession objects across multiple requests.
- Furthermore, locking the HttpSession objects with 'synchronized' keyword of java is also meaningless. Do not lock the `HttpSession` objects.

(2) Storage period of the HttpSession object

The HttpSession objects are stored only in single JavaVM. Therefore, if a failure occurs in a JavaVM process (J2EE server) running as a servlet engine, the session information stored in the HttpSession object is lost.

Also, the session information is lost when the J2EE server terminates regardless of normal or abnormal termination.

If you want to store the session information even after the J2EE server terminates, use the session failover functionality that is the Application Server functionality. For details on the session failover functionality, see *5. Inheriting Session Information Between J2EE Servers* in the *uCosminexus Application Server Expansion Guide*.

2.7.2 Session ID format

This subsection describes the *session ID* format used for identifying the session information. The HttpSession object is identified by the session ID.

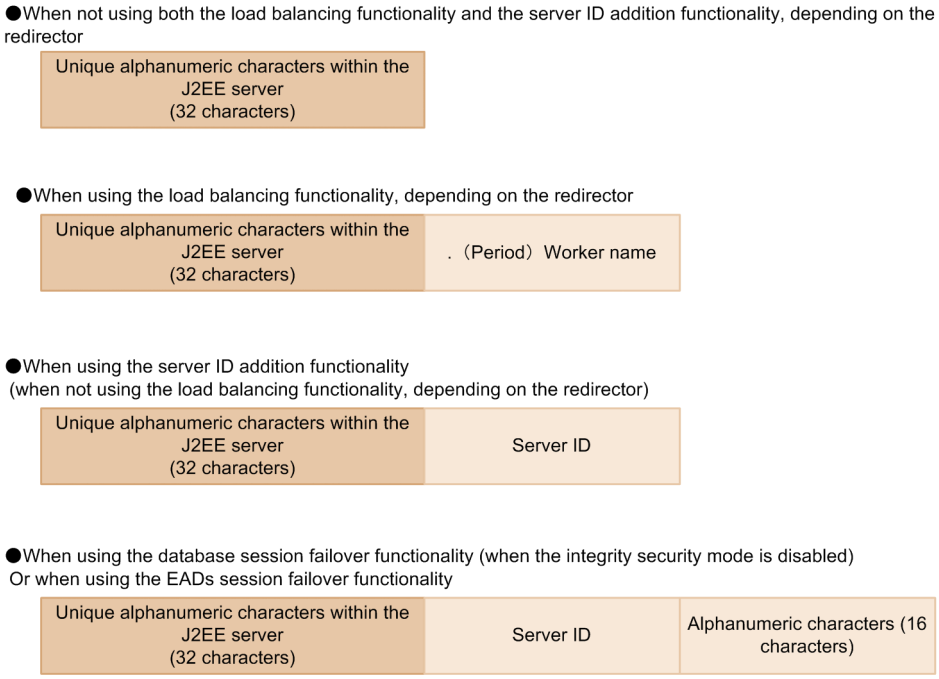
The session ID format depends on whether the following functionality is used:

- Redirector-based load balancing functionality
- Functionality to append a server ID to the session ID

The session ID is guaranteed to be unique in the J2EE server. However, if you want to set a unique value across multiple J2EE servers, add a worker name or server ID to the session ID and ensure that the worker name or server ID is unique across multiple J2EE servers.

The following figure shows the session ID formats for the used functionality:

Figure 2-11: Session ID format



Each case is described below.

- **When both the redirector-based load balancing functionality and the server ID addition functionality are not used**

  The session ID is 32 alphanumeric characters. These alphanumeric characters form a unique value on a J2EE server.

- **When the redirector-based load balancing functionality is used**

  The session ID consists of 32 alphanumeric characters, period (.), and worker name. You must specify a unique name for each server with the worker name.

  For details on the redirector-based load balancing functionality, see the description related to the use of the load balancer in *4.2 Distributing requests with the Web server (Redirector)*.

- **When the server ID addition functionality is used (When the redirector-based load balancing functionality is not used)**

  The session ID consists of 32 alphanumeric characters and the server ID. You must specify a unique value for each server with the server ID.

  For details on the server ID addition functionality, see *2.7.6 Adding a server ID to the session ID and Cookie*.

  Note that when you use the redirector-based load balancing functionality, the server ID is not added.

- **When the database session failover functionality is used (when completeness guarantee mode is disabled) or when the EADs session failover functionality is used**

  A session ID is formed of 32 alphanumeric characters, a server ID and 16 alphanumeric characters. The addition of a server ID to the session ID, by Server ID addition functionality is a prerequisite for using the database session failover functionality (when completeness guarantee mode is disabled) and EADs session failover functionality.

  Note that you must set a unique value for each server, in the server ID.

  For details on the Server ID addition functionality, see *2.7.6 Adding a server ID to the session ID and Cookie*.

  For details on the settings that are the prerequisites for the database session failover functionality and EADs session failover functionality, see *5.4.2 Prerequisite settings* in the *uCosminexus Application Server Expansion Guide*.

2.7.3 Session management method

This section gives the details on how to manage a session and the session ID management.

(1) How to manage a session

Specify details about how to manage a session of a Web container, in the tracking mode. There are two types of tracking mode; a mode using an HTTP Cookie and a mode using the URL rewrite. You can choose to use either one or both of the tracking modes.

- When using only an HTTP Cookie

  Note that only the session management by the HTTP Cookie is enabled. At this time, a character string with the URL rewrite does not include a URL path parameter that indicates the session ID.
- When using only the URL rewrite

  Note that only the session management by the URL rewrite is enabled. At this time, the response does not include the information of an HTTP Cookie that indicates the session ID.
- When using both the HTTP Cookie and the URL rewrite

  The session management by an HTTP Cookie and the session management by a URL re-write is enabled. The Web container determines the method used for managing the session, depending on the method from which the session ID is acquired. When the session ID is acquired from an HTTP Cookie, the Web container determines that the session is managed by the HTTP Cookie. When the session ID is obtained from a path parameter of a URL, the Web container determines that the session is managed by the URL rewrite. The Web container carries out this determination for each request.

(2) Session ID management when HTTP Cookie is used for session management

The session ID is managed as an HTTP Cookie. You can attach the HttpOnly property to the HTTP Cookie.

When a new HTTP session is created, an HTTP Cookie indicating the session ID is added to the HTTP response header. The name of the HTTP Cookie indicating the session ID is `JSESSIONID`. You can change the Cookie name if you are using Application Server 09-00 or later. Note that if the created HTTP session is disabled before commit, the HTTP Cookie is not added.

(3) Session ID management when URL rewrite is used for session management

The session ID is managed as a URL path parameter.

The name of the URL path parameter indicating the session ID is `jsessionid`. You can change the name if you are using Application Server of 09-00 or later. The session ID is added in the format `;jsessionid= session ID` at the end of the URL path, when the URL is rewritten by the Web container.

The URL path has a hierarchical structure and includes a value used for identifying resources. A query and fragment is not included in the URL path. Therefore, when these elements are included in the URL, the session ID is added immediately after the query or fragment. Also, if a path parameter other than the session ID is included in the URL, the path parameter indicating the session ID is added at the end of the path parameter included in the URL.

Tip

When you add the session ID to the URL path parameter, the URL string length increases.

The following table lists the increased string length:

Table 2-28: URL string length increased by URL rewrite

Functionality usage status	Increases URL string length (unit: string length)
When the redirector-based load balancing functionality or server ID addition functionality is not used	`44`#
When the redirector-based load balancing functionality is used	`44`# + `1` (*string-length-of-period*) + *string-length-of-worker-name*
When the server ID addition functionality is used (when the redirector-based load balancing functionality is not used)	`44`# + *string-length-of-server-ID*

Functionality usage status	Increases URL string length (unit: string length)
When the database session failover functionality (when completeness guarantee mode is disabled) or EADs session failover functionality is used	44# + characters count of server ID + 16 (alphanumeric character count)

#

This is the sum of the 12 characters (`;jsessionid=`) plus the 32 characters of the session ID.When the name of the path parameter of a URL is changed, the total of the following values serves as the size of the path parameter:

- Character count of path parameter
- Character count of the semicolon (;), and equal (=) signs
- Character string length of the session ID of an HTTP session (32 characters)

2.7.4 Deleting invalid session IDs maintained by the Web client

The invalid session IDs stored by a Web client will be deleted with Application Server. As a result, the sending of invalid session ID from Web clients is controlled.

If the HTTP session is disabled or if an HTTP session containing an invalid session ID is received, the HTTP Cookie used for deleting the HTTP Cookie information showing the invalid session ID with the Web container is added to the HTTP response header. As a result, the invalid session ID is deleted.

The HTTP Cookie used for deleting the HTTP Cookie information showing the invalid session ID indicates the HTTP Cookie that fulfills all the following conditions:

- HTTP Cookie specifying the session ID and name is `JSESSIONID` (you can change the Cookie name if you are using Servlet 3.0 or later)
- Value is `""` (null character string).
- A positive number forming the lapsed period is set for the validity period of the HTTP Cookie.

The HTTP Cookie used for deleting the HTTP Cookie information showing the invalid session ID is added to the HTTP response header in the following cases:

- HTTP session is disabled.
- A session ID that does not exist in the J2EE server is received.

The following are the description for each case:

Precautions for using the Web server integration functionality

If the response status code is 304 (Not Modified), the Set-Cookie header might be deleted in the Web server specifications. At this time, the HTTP Cookie used for deleting the HTTP Cookie information showing the invalid session ID is also not added, therefore, you cannot delete the invalid session IDs maintained by the Web client.

(1) When the HTTP session is disabled

When all the following conditions are fulfilled, the HTTP Cookie used for deleting the HTTP Cookie information showing the invalid session ID is added to the HTTP response header:

- The session ID is notified by using the HTTP Cookie.
- The HTTP session is disabled before the HTTP response is committed in the Web application[#1].
- The HTTP session does not exist when the HTTP response is committed[#2].

#1

The HTTP response header is sent to the Web client when the response is committed, so the HTTP Cookie cannot be added to a response after commit. Therefore, when the HTTP session is disabled after the HTTP response is committed in the Web application, the HTTP Cookie used for deleting the HTTP Cookie information is not added. However, when the next request is received, a non-existing session ID is received,

and therefore the subsection *2.7.4(2) When a session ID that does not exist in the J2EE server is received* will be applicable and the HTTP Cookie information will be deleted.

#2

If a new HTTP session is created when the HTTP response is committed, the HTTP Cookie information of the Web client is overwritten in the HTTP Cookie specifying the new session ID, and therefore the HTTP Cookie information is not required to be deleted.

Note that when one of the following conditions are fulfilled and even if the HTTP session is disabled, the HTTP Cookie used for deleting the HTTP Cookie information showing the invalid session ID is not added:

- Request is received in the simple Web server.
- The servlet engine mode is used.

These functions are provided to maintain compatibility with previous versions.

(2) When a session ID that does not exist in the J2EE server is received

When all the following conditions are fulfilled, it is determined that an invalid session ID is received and the HTTP Cookie used for deleting the HTTP Cookie information showing the invalid session ID is added to the HTTP response header:

- The session ID is notified by using the HTTP Cookie.
- The notified session ID does not exist in the J2EE server.
- The HTTP session does not exist when the HTTP response is committed.

(3) Notes on deleting invalid session IDs maintained by the Web client

In a configuration in which requests with the same path are handled in multiple J2EE servers, disable this functionality.

If you handle requests with the same path in which the Path of the Cookie has been rewritten using reverse proxy, the HTTP session might be deleted inappropriately.

2.7.5 Setting the upper limit for the number of HttpSession objects

With Application Server, you can set an upper limit for the number of valid `HttpSession` objects.

Set the upper limit for `HTTPSession` objects as the property of the Web applications contained in a J2EE application. For details on the settings for the J2EE applications, see *2.7.7 Definition in cosminexus.xml*.

! Important note

If the session failover functionality is applied to the Web applications, you must set the upper limit for the number of HTTP sessions when the following settings are made.

- Settings for canceling the start processing of the Web application (`webserver.dbsfo.negotiation.high_level` key of the Easy Setup definition file) when the negotiation processing during the startup of the application fails.

In this case, if the upper limit is not set for the number of HTTP sessions, an error occurs when you start the Web application that inherits the session information and you cannot start the Web application. If you have specified the settings for continuing the startup processing of the Web application when the negotiation processing during the startup of the application fails, the settings of the upper limit of the number of `HttpSession` objects is optional.

For the session failover functionality, see *5.2 Overview of the session failover functionality* in the *uCosminexus Application Server Expansion Guide*.

For the negotiation processing of the session failover functionality during the startup of a Web application, see *6.4.1 Processing at the time of starting an application* in the *uCosminexus Application Server Expansion Guide*.

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

Reference note

When the size estimation functionality of the global session information is enabled, the upper limit for the number of `HttpSession` objects that is the prerequisite for the session failover functionality is not required to be specified. For the size estimation functionality of the global session information, see *6.5 Estimating the size of the global session information* in the *uCosminexus Application Server Maintenance and Migration Guide*.

(1) HttpSession objects for which upper limit is to be set

The upper limit is set for enabled `HttpSession` objects.

Among the HttpSession objects acquired by the `getSession` method of the `javax.servlet.http.HttpServletRequest` interface, valid HttpSession objects implies the objects that satisfy the following two conditions:

- Objects in which the timeout does not occur
- Objects in which the invalidate method is not called

(2) Operations when the number of HttpSession objects exceeds the upper limit (Occurrence of exception)

If the upper limit for the number of `HttpSession` objects is specified and if an attempt is made to generate `HttpSession` objects exceeding the specified upper limit, an exception occurs in the `getSession` method of the `javax.servlet.http.HttpServletRequest` interface. Depending on the settings, one of the following exceptions will occur:

- `java.lang.IllegalStateException`
- `com.hitachi.software.web.session.HttpSessionLimitExceededException` that is an inherited class of `java.lang.IllegalStateException`

Specify which exception will be thrown in the J2EE application parameters. For details on the settings for the J2EE applications, see *2.7.8(1) Setting up the J2EE server*.

The `HttpSession` object is generated at the following timing in the Web Container:

- JSP execution time
  Indicates the cases in which JSP is executed when `true` is specified in the session attribute of the page directive or when the session attribute is omitted.
- When a URL requiring `FORM` authentication is accessed.

If the `HttpSession` object generation process exceeds the specified upper limit, perform the following operations:

- When the upper limit is exceeded during execution of JSP
  `java.lang.IllegalStateException` or `com.hitachi.software.web.session.HttpSessionLimitExceededException` is thrown before execution of user code of JSP.
- When the upper limit is exceeded during access to a URL that requires FORM authentication
  `java.lang.IllegalStateException` or `com.hitachi.software.web.session.HttpSessionLimitExceededException` is thrown before a request is transferred to the page for logging in.

  Tip

  If the `com.hitachi.software.web.session.HttpSessionLimitExceededException` class is used, add *Cosminexus-installation-directory*`/CC/lib/ejbserver.jar` in the class path when the J2EE applications are developed, and compile the Java program.

(3) Operations when the number of HttpSession objects exceeds the upper limit (message output)

If the upper limit for the number of `HttpSession` objects is specified and if an attempt is made to generate `HttpSession` objects exceeding the specified upper limit, the KDJE39225-E message is output to the log.

The message KDJE39225-E is output every time the request that uses HTTP session is executed, therefore, the log might be filled up with the same message. To restrain the repeated output of the same message, you can set the interval (output interval) for KDJE39225-E. Set the interval as required. This interval is applied to each Web application.

For the message output interval settings, see *2.7.8(1) Setting up the J2EE server*.

2.7.6 Adding a server ID to the session ID and Cookie

On Application Server, you can add a server ID to the session ID and Cookie of HttpSession. This is called *server ID addition functionality*. Specify a different server ID value for each Web container.

Customize the properties of the J2EE server to specify the settings for adding a server ID to the session ID and Cookie. For details on customizing the settings for the J2EE server operations, see *2.7.8 Execution environment settings*.

Note that when you use the session failover functionality, it is mandatory to add the server ID to the memory session ID. For details on the memory session failover functionality, see *6. Compatibility functionality of the extended functionality (session failover functionality)* in the *uCosminexus Application Server Maintenance and Migration Guide*.

Important note

The cookie name specified in this functionality must not duplicate a Cookie name specified in the Servlet or JSP or a Cookie name automatically specified by the Web container. The following name is automatically specified by the Web container:

`JSESSIONID`

With Application Server version 09-00 or later, you can change the name of the Cookie set by the Web container. For precautions to be taken when the server ID is attached to the Cookie and you want to change the name of the Cookie set by a Web container, see *2.7.9 Precautions related to the session management*.

(1) Adding server ID to the session ID of HttpSession

The session ID is normally unique in the same Web Container. However, in a system consisting of multiple Web containers by using a load balancer, the session ID might not be unique for the whole system. If you use the functionality for adding server ID to the session ID of HttpSession, a server ID that differs for each Web container is added to the session ID of HttpSession. Consequently, a unique session ID can be maintained in the system.

Reference note

When integrating with a Web server, if requests are distributed using round robin as per the settings of the redirector, the worker name is added to the session ID regardless of whether the settings for adding the session ID of HttpSession are specified or not. The server ID is not added.

(2) Adding a server ID to a Cookie

To transfer the requests of the same session to the same Web container, use the functionality for specifying the request transfer destination by the Cookie of a load balancer and the functionality for adding the server ID to the Cookie.

When you use the functionality for adding the server ID to the Cookie, a server ID that is different for each Web container is added to the Cookie. You can add the Cookie (in which the server ID was added) to an HTTP response, and hence, the requests of the same session can be transferred to the same Web container. Note that the Cookie with a server ID is added to the response for the request that generates the HttpSession.

Note that in the following cases, the `Secure` property is added to the Cookie generated by the server ID addition functionality:

- When an HTTP request is sent using the HTTPS protocol

- When the scheme is set with the gateway specification functionality so that the scheme is considered as HTTPS

2.7.7 Definition in cosminexus.xml

This subsection describes the definitions in `cosminexus.xml` required in the application development environment.

Setting the upper limit for the number of HttpSession objects

Specify the upper limit for the number of `HttpSession` objects in the `<http-session>-<http-session-max-number>` tag within the `<war>` tag of `cosminexus.xml`. In the Web applications that inherit the session information, you are not required to set a valid value of one or more as the upper limit for the number of HTTP sessions.

Customizing the session parameters

Specity a session parameter in the `war-session-config` tag of `cosminexus.xml`.

The following table lists the definitions of session parameters.

Table 2-29: Customizing the session parameters in cosminexus.xml

Tags to be specified	Setting contents
`cookie-config-name` tag	Specifies the the HTTP Cookie name and path parameter name of a URL.
`cookie-config-http-only` tag	Specifies whether `HTTP Cookie` has only the `HttpOnly` attribute.
`tracking-mode` tag	Specifies the HTTP session management method.

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*.

2.7.8 Execution environment settings

To use the session management functionality, you must set up a J2EE server.

Also, when you use J2EE applications that do not contain `cosminexus.xml`, you are not required to set up or change the execution environment properties.

(1) Setting up the J2EE server

Implement the J2EE server settings in the Easy Setup definition file. Define the session management functionality in the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file.

The following table describes the session management functionality defined in the Easy Setup definition file:

Table 2-30: Definition of the session management functionality in the Easy Setup definition file

Items	Parameter to be specified	Setting contents
Customizing the session parameters	`webserver.session.cookie_config.name`	Specifies the name of HTTP Cookie and the path parameter name of a URL.
	`webserver.session.cookie_config.http_only`	Specifies whether to add the `HttpOnly` attribute in HTTP Cookie.
	`webserver.session.tracking_mode`	Specifies the HTTP session management method.

Items	Parameter to be specified	Setting contents
Set maximum number of HttpSession objects	webserver.session.max.throwHttpSessionLimitExceedException	Specify whether to throw com.hitachi.software.web.session.HttpSessionLimitExceedException when the number of HttpSession objects exceeds the upper limit. To throw java.lang.IllegalStateException, specify false.
	webserver.session.max.log_interval	Specify the output interval for KDJE39225-E.
Adding server ID to the session ID and Cookie	webserver.session.server_id.enabled	Specify whether to add the server ID to the session ID.
	webserver.session.server_id.value	Specify the server ID to be added to the session ID.
	webserver.container.server_id.enabled	Specify whether to add the server ID to the Cookie.
	webserver.container.server_id.name	Specify the name of the Cookie to be added to the server ID.
	webserver.container.server_id.value	Specify the server ID to be added to the Cookie.

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

(2) Setting up the J2EE application

Set up the J2EE applications in the execution environment using the server management commands and property files. Use the WAR property file to define the session management functionality.

The tags specified in the WAR property file correspond to cosminexus.xml. The definition of the session parameter customization conforms to the standard specifications of Servlet 3.0. For details on the definitions in cosminexus.xml, see *2.7.7 Definition in cosminexus.xml*.

2.7.9 Precautions related to session management

This subsection describes the precautions related to session management.

(1) Customizing session parameters and the precautions related to the session parameters

You can change the name of HTTP Cookie and the path parameter name of a URL by performing the following settings:

- Easy Setup definition file (webserver.session.cookie_config.name parameter)
- cosminexus.xml (war-session-config-cookie-config-name tag)
- web.xml (/web-app/session-config/cookie-config/name element) (in the case of Servlet 3.0 and later)
- Servlet API (setName() method of the javax.servlet.SessionCookieConfig interface) (in the case of Servlet 3.0 and later)

Important note

If settings made with different methods exist, the settings are applied in the order of the settings made using the Servlet APIs, coding in the `cosminexus.xml`, coding in `web.xml`, and coding in the Easy Setup definition file.

When the `webserver.session.cookie_config.name` parameter of the Easy Setup definition file is `true`, and the specified HTTP Cookie name is duplicated with the path parameter name of the URL as well as the name of the Cookie specified in the Cookie addition function, the system performs the following operations.

When duplicated with the name specified in the `webserver.session.cookie_config.name` parameter of the Easy Setup definition file

A default session ID is set. In the case of an HTTP Cookie, the session ID is '`JSESSIONID`' and in the case of a path parameter of a URL, the session ID is '`jsessionid`'.

When duplicated with the name specified in cosminexus.xml, web.xml, or Servlet API

KDJE39338-E is output at the time of starting the Web application, and the Web application fails to start. KDJE39338-E is output at the time of executing the Web application reload process and the reload process is continued. When KDJE39338-E is output during the reload process, first modify the file in which the name of the Cookie was changed by using Servlet API, and then re-execute the reload process.

(2) API used for URL rewrite and related precautions

URL rewrite is executed when the Servlet API that executes URL rewrite in the J2EE applications is invoked.

- The Servlet API that executes URL rewrite is a method of the following `javax.servlet.http.HttpServletResponse` interface:
- `encodeURL(java.lang.String` *url*`)` method
- `encodeRedirectURL(java.lang.String` *url*`)` method
- `encodeUrl(java.lang.String` *url*`)` method
- `encodeRedirectUrl(java.lang.String` *url*`)` method

For details on these methods, see the Servlet specifications. Note that the `encodeUrl(java.lang.String url)` method and the `encodeRedirectUrl(java.lang.String url)` method are the deprecated APIs in Servlet 2.1 and later versions. Therefore, Hitachi recommends that you use methods other than these two methods.

As the API operations not specified in the Servlet specifications, this subsection describes the operations on Application Server for the return value of the Servlet API for executing URL rewrite.

An HTTP session is enabled only in Web applications that are processing the requests. Therefore, URL rewrite will only be performed if the URL specified in the Servlet API argument is the URL indicated in the Web applications that are processing the requests. The following table lists the conditions for determining whether the URL is in the Web applications that are processing the requests for each URL argument:

Table 2-31: Conditions for determining whether the URL is in the Web applications that are processing the requests for each URL argument

Types of argument URL	Conditions
Relative URL (Example: `/ex/a.html`)	The URL is determined as being in the Web application only if the following condition is fulfilled: • The normalized path of the argument URL contains the context root name of the Web application that is processing the requests.[#1]
Absolute URL (Example: `http://host1/ex/`)	The URL is determined as being in the Web application only if the following conditions are fulfilled: • The schema of the argument URL is `http` or `https`.[#2] • If the argument URL and request URL have the same schema, the port number is matching. • The host name of the argument URL matches with the request host name. [#1#3] • The normalized path of the argument URL contains the context root name of the Web application that is processing the requests.[#1]

#1

Case-sensitive when names are compared.

#2

Case-insensitive when names are compared.

#3

The request host name is the host name part of the request Host header and compare the strings without performing name resolution of the host name. Use the return value of the `javax.servlet.ServletRequest.getServerName` method for the request host name. Note that in the following cases even if the host is the same, the host will be determined as a different host:

- When the argument URL is the host name specification, the request URL is the IP address
- When the argument URL is the IP address specification, the request URL is the host name

This subsection also describes the return values when a string other than URL is specified in the Servlet API argument for executing URL rewrite. Furthermore, the specification of a query or fragment at the beginning of the URL is also described here.

The following table lists the return values for each Servlet API argument for executing URL rewrite:

Table 2-32: Return values for each Servlet API argument for executing URL rewrite

Item No.	Conditions		Return value or exceptions
	HTTP session	Argument	
1	N	null	null is returned.
2	N	Invalid format for URL	The java.lang.IllegalArgumentException exception has occurred.
3	• A new HTTP session is present during the request processing. • Session ID is notified during URL rewrite.	Null character string	The value with the session ID added is returned for the URL path of the HTTP request and the query.[#1]
4		URL beginning with query (when the first character is a question mark (?))	The value with the value specified in the session ID and argument added is returned for the URL path of the HTTP request.[#1]
5		URL beginning with fragment (when the first character is a hash mark (#))	The value specified in the argument is returned.[#2]
6		URL containing the path parameter that indicates the session ID of the current HTTP session	The value specified in the argument is returned.
7		URL determined to be in the Web application that is processing the requests.	The value with the session ID added in the argument is returned.
8	Other conditions		The value specified in the argument is returned.

Legend:

--: Not applicable

Note:

The item numbers in this table indicate the priority of the conditions, the smaller the item number, the higher the priority of the condition.

#1

The path is not included in the argument URL, so you cannot directly add a path parameter to the argument URL. A URL in which the argument begins with a null character string or query indicates request URL resources, therefore, use the value with the path parameter added to the request URL to perform URL rewrite.

#2

A URL that is only a fragment indicates a specific location in the current resources. In a Web browser, normally this URL is treated as one that indicates shift in the displayed contents. At this time, a request is not sent to the server. This operation is in accordance with `RFC3986`.

The following is an example of URL where URL rewrite is used to add the session ID. Note that this example is based on the following prerequisites:

Preconditions

- Servlet API is executed after the HTTP session is generated.
- HTTP request URL is `http: //host1/gyoumu1/app1/index.jsp?type=1`.
- The context root name is `/gyoumu1`.

The following is an example table describing the compliance between the specified value of the servlet API arguments used in URL rewrite and the return value after rewrite (URL):

Table 2-33: Specified value of the servlet API arguments used in URL rewrite and the corresponding return value after rewrite (URL)

Servlet API arguments	Return values
b.html	b.html;jsessionid=AAAAA1111122222333334444445555566svr0
../b.html	../b.html;jsessionid=AAAAA1111122222333334444445555566svr0
../../b.html	../../b.html
http: //host2/	http: //host2
https: //host1/gyoumu1/	https: //host1/gyoumu1/;jsessionid=AAAAA1111122222333334444445555566svr0
"" (null character string)	"/gyoumu1/app1/index.jsp;jsessionid=AAAAA1111122222333334444445555566svr0?type=1"
"?mode=2"	"/gyoumu1/app1/index.jsp;jsessionid=AAAAA1111122222333334444445555566svr0?mode=2"
"#aaa"	"#aaa"

(3) Precautions for using URL rewrite

This point describes the precautions for using URL rewrite.

■ Screen transition from static contents

In the case of screen transition from the static contents (such as HTML file), the session managed with URL rewrite is not maintained.

When you use URL rewrite to manage a session, implement settings so that the screen will always transit by using servlets or JSPs. Also, implement the process for adding the session ID by rewriting URL with Servlet API in the servlets or JSPs.

■ Request URL acquired in the Web applications

Even if the HTTP request URL contains the path parameter indicating the session ID managed by URL rewrite, the path parameter indicating the session ID is not included in the URL acquired with the following methods:

Interface

`javax.servlet.http.HttpServletRequest` interface

Method

- `getRequestURI()` method

- `getRequestURL()` method

(4) Secure property of the HTTP Cookie used in session management

When an HTTP request is send using the HTTPS protocol, the session ID generated by the Web container is returned to the client by the HTTP Cookie. At that time, the `Secure` property is allocated to the HTTP Cookie.

Also, when the scheme is set up using the gateway specification functionality so that the scheme is considered as HTTPS, and if the session ID generated by the Web container is returned to the client by the HTTP Cookie, the `Secure` attribute is allocated to the respective HTTP Cookie.

2.8 Event listener of an application

This section explains the event listener functionality of an application.

Event listener functionality is present in each Web application. The application event listener is instantiated when a Web application is deployed. The instantiated application event listener receives the state change event of either one or both of the servlet context object and the session object. The events received by the listener object are as follows:

- Generating a new session object
- Before serializing a session object[#1]
- After de-serializing a session object[#1]
- Deleting a session object
- Adding, deleting, and changing attributes of a session object
- Generating a servlet context object
- Deleting a servlet context object
- Adding, deleting, and changing attributes of a servlet context object
- Arrival of requests in a Web application[#2]
- Completion of request processing in a Web application[#2]
- Adding, deleting, and changing attributes of a request object[#2]

#1

Servlet API assumes that due to the vendor-specific processing of a session object, a session object is saved and communicated after serializing, and processing is restarted after de-serializing a session object at another point of time.

The intention of providing such event notifications in session object is to add not just data, but also resources, such as database connections and object references in a session object. In an application designed to add resources to a session object, the resources must be released once before serializing, and then acquired again after de-serializing.

A configuration in which the resources are added to the session object, if not used with utmost care, the amount of resources required in the entire server will increase immensely, and there might be a shortage of resources. Therefore, use the event listener functionality if you can secure the resources.

#2

Can only be used in Web applications compliant with Servlet 2.4 or later specifications.

2.9 Functionality of filtering requests and responses

This subsection describes the functionality for filtering the requests and responses.

The following table describes the organization of this section.

Table 2-34: Organization of this section (Functionality for filtering requests and responses)

Category	Title	Reference
Description	Servlet filter provided by Application Server (built-in filter)	*2.9.1*
	Examples of recommended filter chain	*2.9.2*
Implementation	Definition in the DD	*2.9.3*
Settings	Execution environment settings (Web application settings)	*2.9.4*

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

The filtering functionality available on Application Server is the functionality defined in the Servlet specifications and the functionality provided with Application Server. Both of the above functions filter requests and responses of servlets or JSPs.

The filtering functionality defined in the Servlet specifications wraps the requests before executing the servlets and JSPs or the responses after executing the servlets and JSPs. As a result, you can perform operations such as changing the data and acquiring the trace for the resources.

With the filtering functionality provided with Application Server, you can inherit the session information and compress the HTTP responses. In Application Server, a servlet filter is provided for using this filtering functionality. The servlet filter provided in Application Server is called a *built-in filter*. The following subsection explains the built-in filter provided in Application Server.

2.9.1 Servlet filter provided by Application Server (built-in filter)

In Application Server, a servlet filter (built-in filter) is provided to use the following functionality:

- Inheriting the session information between the J2EE servers (Memory session failover functionality)
  To inherit the session information between the J2EE servers, Application Server provides a filter for session failover as the built-in filter.
- Compressing the HTTP responses
  To compress the HTTP responses for the HTTP requests, Application Server provides an HTTP response compression filter as the built-in filter.

The following table describes the types of built-in filters. Further, references of the functionality that you can use by embedding the built-in filter in a Web application are also described.

Table 2-35: Types of built-in filters

Type of built-in filter	Description of functionality	Reference manual	Reference
Filter for session failover	This functionality manages the session information executed in a J2EE application. In the case of failure of the J2EE server, the managed session information is inherited to another J2EE server.	*uCosminexus Application Server Maintenance and Migration Guide*	*6.2*, *6.4*
HTTP response compression filter	This functionality compresses the HTTP responses to the HTTP requests for servlets, JSPs, and static contents, in the gzip format.	This manual	*2.10*

Tip

When you build in each filter in the Web application, specify in the order of 'filter for session failover', 'HTTP response compression filter' in the filter mapping definition. The filter for session failover must be deployed before all the built-in filters and user filters.

Note that with Servlet 3.0 or later, you can define a filter by using an API and not the `web.xml` file. However you cannot use an API to define a built-in filter.

The action of the built-in filter on HTTP requests and HTTP responses, the restrictions on the operation conditions of the built-in filter are explained below:

(1) Action on the HTTP requests and HTTP responses

The built-in filter acts on the request header and the request body of the HTTP requests sent from the client, and may delete, add, and change the information. In the same way, the built-in filter may also act on the response header and the response body of the HTTP responses sent from the server. The following table describes the action of the built-in filter on HTTP requests and HTTP responses:

Table 2-36: Action of the built-in filter on HTTP requests and HTTP responses

Type of built-in filter	Action on the HTTP request	Action on the HTTP response
Filter for session failover	--	--
HTTP response compression filter	--	When the response body is compressed, `gzip` is specified in the Content-Encoding header. The response body is compressed in gzip format.

Legend:
--: Not applicable

(2) Restriction on operation conditions

When you use the user filter and the built-in filter simultaneously, there are restrictions on the order in which the built-in filter is invoked in the filter chain.

The following restrictions are explained for each built-in filter:

- Restriction on the location
  This is the restriction on the location (invocation order) of the built-in filter in the filter chain.
- Restriction on operation conditions
  This is a restriction on the operation conditions, such as the preconditions for operation of the built-in filter.
- Restriction on the other servlet filters deployed before and after the built-in filter
  This is a restriction with respect to servlet filters that are deployed before and after the built-in filter.

(a) Restrictions on the filter for session failover

The following table describes the restrictions on the filter used for session failover:

Table 2-37: Restrictions on the filter used for session failover

Type of restriction	Description
Restriction on the location	The filter for session failover must be invoked first in the filter chain. You need to deploy the filter for session failover before all the user filters and the built-in filter.
Restriction on operation conditions	--
Restriction on the other servlet filters deployed before and after the built-in filter	--

Legend:
--: Not applicable

(b) Restrictions on an HTTP response compression filter

The following table describes the restrictions on the HTTP response compression filter:

Table 2-38: Restrictions on the HTTP response compression filter

Type of restriction	Description
Restriction on the location	--
Restriction on operation conditions	--
Restriction on the other servlet filters deployed before and after the built-in filter	In the case of concurrent use of the HTTP response compression filter and servlet filter that changes the settings of the Content-Length header or the Content-Encoding header by using the `setHeader` method, `addHeader` method, `setIntHeader` method, or `addIntHeader` method of the `javax.servlet.http.HttpServletResponse` interface, you need to deploy the servlet filter after the HTTP response compression filter.

Legend:
--: Not applicable

2.9.2 Examples of recommended filter chain

The examples of recommended filter chain are explained below. Deploy the filters to form a chain in the following order:

Note that the general filters that act on the request body and the response body are assumed as user filters explained in the examples.

- **When using filters for session failover, and an HTTP response compression filter**
  1. Filter for session failover
  2. HTTP response compression filter
- **When using filters for session failover, an HTTP response compression filter, and a user filter (Filter A)**
  1. Filter for session failover
  2. Session failover filter
  3. HTTP response compression filter
  4. User filter (Filter A)
- **When using a filter for session failover and a user filter (Filter C)**
  1. Filter for session failover
  2. User filter (Filter C)
- **When using an HTTP response compression filter and a user filter (Filter D)**
  1. HTTP response compression filter
  2. User filter (Filter D)

2.9.3 Definition in the DD

Define the functionality for filtering requests and responses in `web.xml`.

For details on the definition of functionality for filtering requests and responses in the DD, see the following locations respectively:

- For the filter for session failover, see *5. Inheritance of Session Information Between J2EE Servers* in the *uCosminexus Application Server Expansion Guide*.
- For details on the HTTP response compression filter, see *2.10 HTTP response compression functionality*.

2.9.4 Execution environment settings (Web application settings)

To define the functionality for filtering requests and responses you must set up the Web applications. Reference this subsection only if you want to set or change the properties of Web applications that do not contain `cosminexus.xml`.

Implement the Web application settings in the execution environment by using the server management commands and property files. To define the filtering for requests and responses, use the filter property file and WAR property file.

The tags specified in the filter property file and WAR property file correspond to the DD. For the definitions in the DD (`web.xml`), see *2.9.3 Definition in the DD*.

2.10 HTTP response compression functionality

This subsection describes the HTTP response compression functionality.

The HTTP response compression functionality compresses the HTTP responses for the HTTP requests to the servlets, JSPs, and static contents in the `gzip` format. By using this functionality to compress the HTTP responses, you can reduce the time required for communicating the HTTP responses between the Web container and Web client (browser).

This functionality is provided as a servlet filter that is embedded and runs in the Web application. This is called the *HTTP response compression filter*.

The following table describes the organization of this section.

Table 2-39: Organization of this section (HTTP response compression functionality)

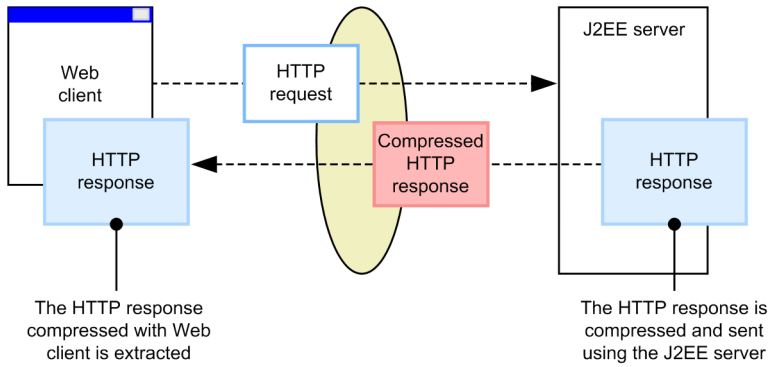
Category	Title	Reference
Description	Overview of HTTP response compression filter	*2.10.1*
	Conditions for using the HTTP response compression filter	*2.10.2*
Implementation	Executing the applications that use the HTTP response compression filter	*2.10.3*
	Definition in the DD	*2.10.4*
	Examples of the DD definitions	*2.10.5*
Settings	Execution environment settings (Web application settings)	*2.10.6*

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

2.10.1 Overview of HTTP response compression filter

If you enable the HTTP response compression functionality, the response body of the HTTP response is compressed in a gzip format. The following figure shows an overview of HTTP response compression functionality.

Figure 2-12: Overview of HTTP response compression functionality



To enable the HTTP response compression functionality, you need to embed the HTTP response compression filter provided by Application Server, in the Web application. In the case of applying the HTTP response compression functionality, add the filter definition of the HTTP response compression filter and the definition of the filter mapping to the DD (`web.xml`) of Web application. In the case of applying the HTTP response compression functionality to a Web application that is already deployed on a J2EE server, use the server management commands to add the filter definition of the HTTP response compression filter and the definition of filter mapping.

2.10.2 Conditions for using the HTTP response compression filter

This subsection describes the conditions and precautions for using the HTTP response compression filter.

(1) Preconditions

To use the HTTP response compression functionality, the following preconditions need to be satisfied:

- **gzip format compliant Web client**
  When the HTTP response compression functionality is enabled, you need to decompress HTTP responses that are compressed in gzip format with the Web client. The Web client, therefore must support the gzip format. If the Web client does not support the gzip format, HTTP responses are not compressed, even if the HTTP response compression functionality is enabled.
- **HTTP/1.1 compliant Web client**
  In HTTP response compression functionality, the value specified in the Accept-Encoding header of the HTTP request determines whether the Web client supports the gzip format. The Web client is therefore required to support HTTP/1.1 as defined in the specifications of the Accept-Encoding header.

(2) Required memory size

The memory required for the HTTP response compression functionality is obtained with the following formula:

Memory-required-for-the-HTTP-response-compression-functionality `(bytes)` = *Number-of-concurrent-executions-of-the-HTTP-requests-that-enable-the-HTTP-response-compression-functionality* × *Response-compression-threshold* `(bytes)`

The compression threshold is used for determining whether to compress the HTTP response depending on the size of the HTTP response body. Only when the size of the HTTP response body exceeds the size defined in the compression threshold, the HTTP response will be compressed. Note that the compression threshold is specified for HTTP requests.

Define the compression threshold in the DD (`web.xml`). When the size of the HTTP response is small, by defining the compression threshold you ensure that the time required for HTTP response compression is not longer than the time required for communication.

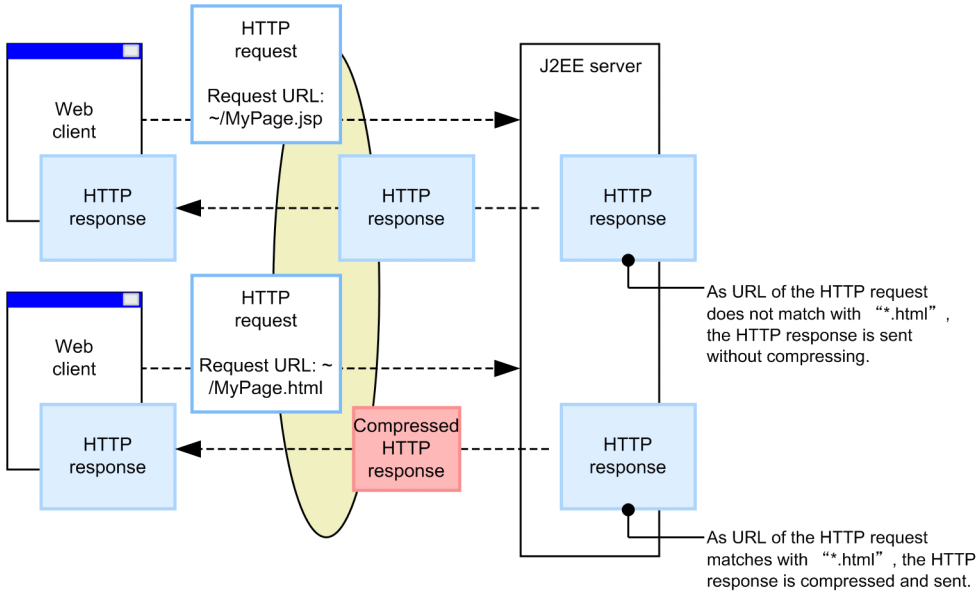
Decide an appropriate size for the compression threshold depending on the type of resources you want to compress and the speed of communications line. Hitachi recommends that you acquire the size defined in the compression threshold using the actual measurements and define the appropriate size.

(3) Conditions for enabling the HTTP response compression functionality

You can specify conditions for enabling the HTTP response compression functionality. The conditions that can be specified are explained below.

- **URL pattern of HTTP request**
  If the URL of the request to a Web application (in which the HTTP response compression filter is installed) matches with the specified URL pattern, compress the response to that request.
  The following figure shows an example with '*.html' specified as the URL pattern of the HTTP request that executes the compression of HTTP response:

Figure 2-13: Example with '*.html' specified as the UTL pattern of the HTTP request that executes HTTP response compression



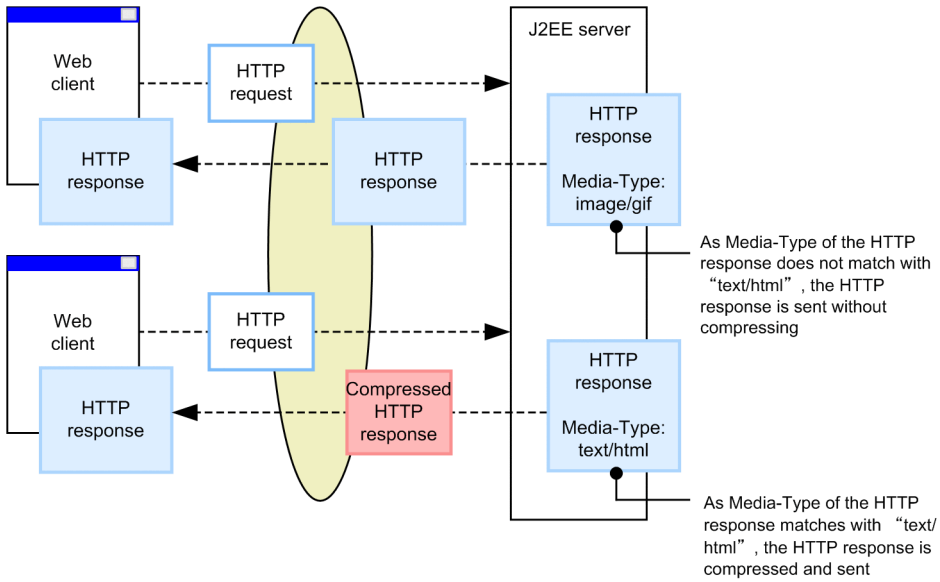
- **Media-Type of HTTP response**

  If the value of the Media-Type included in the Content-Type header of the HTTP response matches with the specified value, compress the HTTP response.

  In the case of servlets or JSPs, the value of the Media-Type of HTTP response is set by the `setContentType` method of a J2EE application. In the case of static contents, the Media-Type is the MIME type associated with the extension.

  The following figure shows an example with 'text/html' specified as the Media-Type of HTTP response that executes the compression of HTTP response:

  Figure 2-14: Example with 'text/html' specified as the Media-Type of HTTP response that executes HTTP response compression

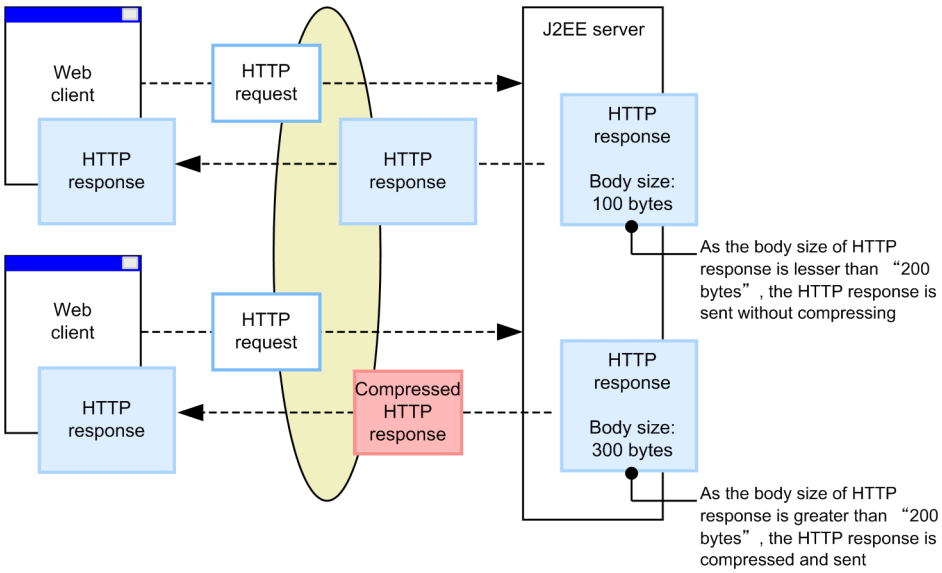


- **Body size of HTTP response**

  Set a threshold value to execute the compression of an HTTP response. If the body size exceeds this threshold value, compress the HTTP response.

  The following figure shows an example in which the HTTP response compression functionality is enabled by specifying '200 bytes' as the body size of the HTTP response that executes the compression of HTTP response:

Figure 2-15: Example with '200 bytes' specified as the body size of the HTTP response that executes HTTP response compression



(4) Notes

Precautions related to the definition of the HTTP response compression filter

When using the HTTP response compression filter, after considering the action of the built-in filter on the HTTP requests and HTTP responses and the restrictions on the order of the filter chain, you need to embed HTTP response compression filter in a Web application. For details on the built-in filter, see *2.9.1 Servlet filter provided by Application Server (built-in filter)*.

Note that with Servlet 3.0 or later, you can define a filter by using an API and not the `web.xml` file. However, you cannot use an API to define a built-in filter.

Precautions related to error pages

In the Web applications that use the HTTP response compression functionality, you can customize the error pages by using the following functionality:

- Error page customization with the Web server functionality
- Error page customization based on in-process HTTP server
- Error page customization with the `<error-page>` tag of `web.xml`

When using the error page with the `<error-page>` tag of `web.xml`, specify the servlet that acquires and uses `javax.servlet.ServletOutputStream` from the static contents or response in the error page.

In the HTTP response compression functionality, you use `javax.servlet.ServletOutputStream` acquired from the response object to output the compressed data. Therefore, `java.io.PrintWriter` cannot be acquired from the response object in the servlet or JSP that generates an error page.

2.10.3 Executing the applications that use the HTTP response compression filter

This subsection describes the precautions to be taken when developing applications that use the HTTP response compression filter.

(1) Order of invocation when the HTTP response compression filter is combined with other filters

The HTTP response compression filter must be invoked before the other filters specified in the HTTP response header. When you use the `setHeader` method, `addHeader` method, `setIntHeader` method, and

addIntHeader method of javax.servlet.http.HttpServletResponse to use other filters that set Content-Length header and Content-Encoding header, deploy the other filters after the HTTP response compression filter.

(2) Precautions related to HTTP response buffer

When the HTTP response compression functionality is enabled, the buffer with the size specified in the compression threshold is installed before the HTTP response buffer. Data is written in the HTTP response buffer when the output data exceeds the compression threshold.

Unless the compressed data size exceeds the HTTP response buffer size, the HTTP response is not written in the Web client. If you are required to write the HTTP response in the Web client before the output data size exceeds the compression threshold, you must explicitly invoke the flush method of the stream for response output#. However, if the flush method or flushBuffer method of the javax.servlet.ServletResponse interface is invoked before the output data size exceeds the compression threshold, the output data is written in the Web client without being compressed.

#

The stream for response output indicates the following objects:

- javax.servlet.ServletOutputStream acquired by the getOutputStream method of the javax.servlet.ServletResponse interface
- java.io.PrintWriter acquired by the getWriter method of the javax.servlet.ServletResponse interface
- javax.servlet.jsp.JspWriter implicitly available in JSP

(3) Precautions related to the response header of the HTTP response

When the response body of the HTTP response is compressed with the HTTP response compression functionality, gzip is specified in the Content-Encoding header and Accept-Encoding is specified in the Vary header of this HTTP response. Nothing is specified in the Content-Length header.

Therefore, note the following points when you use the setContentLength method of the javax.servlet.ServletResponse interface and when you use API# for adding and changing the response header of the javax.servlet.http.HttpServletResponse interface:

- When you use one of the following APIs to add the filter for setting the Content-Length header and Content-Encoding header, define in the DD (web.xml) that the API be executed after the filter for response compression:
  - setContentLength method of the ServletResponse class
  - API# for adding and changing the response header of the HttpServletResponse class
- When the response body of the HTTP response is compressed, the Content-Length header of the HTTP response is not added even though the setContentLength method of the ServletResponse class and the API# for adding and changing the response header of the HttpServletResponse class are used. The HTTP response for which the Content-Length header is not added is sent in the chunk format to the client by the Web container.
- When the response body of the HTTP response is compressed, a value is not set in the Content-Encoding header even though the API# for adding and changing the response header of the HttpServletResponse class is used. When the response body is compressed, gzip is specified in the Content-Encoding header by the Web Container.

#

The API for adding and changing the response header indicates the following methods of the javax.servlet.http.HttpServletResponse interface:

- setHeader method
- addHeader method
- setIntHeader method
- addIntHeader method

(4) Precautions related to data output for HTTP response

Note the following points when `ServletOutputStream` or `PrintWriter` is acquired with the `getOutputStream` method or the `getWriter` method of the `javax.servlet.ServletResponse` interface and the HTTP response is output:

- When you invoke the `setContentType` method of `ServletResponse` while the data is being written in the HTTP response buffer by using `ServletOutputStream` or `PrintWriter`, even if HTTP response with Media-Type specified for compression exists, the HTTP response is not compressed.

  However, if an asterisk (*) is specified in the Media-Type to be compressed, the HTTP response is compressed.
- To compress the JSP output by specifying Media-Type, either specify the `contentType` attribute of the `Page` directive or invoke the `setContentType` method of the `ServletResponse` class before the JSP buffer is exceeded.

(5) Precautions for compressing HTTP responses in applications

For HTTP responses compressed in applications, specify settings so that the HTTP response compression functionality is not enabled. If the HTTP response compression functionality is enabled for the HTTP responses compressed in applications, the operations might not function properly.

2.10.4 Definition in the DD

This subsection describes the DD definitions required for using the HTTP response compression functionality.

To enable the HTTP response compression functionality, you must add the filter definition and filter mapping definition in the DD of the Web applications. The HTTP response compression functionality is enabled only when requests exist for the resources with the URL pattern mapped by the filter mapping definition.

Define the HTTP response compression functionality in `web.xml`.

The following table lists the definition of HTTP response compression functionality in the DD:

Table 2-40: Definition of HTTP response compression functionality in the DD

Settings	Specified tags	Setting contents
Filter definition	`<filter-name>` tag in the `<filter>` tag	Specify the name of the filter you want to add. The set value is a fixed value. **Set value (fixed value)** `com.hitachi.software.was.web.ResponseCompressionFilter`
	`<filter-class>` tag in the `<filter>` tag	Specify the class name of the filter you want to add. The set value is a fixed value. **Set value (fixed value)** `com.hitachi.software.was.web.ResponseCompressionFilter`
Mapping of the URL pattern and HTTP response compression rules	`<param-name>` tag and `<param-value>` tag in the `<filter-class><init-param>` tag	Specify the mapping of the URL pattern and the HTTP response compression functionality. For details, see *2.10.4(1) Mapping of the URL pattern and HTTP response compression rules (url-mapping)*.
HTTP response compression rules		Specify the compression rule name, Media-Type, and compression threshold as the compression rules for HTTP response. For details, see *2.10.4(2) HTTP response compression rules*.
Filter mapping definition	`<filter-name>` tag in the `<filter-mapping>` tag	Specify the filter name. The set value is a fixed value. **Set value (fixed value)** `com.hitachi.software.was.web.ResponseCompressionFilter`

Settings	Specified tags	Setting contents
Filter mapping definition	<url-pattern> tag in the <filter-mapping> tag	Specify the mapping of the URL pattern or servlet class and the HTTP response compression filter. The HTTP response compression functionality is enabled only when requests exist for the resources with the URL pattern mapped by the filter mapping definition. The set value is optional.

The definition contents of the DD are described in the following examples of the DD definitions:

```
 ...
<filter>
 <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
 <filter-class>com.hitachi.software.was.web.ResponseCompressionFilter</filter-class>
 <init-param>
 <param-name>url-mapping</param-name>
 <param-value>
 /*=rule1;
 </param-value>
 </init-param>
 <init-param>
 <param-name>rule1</param-name>
 <param-value>
 *: 1000;
 </param-value>
 </init-param>
</filter>
 ...
<!-- The filter mappings for Response Compression Filter -->
<filter-mapping>
 <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
 <url-pattern>/*</url-pattern>
</filter-mapping>
```

The part enclosed in the <filter> tag is the filter definition and the part enclosed in the <filter-mapping> tag is the filter mapping definition. The value specified in the <filter-name> tag and <filter-class> tag of the filter definition is fixed in the following package name:

com.hitachi.software.was.web.ResponseCompressionFilter

The contents defined in the <init-param> tag of the DD are as follows:

(1) Mapping of the URL pattern and HTTP response compression rules (url-mapping)

The <init-param> tag of the DD specifies the mapping of the URL pattern that enables the HTTP response compression functionality and the name of the HTTP response compression rule applied to the specified URL pattern.

The rules for specifying parameters and the mapping rules for URL pattern are as follows:

(a) Rules for specifying parameters

- The URL pattern is case-sensitive.
- The name of the HTTP response compression rule is case-sensitive.
- The URL pattern and HTTP response compression rule name are delimited by one-byte equal sign (=).
- Delimit multiple specifications with one-byte semicolon (;).
- If multiple URL patterns are applicable, the URL pattern specified earlier is used.
- The linefeed, tabs, and spaces in the parameter value are ignored.
- The one-byte semicolon (;) at the end of the parameter value is ignored.

(b) Mapping rules for URL pattern

The path match, extension match, and exact match mapping rules are applicable to the URL pattern defined in the url-mapping parameter. The following shows the mapping rules:

- **Path match**

If a string beginning with / and ending with /* is specified as the URL pattern and if the relative path from the context root of the request URL begins with the string excluding * from the URL pattern, the string is considered as matching. Also, if /* is specified as the URL pattern, all the request URLs are considered to be matching.

Examples:

In the URL pattern /jsp/*, if the relative path from the context root of the request URL is /jsp/index.jsp, the string is considered as matching.

- **Extension match**

  If a string beginning with *. is specified as the URL pattern and if the string is the same as the string continuing after *. of the request URL extension and URL pattern, the string is considered as matching.

  Example:

  In the URL pattern *.jsp, if the relative path from the context root of the request URL is /jsp/index.jsp, the string is considered to be matching.

- **Exact match**

  If a string other than those mentioned above begins with / as the specified URL pattern and if the relative path from the context root of the request URL is exactly same as this URL pattern, the string is considered as matching.

  Example:

  In the URL pattern /jsp/index.jsp, if the relative path from the context root of the request URL is /jsp/index.jsp, the string is considered as matching.

(2) HTTP response compression rules

The <init-param> tag of the DD specifies the Media-Type of the HTTP response to be compressed and the compression threshold.

Media-Type

If the Media-Type specified as the condition is included in the HTTP response, the HTTP response will be compressed with the HTTP response compression functionality.

Compression threshold

The compression threshold is specified as an integer value from 100 to 2,147,483,647. By default, compression threshold 100 is applied to the all Media-Type.

The compression threshold is used for determining whether to compress the HTTP response depending on the size of the HTTP response body. When the size of the HTTP response body exceeds the size defined in the compression threshold, the HTTP response will be compressed with the HTTP response compression functionality.

When the size of the HTTP response is small, by defining the compression threshold you ensure that the time required for HTTP response compression is not longer than the time required for communication.

An appropriate size is decided for the compression threshold depending on the type of resources you want to compress and the speed of communications line; therefore, it is recommended that the size defined in the compression threshold be acquired using actual measurements and be defined appropriately.

The following shows the rules for specifying parameters:

- If an asterisk (*) is specified in Media-Type, all Media-Type are shown. However, if each Media-Type is specified, the specification for each Media-Type is given priority.
- Media-Type is case-insensitive.
- Media-Type and compression threshold are delimited with one-byte colon (:).
- Delimit multiple specifications with one-byte semicolon (;).
- If the same Media-Type is specified multiple times, the Media-Type specified later is used.
- The linefeed, tabs, and spaces in the parameter value are ignored.
- The one-byte semicolon (;) at the end of the parameter value is ignored.

(3) Notes

Take the following precautions when you set up the HTTP response compression filter:

- Check the validity of the filter initialization parameter when you initialize the filter for response compression. If a problem occurs in the value defined in the initialization parameter, an error occurs in the filter initialization process and the Web application will not start.
- If the request URL matches the URL pattern specified in the `url-mapping` parameter, but if the response compression filter does not match the mapped URL pattern, the response compression rules specified in the `url-mapping` parameter are not applied.
- If multiple URL patterns are mapped to the response compression filter, you need to specify the filter mapping definition in such a way so that the request URL does not match with multiple URL patterns simultaneously. To specify different response compression functionality for multiple URL patterns, specify multiple URL patterns in the `url-mapping` parameter.
- To use the memory session failover functionality, specify the filter mapping definition of the response compression filter under the filter mapping definition of the filter for session failover.
- If the Web application version is Servlet 2.4 or later versions, do not specify the `<dispatcher>` tag of `<filter-mapping>`. Also, if 'FORWARD', 'INCLUDE', and 'ERROR' is specified in the `<dispatcher>` tag, note that an error will occur while starting the Web application and the application cannot be started.

2.10.5 Examples of the DD definitions

This subsection describes the examples of the DD definitions that use the HTTP response compression functionality as the examples for each of the following cases:

- When the compression condition is specified for the body size of the HTTP response
- When the compression condition is specified for the URL pattern
- When the compression condition is specified for the Media-Type of the HTTP response
- When the compression conditions are combined and specified

(1) When the compression condition is specified for the body size of the HTTP response

The following is an example of definition when the compression condition is specified for the body size of the HTTP response:

```
<web-app>
 ...
<!-- The filter for Response Compression Filter -->
<filter>
 <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
 <filter-class>com.hitachi.software.was.web.ResponseCompressionFilter</filter-class>
 <init-param>
 <param-name>url-mapping</param-name>
 <param-value>
 /*=rule1;
 </param-value>
 </init-param>
 <init-param>
 <param-name>rule1</param-name>
 <param-value>
 *: 1000;
 </param-value>
 </init-param>
</filter>
 :
<!-- The filter mappings for Response Compression Filter -->
<filter-mapping>
 <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
 <url-pattern>/*</url-pattern>
</filter-mapping>
 ...
</web-app>
```

In the definition example, the HTTP response for accessing the URL pattern `/*` and the body size exceeding 1,000 bytes is compressed.

(2) When the compression condition is specified for the URL pattern

The following is an example of definition when the compression condition is specified for the URL pattern:

```
<web-app>
 ...
<!-- The filter for Response Compression Filter -->
<filter>
 <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
 <filter-class>com.hitachi.software.was.web.ResponseCompressionFilter</filter-class>
 <init-param>
 <param-name>url-mapping</param-name>
 <param-value>
 /app/dir/*=rule1;
 *.html=rule1;
 </param-value>
 </init-param>
 <init-param>
 <param-name>rule1</param-name>
 <param-value>
 *: 100;
 </param-value>
 </init-param>
</filter>
 ...
<!-- The filter mappings for Response Compression Filter -->
<filter-mapping>
 <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
 <url-pattern>/app/*</url-pattern>
</filter-mapping>
 ...
</web-app>
```

In this definition example, the HTTP response for accessing the URL pattern `/app/*` and fulfilling the following conditions is compressed:

- HTTP response for which the HTTP request URL pattern is `/app/dir/*` and the body size exceeds 100 bytes
- HTTP response for which the HTTP request URL pattern is `*.html` and the body size exceeds 100 bytes

(3) When the compression condition is specified for the Media-Type of the HTTP response

The following is an example of definition when the compression condition is specified for the Media-Type of HTTP response:

```
<web-app>
 ...
<!-- The filter for Response Compression Filter -->
<filter>
 <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
 <filter-class>com.hitachi.software.was.web.ResponseCompressionFilter</filter-class>
 <init-param>
 <param-name>url-mapping</param-name>
 <param-value>
 /*=rule1;
 </param-value>
 </init-param>
 <init-param>
 <param-name>rule1</param-name>
 <param-value>
 text/html: 500;
 application/pdf: 1000;
 </param-value>
 </init-param>
</filter>
 ...
<!-- The filter mappings for Response Compression Filter -->
<filter-mapping>
 <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
 <url-pattern>/*</url-pattern>
</filter-mapping>
 ...
</web-app>
```

In this definition example, the HTTP response for accessing the URL pattern `/*` and fulfilling the following conditions is compressed:

- HTTP response for which Media-Type is text or html and the body size exceeds 500 bytes
- HTTP response for which Media-Type is application or pdf and the body size exceeds 1,000 bytes

(4) When the compression conditions are combined and specified

You can also combine and define the compression conditions as shown in the following example:

```
<web-app>
 ...
<!-- The filter for Response Compression Filter -->
<filter>
 <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
 <filter-class>com.hitachi.software.was.web.ResponseCompressionFilter</filter-class>
 <init-param>
 <param-name>url-mapping</param-name>
 <param-value>
 /app/dir1/*=rule1;
 /app/dir2/*=rule2;
 *.html=rule3;
 </param-value>
 </init-param>
 <init-param>
 <param-name>rule1</param-name>
 <param-value>
 *: 500;
 application/pdf: 1000;
 </param-value>
 </init-param>
 <init-param>
 <param-name>rule2</param-name>
 <param-value>
 application/pdf: 2000;
 </param-value>
 </init-param>
 <init-param>
 <param-name>rule3</param-name>
 <param-value>
 *: 100;
 </param-value>
 </init-param>
</filter>
 ...
<!-- The filter mappings for Response Compression Filter -->
<filter-mapping>
 <filter-name>com.hitachi.software.was.web.ResponseCompressionFilter</filter-name>
 <url-pattern>/app/*</url-pattern>
</filter-mapping>
 ...
</web-app>
```

In this definition example, the HTTP response for accessing the URL pattern `/app/*` and fulfilling the following conditions is compressed:

- HTTP response for which the HTTP request URL pattern is `/app/dir1/*`, Media-Type is other than application or pdf, and the body size exceeds 500 bytes
- HTTP response for which the HTTP request URL pattern is `/app/dir1/*`, Media-Type is application or pdf, and the body size exceeds 1,000 bytes
- HTTP response for which the HTTP request URL pattern is `/app/dir2/*`, Media-Type is application or pdf, and the body size exceeds 2,000 bytes
- HTTP response for which the HTTP request URL pattern is `*.html` and the body size exceeds 100 bytes

2.10.6 Execution environment settings (Web application settings)

To use HTTP response compression based on filtering, you must set up the Web application.

Reference the Web application settings only to set or change the properties of Web applications that do not contain `cosminexus.xml`.

Implement the Web application settings in the execution environment using the server management commands and property files. To define HTTP response compression based on filtering, use the filter property file and WAR property file.

The tags specified in the filter property file and WAR property file correspond to the DD. For details on definitions in the DD (`web.xml`), see *2.10.4 Definition in the DD* and *2.10.5 Examples of the DD definitions*.

2.11 Integrating with an EJB container

A Web container integrates with an EJB container to operate as a J2EE server. This section explains the invocation of the Enterprise Beans. Note that you can even use a business interface to invoke Enterprise Beans.

The following table describes the organization of this section.

Table 2-41: Organization of this section (Integrating with an EJB Container)

Category	Title	Reference
Description	Enterprise Beans invocation method	*2.11.1*
Implementation	Implementation for integrating with an EJB Container	*2.11.2*
Settings	Execution environment settings (J2EE server settings)	*2.11.3*

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

2.11.1 Enterprise Bean invocation method

Invocation methods for Enterprise Beans include invocation by using Lookup and invocation by using DI. When you use the Lookup, the invocation method depends upon whether you use the functionality for switching the CORBA Naming Service.

When you use the functionality for switching the CORBA Naming Service, or when invoking the Enterprise Beans contained in the same EAR

When the Enterprise Beans to be invoked are contained in the same EAR, invoke the Enterprise Beans as shown in the following example:

Even if the Enterprise Beans to be invoked are not present in the same EAR, you can use the functionality for switching the CORBA Naming Service and invoke the Enterprise Beans as shown below. For the functionality used for switching the CORBA Naming Service, see *2.10 Switching the CORBA Naming Service* in the *uCosminexus Application Server Common Container Functionality Guide*.

Example:

```
Context ctx = new InitialContext() ;
Object o = ctx.lookup("java: comp/env/ejb/cart") ;
CartHome h = (CartHome) PortableRemoteObject.narrow(o, CartHome.class) ;
Cart c = h.create() ;
c.call() ;
```

When invoking the Enterprise Beans contained in different EAR without using the functionality for switching the CORBA Naming Service

When you do not use the functionality for switching the CORBA Naming Service, and if the Enterprise Beans to be invoked are contained in different EAR, invoke Enterprise Beans An example is shown below.

Example:

```
Context ctx = new InitialContext() ;
Object o =
 ctx.lookup("HITACHI_EJB/SERVERS/MyServer/EJB/APName/Cart") ;
CartHome h = (CartHome) PortableRemoteObject.narrow(o, CartHome.class) ;
Cart c = h.create() ;
c.call() ;
```

2.11.2 Implementation for integrating with an EJB Container

When using the Enterprise Beans invocation, before deploying the corresponding Web application, acquire the RMI-IIOP stubs (stubs.jar) and the RMI-IIOP interface from the J2EE server, and save in the `WEB-INF/lib` directory of the Web application.

Change the stub names appropriately before saving the stubs (stubs.jar) of multiple RMI-IIOP in the `WEB-INF/lib` directory to avoid the duplication of names.

Note that you can use the dynamic class loading of J2EE server for the RMI-IIOP stubs. For dynamic class loading, see *3.7.3 Dynamic class loading* in the *uCosminexus Application Server EJB Container Functionality Guide*.

When invoking an EJB from WAR, you need to define ejb-ref in the DD of WAR. If, however, you use annotations to define the references, you need not define the reference in `web.xml`. When invoking EJBs of the same application from WAR, you need not include stubs and remote interface in the WAR.

When invoking EJBs running on different application or different J2EE server from WAR, remote interface and stubs are necessary. Specify settings to include the remote interface in WAR and auto-generate the stubs when the J2EE application is started in the execution environment. For details on the execution environment settings, see *2.11.3 Execution environment settings (J2EE server settings)*.

2.11.3 Execution environment settings (J2EE server settings)

Implement the J2EE server settings in the Easy Setup definition file. Specify the definition for integrating with an EJB container in `ejbserver.deploy.stub.generation.scope` in the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file. In this parameter, you specify the stubs for invoking the EJB on other applications by a remote interface.

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

2.12 Connecting to the database

When you use a Web container, you can use a resource adapter of the J2EE service to access the database. When you access the database through a resource adapter, Use a DBConnector. When accessing the database, you can use the following functionality:

- JDBC connection pooling
- JDBC connection sharing
- Distributed transaction

For details on connecting to the database, see *3.6 Connecting to the database* in the *uCosminexus Application Server Common Container Functionality Guide*. For settings of the resource adapter for connecting to the database, see *3.6.9 Settings in the execution environment (settings in the resource adapter)* in the *uCosminexus Application Server Common Container Functionality Guide*.

2.13 Creating threads by a Web container

This section describes the creation of threads by a Web container.

The following table describes the organization of this section.

Table 2-42: Organization of this section (Creating threads by a Web container)

Category	Title	Reference
Description	Types and number of the threads created	*2.13.1*
	Total number of threads created	*2.13.2*

Note:

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

In a Web container, create threads for Web server integration, and for Web applications. Determine if the system has enough resources for these threads.

Note that in Application Server, you can even create and use the threads (user threads) from servlets and JSPs. For details on Application Server functionality that can be used in the user threads, see *2.14 Using the user threads*.

The threads created by a Web container are explained below:

2.13.1 Types and number of the threads created

The following table describes the threads created by a Web container:

Table 2-43: Threads created by a Web container

Classification of threads	Reference
Threads for the simple Web server of a Web container	*(1)*
Threads for using the Web server integration functionality#1	*(2)*
Threads for using the in-process HTTP server#2	*(3)*
Threads for the Web application	*(4)*
Threads for the context used for management	*(5)*
Threads for monitoring the timeout when sending a response	*(6)*

#1

Threads created when the Web server integration functionality is used.

#2

Threads created when the in-process HTTP server is used.

(1) Threads for the simple Web server of a Web container (mandatory)

The types and the number of threads created by the Web container for the simple Web server are explained below:

Threads created

Web container creates the threads for receiving the requests for establishing TCP connection, as well as threads for processing the received requests.

Number of threads

One thread is created for receiving the requests for establishing TCP connection. A minimum of five threads and maximum 100 threads are created for processing the requests received by the simple Web server.

During invocation, five threads are created for processing the requests. When the number of concurrently executing threads exceeds the number of already invoked threads, up to 100 threads are created.

Always use the simple Web server of the Web container.

(2) Threads for using the Web server integration functionality

The types and the number of threads created by the Web container for integrating with the Web server are explained below:

Threads created

Web container creates the threads for receiving requests from the redirector for establishing a connection, as well as threads for processing the requests received from the redirector.

Number of threads

One thread is created for receiving requests from the redirector for establishing a connection. Threads equal to the number of connections between the Web server and the Web container are created for processing the requests received from the redirector

During invocation, threads equal to the number specified in the `webserver.connector.ajp13.max_threads` key of `usrconf.properties` are created for processing the requests received from the redirector (default value of `webserver.connector.ajp13.max_threads` key is `10`).

(3) Threads for using the in-process HTTP server

The types and the number of threads created by the Web container for the in-process HTTP server are explained below:

Threads created

The Web container creates the threads for processing the requests and the threads for monitoring the number of request processing threads.

Number of threads

The threads equal to the number of connections from the Web client or proxy server are created for processing the requests. The request processing threads equal to the number specified in the `webserver.connector.inprocess_http.init_threads` key of `usrconf.properties` are created when the J2EE server starts (default value of `webserver.connector.inprocess_http.init_threads` key is `10`). When the number of concurrently executing threads exceeds the number of threads created during the invocation of the J2EE server, the request processing threads are created with the maximum number of connections from the Web client as the upper limit (default value of `webserver.connector.inprocess_http.max_connections` key is `100`).

Furthermore, one thread is created for monitoring the number of request processing threads.

(4) Threads for the Web application

In each Web application, Web container creates threads for monitoring the valid time period of a session. A minimum of one thread and maximum three threads are created.

(5) Threads for the context used for management

Web container creates two threads for generating two contexts for management.

(6) Threads for monitoring the timeout when sending a response

If you enable the timeout when sending a response, Web container creates one thread for monitoring the send timeout.

2.13.2 Total number of threads created

The total number of threads that the Web container creates by default when the process starts is described separately as the threads created while integrating with the Web server and the threads created while using the in-process HTTP

server. This numeric value, however, does not include threads other than the Web container, and the threads created by JavaVM.

(1) When integrating with the Web server

This subsection explains the total number of threads created when integrating with the Web server. This subsection also explains the number of threads used in Web server integration.

(a) Total number of threads created

The total number of threads created depends upon whether you set the timeout when sending a response.

If you set the timeout when sending a response

Total-number-of-threads = A + B + C + D + E

If you do not set the timeout when sending a response

Total-number-of-threads = A + B + C + D

Legend:

A: Number of threads for the simple Web server of the Web container

B: Number of threads used for integrating with the Web server

C: Number of threads in each Web application

D: Number of threads for the context used for management

E: Number of threads for monitoring the timeout when sending a response

Consequently, the total number of threads created during invocation of a process in Web server integration is as follows:

Total-number-of-threads-in-the-case-you-set-the-timeout-when-sending-a-response

= 6 + 11 + (1 × *number-of-Web-applications*) + 2 + 1

= 20 + *number-of-Web-applications*

Total-number-of-threads-in-the-case-you-do-not-set-the-timeout-when-sending-a-response

= 6 + 11 + (1 × *number-of-Web-applications*) + 2

= 19 + *number-of-Web-applications*

After invocation of a process, the number of threads increases depending on the number of connections to the Web server, and the number of concurrently executing simple Web servers.

(b) Number of threads used for integrating with a Web server

During invocation of a Web container, threads equal to the number specified in the `webserver.connector.ajp13.max_threads` key of `usrconf.properties` are created for the request processing threads when using the Web server integration functionality. Threads equal to the number of connections from the redirector are created subsequently. The maximum number of request processing threads, therefore depends upon the maximum number of connections of the Web server.

Note that when the connection between the redirector and the Web container is disconnected due to timeout, request processing threads are created more than the maximum number of connections to the Web server. The expression for calculating the maximum number of request processing threads is shown below:

- When Cosminexus HTTP Server is used

  Maximum-number-of-request-processing-threads = A + B

  Legend:

  A: Settings of the `ThreadsPerChild` directive of Cosminexus HTTP Server

  B: Number of requests running when the connection between the redirector and the Web container is disconnected due to timeout (maximum value is the value specified for webserver.connector.ajp13.max_threads)

- In the case of using the Microsoft IIS

  Maximum-number-of-request-processing-threads = A × B + C

Legend:

A: Number of threads of Microsoft IIS

B: Number of processes of Microsoft IIS

C: Number of requests running when the connection between the redirector and the Web container is disconnected due to timeout (maximum value is the value specified for webserver.connector.ajp13.max_threads)

(2) When using the in-process HTTP server

The total number of threads created depends upon whether you set the timeout when sending a response.

If you set the timeout when sending a response

Total-number-of-threads = A + B + C + D + E

If you do not set the timeout when sending a response

Total-number-of-threads = A + B + C + D

Legend:

A: Number of threads for the simple Web server of the Web container

B: Number of threads for using the in-process HTTP server

C: Number of threads in each Web application

D: Number of threads for the context used for management

E: Number of threads for monitoring the timeout when sending a response

Therefore, the total number of threads during process invocation when the in-process HTTP server is used is as follows:

Total-number-of-threads-in-the-case-you-set-the-timeout-when-sending-a-response

= 6 + 11 + (1 × *number-of-Web-applications*) + 2 + 1

= 20 + *number-of-Web-applications*

Total-number-of-threads-in-the-case-you-do-not-set-the-timeout-when-sending-a-response

= 6 + 11 + (1 × *number-of-Web-applications*) + 2

= 19 + *number-of-Web-applications*

After invocation of a J2EE server, the number of threads increases depending on the number of connections to the Web clients, and the number of concurrently executing simple Web servers.

2.14 Using the user threads

In Application Server, you can create the threads from the servlets and the JSPs and use them. The threads created explicitly in a program by the user are called *User threads*. This section describes how to use the user threads.

The following table describes the organization of this section.

Table 2-44: Organization of this section (Using the user threads)

Category	Title	Reference
Description	Availability of the functionality in user threads	*2.14.1*
Settings	Setting the permissions for generating user threads	*2.14.2*

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

2.14.1 Availability of the functionality in user threads

This section explains the Application Server functionality available in user threads. For details on how to use user threads, see *6.2.1(15) Using user thread*.

The following table describes whether each functionality provided in an Application Server is available in user threads:

Table 2-45: Availability of the functionality in user threads

Functionality name	Availability	Reference
Using servlet API	P	*(1)*
Invoking an Enterprise Bean	N	--
Naming Service	Y	*(2)*
Resource connection	P	*(3)*
Transaction service	P	
Integrated user management	N	--
Log operation and failure detection	Y	*(4)*
J2EE application operations	P	*(5)*
Using the container extension library	Y	*(6)*

Legend:
Y: Available
P: Partially available
N: Not available
--: Not applicable

The functionality available in the user threads is further classified into detailed functionality, and whether each functionality is available in servlets or JSPs, and user threads is explained below. The notes on using the functionality in user threads are also explained.

(1) Servlet API

When you use the servlet API in user threads, you cannot use the request objects and response objects. Use the servlet API only in request processing threads. For details, see the section on *Thread Safety in the Servlet Specifications*.

(2) Naming Service

The following table describes whether the functionality provided as the Naming Service is available in servlets or JSPs and user threads:

Table 2-46: Availability of the functionality of Naming Service (user threads)

Classification/Functionality name			Servlets or JSPs	User threads
JNDI Lookup	Resource adapter	DB Connector	Y	Y
		DB Connector for Cosminexus RM and Cosminexus RM	Y	Y
		TP1/Message Queue - Access	Y	Y
		uCosminexus TP1 Connector	Y	Y
	Java Mail		Y	Y
	JavaBeans resource		Y	Y
	User transaction		Y	Y

Legend:
Y: Available

When you use the Naming Service, do not stop the application when the user threads are running.

(3) Resource connection and transaction service

The following table describes whether the functionality provided as resource connection and transaction service is available in servlets or JSPs and user threads:

Table 2-47: Availability of the functionality of resource connection and transaction management (user threads)

Classification/Functionality name		Servlets or JSPs	User threads
Connection pooling	Connection pooling by DB Connector	Y	Y
	Connection pooling by DB Connector for Cosminexus RM and Cosminexus RM	Y	Y
	Connection pooling with uCosminexus TP1 Connector	Y	Y
	Connection pooling to TP1/Message Queue - Access	Y	Y
	Connection pooling to SMTP server	--	--
	Warming up of connection pool	Y	Y
	Adjusting the number of connections	Y	Y
Connection sharing		Y	Y
Connection association		Y	Y
Statement pooling (DB Connector)		Y	Y
Light transaction		Y	Y
In-process transaction service		Y	Y
DataSource object caching		Y	Y

Classification/Functionality name		Servlets or JSPs	User threads
Optimizing container management sign-on in DB Connector		Y	Y
Pooling the receiving buffer		Y	Y
Detecting a connection failure	Failure detection by DB Connector	Y	Y
	Failure detection by DB Connector for Cosminexus RM and Cosminexus RM	Y	Y
	Detecting connection failure with uCosminexus TP1 Connector	--	--
	Detecting connection failure to TP1/Message Queue - Access	--	--
	Detecting connection failure of SMTP server	--	--
Waiting to acquire connections in the case of connection depletion		Y	Y
Retrying to acquire connection	Retrying connection acquisition by DB Connector	Y	Y
	Retrying connection acquisition by DB Connector for Cosminexus RM and Cosminexus RM	Y	Y
	Retrying acquisition of connection with uCosminexus TP1 Connector	Y	Y
	Retrying acquisition of connection to TP1/Message Queue - Access	Y	Y
	Retrying acquisition of connection to SMTP server	--	--
Connection pool clear		Y	Y
Closing and releasing a connection	Automatic closing of connection (Web container)	Y	--
Connection sweeper		Y	Y
Transaction timeout		Y	Y
Transaction recovery		Y	Y
Automatic conclusion of transaction#		Y	N
SQL output for troubleshooting		Y	Y
Connection pool clustering		Y	Y

Legend:

Y: Available

--: Not available

#

This functionality is used for looking up an unconcluded transaction when returning from a method of the servlet.

The precautions when using the resource connection and transaction service in user threads are as follows:

- You cannot start and conclude a transaction, and acquire and release a connection on threads generated in a servlet in which the SingleThreadModel interface is implemented.
- A transaction cannot be inherited when threads are generated.
- Start and conclude a transaction on the same thread.
- You cannot pass a connection between threads. If a connection is used, the operation becomes invalid.
- When a connection is acquired in user threads, ensure that you close the connection on the thread in which the connection was acquired.

(4) Log operation and failure detection

The following table describes whether the functionality provided as log operation and error detection is available in servlets or JSPs and user threads:

Table 2-48: Availability of the functionality of log operation and failure detection (user threads)

Classification/Functionality name	Servlets or JSPs	User threads
Automatic execution of processing by Management events	Y	Y
Monitoring a system by JP1 event	Y	Y
User log output	Y	Y
Performance analysis trace	Y	Y
Monitoring the CTM statistics	Y	Y

Legend:
Y: Available

(5) J2EE application operations

The following table describes whether the functionality provided for operating a J2EE application is available in servlets or JSPs and user threads:

Table 2-49: Availability of the functionality for operating a J2EE application (User threads)

Classification/Functionality name		Servlets or JSPs	User threads
Controlling the number of concurrently executing threads in the Web container		Y	--
Dynamically changing the number of concurrently executing threads in the schedule queue		Y	Y
Monitoring the execution time of a J2EE application	Method timeout functionality	Y	--
	Method cancellation functionality	Y	--
Terminating the J2EE application	Normal termination	Y	Y#1
	Forced termination	Y	Y#1
Switching the J2EE application	Switching by redeploy functionality	Y	Y#2
	Switching by reload functionality	Y	Y

Legend:
Y: Available
--: Not available

#1
Do not stop user threads in the container.

#2
Do not stop user threads in the container when switching the J2EE applications that are started.

(6) Container Extension Library

The container extension library is available in both servlets or JSPs and user threads.

2.14.2 Setting the permissions for generating user threads

To generate threads that the user explicitly generates in a program (user threads), permissions must be granted to the target servlet and JSP to generate the thread. This subsection describes the permission settings for generating the user threads.

To generate the user threads, confirm that the following coding exist in the `server.policy`. This definition grants the permission to generate the user threads.

```
permission java.lang.RuntimePermission "modifyThread";
permission java.lang.RuntimePermission "modifyThreadGroup";
```

Use the Smart Composer functionality commands to specify `server.policy` after the system is built. The coding example of the `server.policy` file is as follows:

```
...
//
// Grant access 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.lang.RuntimePermission "modifyThread";
permission java.lang.RuntimePermission "modifyThreadGroup";
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.*";
};
...
```

2.15 Overview of controlling the number of concurrently executing threads

In the Web container, the servlet requests are processed in multi-threads. You can set the upper limit for the number of threads that you can execute concurrently. By setting the upper limit, you can prevent a decline in the performance due to slashing. If you set an appropriate number of threads, you can tune the performance as per the access status.

This section describes the settings for controlling the number of concurrently executing threads.

The following table describes the organization of this section.

Table 2-50: Organization of this section (Controlling the number of concurrently executing threads)

Category	Title	Reference
Description	Control units of the number of threads	*2.15.1*
	Parameters for controlling the number of concurrently executing threads	*2.15.2*
	Number of threads used in error processing of static contents and requests	*2.15.3*

Note:

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

2.15.1 Control units of the number of threads

The methods for controlling the number of concurrently executing threads include the controlling in Web container, in Web application, and in URL group.

- Controlling the number of concurrently executing threads in a Web container

  Set the number of threads for processing the requests concurrently in all Web applications on a Web container. For details, see *2.16 Controlling the number of concurrently executing threads in the Web container*.

- Controlling the number of concurrently executing threads in a Web application

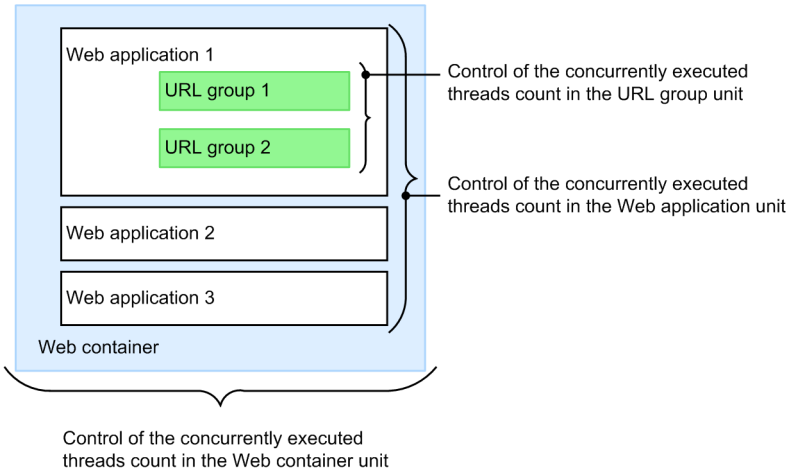
  Set the number of threads for processing the requests concurrently in each Web application on a Web container. You can control the number of threads more carefully as compared to the method for controlling the number of concurrently executing threads in Web container. For details, see *2.17 Controlling the number of concurrently executing threads in the Web application*.

- Controlling the number of concurrently executing threads in a URL group

  Set the number of threads for processing the requests concurrently in URL corresponding to business logic, such as servlet and JavaBeans, in a Web application. A business logic that processes the requests sent to a specific URL is called a *URL group*. Since the number of concurrently executing threads is controlled in URL group, you can control the number of threads more carefully as compared to the method for controlling in Web application. For details, see *2.18 Controlling the number of concurrently executing threads in the URL group*.

The following figure shows the relationship between each control unit:

Figure 2-16: Relationship between each control unit of the number of concurrently executing threads



As shown in the figure, the biggest unit of controlling the number of concurrently executing threads is the Web container. In the case of controlling the number of threads in each Web application of the Web container, specify the control in the Web application. In the case of controlling the number of threads in URL group of a Web application, specify the control in the URL group. The smallest unit of controlling the number of threads is the URL group.

Controlling the number of threads has an inclusion relation, and hence, in the case of controlling the number of threads in the Web application, you also need to specify the setting in the Web container. Moreover, in the case you control the number of threads in the URL group, you also need to specify the setting in the Web container and the Web application.

2.15.2 Parameters for controlling the number of concurrently executing threads

Control the number of concurrently executing threads by parameters, such as the maximum number of concurrently executing threads, the number of dedicated threads, and the size of a pending queue.

This subsection explains the main parameters for controlling the number of threads.

(1) Maximum number of concurrently executing threads

From among the total number of available threads, the maximum number of concurrently executing threads refers to the number of threads that can concurrently execute the maximum number of requests in which the number of concurrently executing threads is to be controlled.

Set the maximum number of concurrently executing threads in a Web container, Web application, and URL group.

(2) Number of dedicated threads

From among the total number of available threads, the number of dedicated threads refers to the number of threads that can definitely execute the requests in which the number of concurrently executing threads is to be controlled. By specifying the control in the Web application and URL group, you can secure the minimum number of threads in each Web application or each URL group.

(3) Size of a pending queue

When the requests in which the number of concurrently executing threads is to be controlled, reach the upper limit of the number of concurrently executing threads, you can specify the size of the request queue. Specify the number of requests to be stored in the queue as the queue size.

The conditions for saving requests in a pending queue are as follows:

- When available number of the shared threads is not specified, in the case *number-of-concurrently-executing-threads* < *maximum-number-of-concurrently-executing-threads*, and *number-of-concurrently-executing-threads* ≥ *number-of-dedicated-threads*
- When *number-of-concurrently-executing-threads* ≥ *maximum-number-of-concurrently-executing-threads*

Note that if space is not available in a pending queue, the requests are not processed and an error is returned to the client.

You can set the size of a pending queue in the Web application and the URL group.

(4) Number of shared threads

The number of shared threads refers to the number of non-dedicated threads, from among the available threads. The number of shared threads includes the number of shared threads of the Web container, and the number of shared threads of the Web application.

- Number of shared threads in a Web container

  The number of shared threads in a Web container is the number of threads shared by all the Web applications deployed on the Web container.
- Number of shared threads in a Web application

  The number of shared threads in a Web application is the number of threads shared by all the processes included in the Web application.

The number of shared threads is calculated from the maximum number of concurrently executing threads and the number of dedicated threads.

For details on how to calculate the number of shared threads, see *(5) Method to calculate the number of shared threads*.

(5) Method to calculate the number of shared threads

This point describes how to calculate the number of shared threads in a Web container and the number of shared threads in a Web application. When you specify the settings to control the number of concurrently executing threads in the Web application, the number of shared threads in the Web application depends upon whether you specify the settings to control the number of concurrently executing threads in the URL group.

Note that the URL group does not have any shared threads. When you specify the method for controlling the number of concurrently executing threads in URL group of a Web application, the number of shared threads of the Web application is used.

- Number of shared threads in the Web container

  When a Web application with the number of dedicated threads specified exists on the Web Container, the number of shared threads will be as follows:

  Total-number-of-shared-threads-in-the-Web-container =

  Maximum-number-of-concurrently-executing-threads-in-the-Web-container − *Total-number-of-dedicated-threads-in-a-Web-application*[#]

  # Total number of dedicated threads that are set in all Web applications deployed on the Web container.

  The number of dedicated threads set in each Web application is the minimum number of threads to be secured in the Web application. This thread count is not used in the request processing of another Web applications.
- Number of shared threads in a Web application (When control of the number of concurrently executing threads in each URL group is specified)

  Number-of-shared-threads-in-a-Web-application =

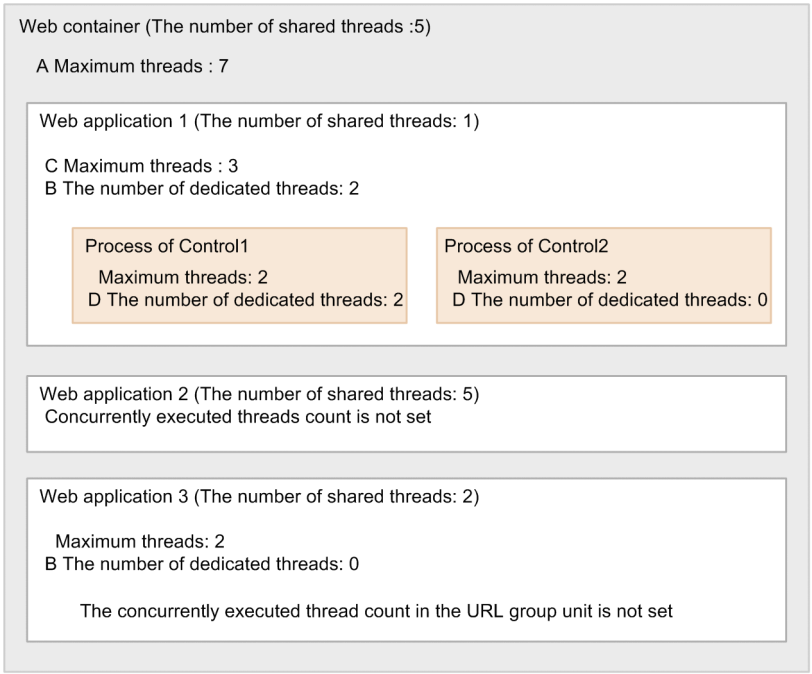
  Maximum-number-of-concurrently-executing-threads-in-a-Web-application-unit − *Total-number-of-dedicated-threads-in-URL-group*[#]

  # Total number of dedicated threads in all the URL groups set in a Web application.
- Number of shared threads in a Web application (When control of the number of concurrently executing threads in each URL group is not specified)

Number-of-shared-threads-in-a-Web-application = Maximum-number-of-concurrently-executing-threads-in-the-Web-application

The following figure shows an example for calculating the number of shared threads in a Web container and Web application when the number of concurrently executing threads is specified:

Figure 2-17: Example for calculating the number of shared threads



- **The number of shared threads in a Web Container = A - (total of B)**
  In this figure, the number of shared threads in a Web Container is 7 - (2 + 0) = 5.
- **The number of shared threads in Web application 1 = C - (total of D)**
  The number of dedicated threads in each URL group is specified in Web application 1. In this figure, the number of shared threads is 3 - (2 + 0) = 1
- **The number of shared threads in Web application 2 = The number of shared threads in the Web Container**
  The settings for controlling the number of concurrently executing threads are not specified in the Web application 2. Therefore, the number of shared threads in the Web Container is applied to the number of shared threads in the Web application 2. In the figure, the number of shared threads is 5.
- **The number of shared threads in Web application 3 = The maximum number of concurrently executing threads**
  The number of dedicated threads in each URL group is not specified in Web application 3. Therefore, number of shared threads the Web application 3 is applied to the number of shared threads in Web application 3. In this figure, the number of shared threads is 2.

2.15.3 Number of threads used in error processing of static contents and requests

The maximum number of concurrent connections of a Web server is used to: a) Process the requests in the Web container b) Process the static contents deployed on the Web server c) Send the requests of the Web container and perform the error processing of the requests that exceed the number of concurrently executing threads and the pending queue. The number of threads used for processing the static contents, and for error processing of the requests that exceed the size of pending queue is as follows:

Number-of-threads-used-for-processing-the-static-contents-and-error-processing-of-requests =
Maximum-number-of-concurrent-connections-of-a-Web-server − (*Maximum-number-of-concurrently-executing-threads-in-Web-container* + *total-size-of-the-pending-queue-of-Web-application-and-URL-group-and-default-pending-queue*)

Note that the number of threads used for static contents is not assigned in the default settings. In the case of securing the number of threads for processing the static contents, specify appropriate values for the size of the pending queue of the Web application, URL group, and default pending queue to satisfy the above formula.

2.16 Controlling the number of concurrently executing threads in the Web container

This section describes the settings for controlling the number of concurrently executing threads in the Web Container.

The following table describes the organization of this section.

Table 2-51: Organization of this section (Controlling the number of concurrently executing threads in the Web Container)

Category	Title	Reference
Description	Mechanism for controlling the number of concurrently executing threads (Web container)	*2.16.1*
Settings	Execution environment settings (J2EE server settings)	*2.16.2*

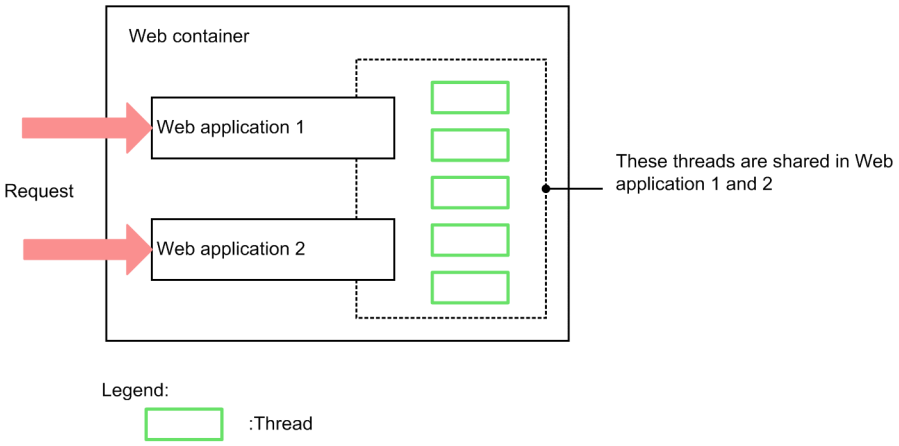
Note:

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

2.16.1 Mechanism for controlling the number of concurrently executing threads (Web container)

The following figure shows the mechanism of controlling the number of concurrently executing threads in the Web container:

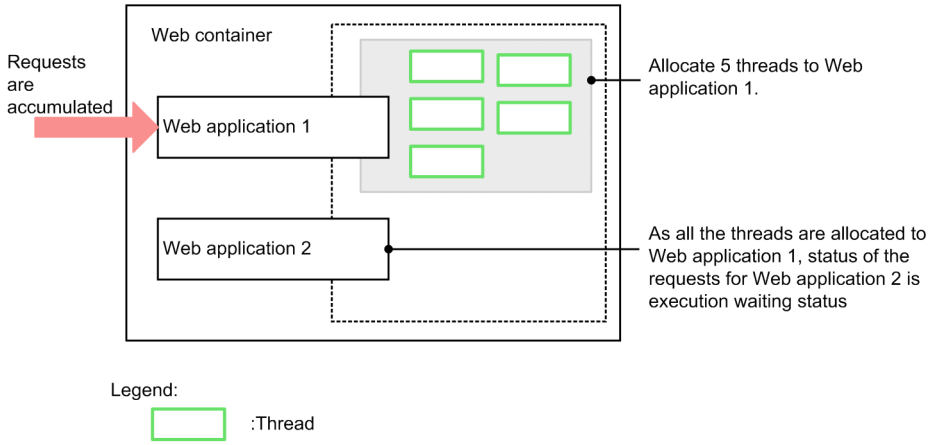
Figure 2-18: Controlling the number of concurrently executing threads in the Web container



For example, if two Web applications are deployed on the Web container, and 5 is set as the number of concurrently executing threads, the number of threads that you can concurrently execute in two Web applications will be 5.

Even when the access is centralized in one of the multiple Web applications deployed on the Web container, by setting the number of concurrently executing threads in the Web container, you can assign the threads to the Web application in which the access is centralized. The following figure shows this mechanism:

Figure 2-19: Handling of threads when the access is centralized (For Web containers)



As shown in the figure, when two Web applications are deployed on the Web container and '5' is set as the number of concurrently executing threads, all the five threads are assigned to the Web application 1 if the requests are concentrated in Web application 1.

On the other hand, the requests for the Web application 2 are accumulated in the pending queue of the Web container until the request processing of the Web application 1 is complete. Note that the requests accumulated in the pending queue of the Web container are executed in a sequence, after the request processing is complete.

2.16.2 Execution environment settings (J2EE server settings)

Implement the J2EE server settings in the Easy Setup definition file. To define the settings for controlling the number of concurrently executing threads in the Web container, specify one of the following parameters in the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file:

- `webserver.connector.ajp13.max_threads`

  Set the maximum number of concurrently executing threads in the entire Web container. Specify this parameter in the case of Web server integration.

- `webserver.connector.inprocess_http.max_execute_threads`

  Set the maximum number of concurrently executing threads in the entire Web container. Specify this parameter when using the in-process HTTP server.

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

2.17 Controlling the number of concurrently executing threads in the Web application

This section describes the settings for controlling the number of concurrently executing threads in the Web applications.

Note that when you control the number of threads in the Web application, you also need to simultaneously specify the settings to control the number of threads in the Web container. Also, see *2.16 Controlling the number of concurrently executing threads in the Web container*.

The following table describes the organization of this section.

Table 2-52: Organization of this section (Controlling the number of concurrently executing threads in the Web applications)

Category	Title	Reference
Description	Mechanism for controlling the number of concurrently executing threads (Web applications)	*2.17.1*
	Parameters required for controlling the number of concurrently executing threads (Web applications)	*2.17.2*
	Guidelines for the settings for number of concurrently executing threads (Web applications)	*2.17.3*
Implementation	Definition in cosminexus.xml	*2.17.4*
Settings	Execution environment settings	*2.17.5*
	Examples of setting the number of concurrently executing threads and the size of a pending queue (Web application)	*2.17.6*
Notes	Notes on controlling the number of concurrently executing threads in the Web application	*2.17.7*

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

2.17.1 Mechanism for controlling the number of concurrently executing threads (Web applications)

If you want to control the number of concurrently executing threads more carefully than the Web container, control the number of concurrently executing threads in the Web application.

When you set the number of concurrently executing threads in the Web application, an upper limit is set for the number of concurrently executing threads in each Web application. As a result, when the number of requests to a specific Web application increases, you can prevent that Web application from occupying the capacity of the entire Web container, and can also ensure smooth execution of other businesses.

For guidelines on setting the number of concurrently executing threads in the Web application, see *8.3.4 Controlling the number of concurrently executing threads of a Web application* in the *uCosminexus Application Server System Design Guide*. Also, for the parameters specified in the Web container, see *2.16.2 Execution environment settings (J2EE server settings)*.

2.17.2 Parameters required for controlling the number of concurrently executing threads (Web applications)

An overview of the parameters required for controlling the number of threads in each Web application is as follows:

- Specifying the setting to control the number of concurrently executing threads in the Web container

Set the maximum number of concurrently executing threads in the Web container. Note that the maximum number of concurrently executing threads set here is shared in all the Web applications deployed on the Web container.

- Specifying the setting to control the number of concurrently executing threads in the Web application

Set the following parameters in the Web application for which you want to control the number of concurrently executing threads:

- Maximum number of concurrently executing threads
  Set the maximum number of threads that you can concurrently execute in a Web application.
- Number of dedicated threads
  Set the number of dedicated threads of a Web application.
- Size of the pending queue of the Web application
  Set up the size of the pending queue of the Web application.
- Default size of a pending queue
  Set the size of the pending queue for a Web application in which you do not specify the setup to control the number of concurrently executing threads in the Web application.

The details on the parameters required for specifying to control the number of concurrently executing threads in the Web applications are as follows:

(1) Maximum number of concurrently executing threads in the Web application

If the value for maximum number of concurrently executing threads in the Web application is set, that value is applied. This subsection explains the concept of the maximum number of concurrently executing threads in the Web applications for which the number of concurrently executing threads and the number of dedicated threads is not set.

(a) In the case of the Web application in which you do not specify the setting to control the number of concurrently executing threads

The maximum number of concurrently executing threads of the Web application in which you dot set the maximum number of concurrently executing threads, is as follows:

Maximum-number-of-concurrently-executing-threads =

Maximum-number-of-concurrently-executing-threads-in-the-Web-container − *Total-number-of-dedicated-threads-in-a-Web-application*[#]

#

Total number of dedicated threads that are set in all Web applications deployed on the Web container.

(b) In the case of the Web application in which dedicated threads are not set

When specifying the setting to control number of concurrently executing threads in the Web application, the setting of the number of dedicated threads is optional. The smaller value in the following two values is applicable as the maximum number of concurrently executing threads of a Web application in which the number of dedicated threads is not set:

Maximum-number-of-concurrently-executing-threads =

Maximum-number-of-concurrently-executing-threads-set-in-the-Web-application

or

Maximum-number-of-concurrently-executing-threads-in-the-Web-container − *Total-number-of-dedicated-threads-in-a-Web-application*[#]

#

Total number of dedicated threads set in all the Web applications deployed on the Web container.

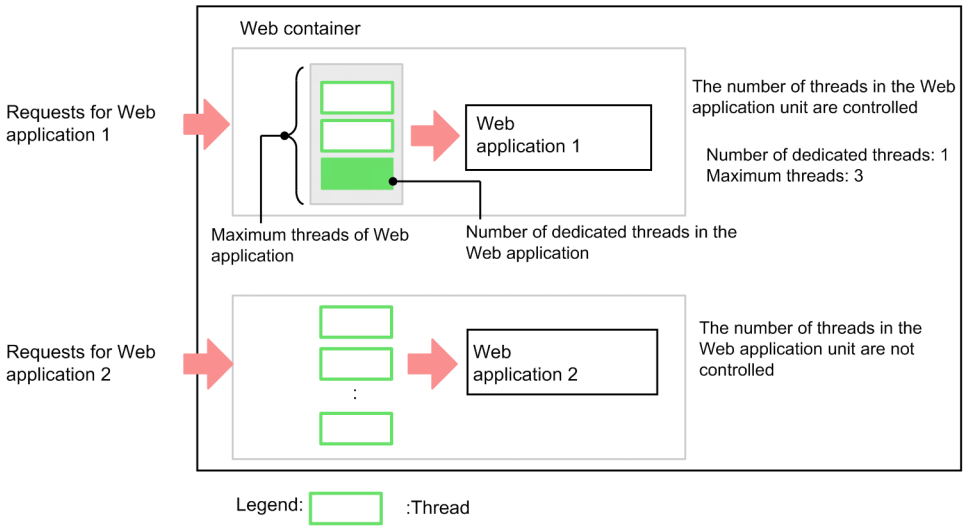
(2) Number of dedicated threads of the Web application

When only the number of concurrently executing threads in the Web container is set, and if the access is centralized to another Web application in the Web container, the threads are used in the application for which the access is centralized. You can secure the minimum number of threads required for execution in the Web application by setting the number of dedicated threads. Therefore, even if the access is centralized to another Web application in the Web container, you can execute the requests without waiting.

(a) Operation of the Web application depending on the presence or absence of the settings for the number of dedicated threads

The following figure shows the number of dedicated threads in the Web application, and illustrates the operations of two Web applications in which one is without the number of dedicated threads set and the other is with the number of dedicated threads set:

Figure 2-20: Number of dedicated threads in the Web applications



The contents of the figure are explained below. Web application 1 and Web application 2 are running in the Web container. The settings to control the number of concurrently executing threads are not specified in Web application 2, while it is specified in Web application 1. In Web application 1, the maximum number of concurrently executing threads is set as 3, and the number of dedicated threads is set as 1.

For example, if the access is centralized in Web application 2, and the number of dedicated threads is not set in Web application 1, all the threads are used in Web application 2. As illustrated in the figure, by setting the number of dedicated threads in Web application 1, you can secure at least one thread in Web application 1 even when the access is centralized in Web application 2. Consequently, you can ensure the processing of Web application 1 in which the number of dedicated threads is set.

As described in the above example, if you set the number of dedicated threads, then even if the access is centralized in another business, you can certainly execute the Web application. Therefore, Hitachi recommends that you set the number of dedicated threads in highly important Web applications, such as applications for management.

Note that the threads equal to the specified number of dedicated threads are not used in request processing of another applications. It is optional to set the number of dedicated threads in the settings for controlling the number of concurrently executing threads in Web application.

(b) Number of dedicated threads and maximum number of concurrent connections

When the maximum number of concurrent connections of the Web server (when the Web server integration functionality is used) or the maximum number of concurrent connections from the Web client (when the in-process HTTP server is used) are less and if the maximum number of concurrent connections are occupied by the requests to the Web applications in which the number of dedicated threads is not set, even when the Web applications with the number of dedicated threads set are accessed, the requests remain pending and an error occurs on the Web server or on the in-process HTTP server.

To properly execute a Web application in which the number of dedicated threads is set, without any dependency on the access flow to other Web applications, you need to set an appropriate value in the maximum number of concurrent connections. The method for setting the maximum number of concurrent connections is explained below:

- **Method for setting the maximum number of concurrent connections of a Web server (when the Web server integration functionality is used)**

  When you use the Web server integration functionality, you need to set a value greater than that shown below as the maximum number of concurrent connections in the Web server:

Maximum-number-of-concurrent-connections-of-a-Web-server > *Total-size-of-the-pending-queue-of-Web-application-and-default-pending-queue* + *Maximum-number-of-concurrently-executing-threads-in-the-Web-container*

Set an appropriate value for the number of dedicated threads so that the above expression is satisfied.

Note that the maximum number of concurrent connections of a Web server is set at the following location:

When Cosminexus HTTP Server is used

`ThreadsPerChild` directive of `httpsd.conf` (in Windows) or `MaxClients` directive of `httpsd.conf` (in UNIX)

In the case of using the Microsoft IIS

Number of connections to the client as set in [Internet Service Manager]

For details on the settings for using Cosminexus HTTP Server, see *uCosminexus Application Server HTTP Server User Guide*. For details on the settings for using Microsoft IIS, see *Microsoft IIS Help*.

- **Method for setting the maximum number of concurrent connections from the Web client (When the in-process HTTP server is used)**

  When you use the in-process HTTP server, you need to set a value greater than that shown below as the maximum number of concurrent connections from the Web clients:

  Maximum-number-of-concurrent-connections-from-Web-clients > *Total-size-of-the-pending-queue-of-Web-application-and-default-pending-queue* + *Maximum-number-of-concurrently-executing-threads-in-the-Web-container*

  Note that the number of concurrent connections from the Web client is a value obtained by subtracting the number of requests that were denied connection from the maximum number of connections from the Web client. For details, see *5.5 Controlling the flow of requests by controlling the number of concurrent connections from the Web client*.

(3) Size of a pending queue of Web application

Set the size of a pending queue of a Web application.

If you set the maximum number of concurrently executing threads in the Web application, the requests are accumulated in a queue when the number of executing threads reaches the maximum number. At this point, you can specify the size of the pending queue in the Web application.

The setting up of the size of the pending queue of a Web application depends upon whether you specify the number of concurrently executing threads in the Web application.

- When you set the number of concurrently executing threads in the Web application

  You can set the size of a pending queue in each Web application.

- When you do not set the number of concurrently executing threads in the Web application

  A common size of the pending queue is used in the Web application. The common pending queue size is called the *default pending queue size*.

Operations of the pending queue of the Web application and default pending queue

In the case of multiple setting (including the setting for size of a pending queue of Web application and the default size of a pending queue) of a pending queue, the requests executed by using the number of shared threads are processed in an order starting from the first arrived request in the queue.

Maximum number of concurrent connections of the Web server and the pending queue of the Web application and default pending queue (when the Web server integration functionality is used)

The maximum number of concurrent connections in the Web server is the upper limit for the multiplicity of the requests transferred to the Web container from the Web server. As a result, when setting the number of dedicated threads, specify a number smaller than the maximum number of concurrent connections in the Web server, as the size of the pending queue of the Web application and default pending queue.

Maximum number of concurrent connections from the Web clients and the pending queue of the Web application and default pending queue (when the in-process HTTP server is used)

The upper limit value for the multiplicity of the number of connections from the Web client is as follows:

Maximum-number-of-concurrent-connections-from-the-Web-client − *Number-of-requests-for-which-connection-is-denied*

As a result, when setting up the number of dedicated threads, specify a number smaller than the maximum number of concurrent connections from the Web clients, as the size of the pending queue of the Web application and default pending queue.

Note that if the maximum number of concurrent connections from the Web client is greater than *Sum-of-size-of-the-pending-queue-of-Web-application-and-default-pending-queue + Maximum-number-of-concurrently-executing-threads-in-Web-Container*, you are not required to control the number of concurrent executions from the Web client with the in-process HTTP server. Also, if the maximum number of concurrent connections from the Web client is smaller than *Sum-of-size-of-the-pending-queue-of-Web-application-and-default-pending-queue + Maximum-number-of-concurrently-executing-threads-in-Web-Container*, an error can be returned immediately to the client without the occurrence of connection pending, by controlling the number of concurrent executions from the Web client through the in-process HTTP server.

Tip

Operation of the requests when the number of concurrently executing threads reaches the maximum number

When the number of concurrently executing threads in Web application reaches the maximum number, requests are accumulated in the queue, if there is a space in the queue. Once the on-going request processing finishes, requests are extracted sequentially from the queues and are executed. Note that if there is no space in the queue, the requests result in an error, and error HTTP 503 is returned to the client.

2.17.3 Guidelines for the settings for number of concurrently executing threads (Web applications)

This subsection provides the guidelines for settings when you want to control the number of concurrently executing threads in the Web applications.

- Follow this order to determine the value you want to set:
  1. Determine the maximum number of concurrently executing threads in the Web container.
  2. Determine the maximum number of concurrently executing threads in the Web applications.
  3. Determine the number of dedicated threads in the Web applications.
  4. Determine the size of the pending queue of the Web application and default pending queue.
- Specify the settings for the number of concurrently executing threads in the Web applications keeping in mind the operations of the Web applications and the capacity of the host on which the J2EE server is running. Execute the Web applications and evaluate whether the set value is valid.

  You can check the statistics for a running Web application by using Management Server. For details on checking the operational status of the Web applications, see *2.19.2(1) Confirming the operational status of a Web application*.
- When you set the number of dedicated threads for multiple Web applications, the total of the number of dedicated threads in each Web application must be less than the maximum number of concurrently executing threads of the Web container. Determine the value to be specified along with the settings for controlling the number of concurrently executing threads in the Web container.
- Note that when the maximum number of concurrently executing threads in the Web container is small, the maximum number of concurrently executing threads in the Web application might be even smaller than the set up number.
- The number of dedicated threads must be less than the maximum number of concurrently executing threads in the Web application.

2.17.4 Definition in cosminexus.xml

This subsection describes the definitions in `cosminexus.xml` required in the application development environment.

Specify the definition for controlling the number of threads in the Web application in the `<war>` tag of `cosminexus.xml`.

The following table lists the definitions in `cosminexus.xml` for controlling the number of threads in the Web applications:

Table 2-53: Definitions in cosminexus.xml for controlling the number of threads in the Web applications

Items	Tag to be specified	Setting contents
Controlling the number of concurrently executing threads in a Web application#	<thread-control-max-threads> tag in <thread-control> tag	The maximum number of threads that you can concurrently execute in a Web application can be specified.
Number of dedicated threads in the Web application.	<thread-control-exclusive-threads> tag in <thread-control> tag	You can specify the minimum number of threads to be secured in the Web application (number of dedicated threads). If you do not want to specify the number of dedicated threads, specify 0.
Pending queue for each Web application	<thread-control-queue-size> tag in <thread-control> tag	When the number of executing threads reaches the maximum number, the request is accumulated in a queue. You specify the pending queue size used at this time for each Web application.

Note:

You can also dynamically change the settings for the number of concurrently executing threads in the Web applications by using the management command (mngsvrutil). As a result, you can change the setting of the number of concurrently executing threads without stopping the service of running Web applications. For details on dynamically changing the maximum number of concurrently executing threads of the Web applications, see *2.19 Dynamic change in the number of concurrently executing threads*.

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*.

2.17.5 Execution environment settings

This subsection explains the settings to control the number of threads in the Web application.

To control the number of threads in the Web application, you must set up the J2EE server and J2EE applications.

Reference the J2EE application settings only when you want to set or change the properties of the J2EE applications that do not contain cosminexus.xml.

(1) Setting up the J2EE server

Implement the J2EE server settings in the Easy Setup definition file. Specify the definition for controlling the number of threads in the Web applications in the <configuration> tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file.

The following table lists the definition in the Easy Setup definition file for controlling the number of threads in the Web applications:

Table 2-54: Definition in the Easy Setup definition file for controlling the number of threads in the Web applications

Items	Parameter to be specified	Setting contents
Controlling the number of concurrently executing threat	webserver.container.thread_control.enabled	Specifies whether the number of concurrently executing threads in the Web applications will be controlled. If you specify true, the controlling of the number of concurrently executing threads in the Web applications is enabled. If you specify false, the controlling of the number of concurrently executing threads in the Web applications is disabled and the number of concurrently executing threads in the Web Container is controlled.
Default size of a pending queue	webserver.container.thread_control.queue_size	When the number of executing threads reaches the maximum number, the request is accumulated in a queue. This parameter specifies the common pending

Items	Parameter to be specified	Setting contents
Default size of a pending queue	`webserver.container.thread_control.queue_size`	queue size for each Web application at this time (default pending queue size). The set value is applied when the number of concurrent executions in the Web applications is not specified.

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

(2) Setting up the J2EE application

Implement the J2EE application settings in the execution environment by using the server management commands and property files. Use the WAR property file to define the controlling of the number of threads in the Web application.

The tags specified in the WAR property file correspond to `cosminexus.xml`. For details on the definitions in `cosminexus.xml`, see *2.17.4 Definition in cosminexus.xml*.

2.17.6 Example of setting the number of concurrently executing threads and the size of a pending queue (Web application)

This subsection describes the examples for setting up the number of concurrently executing threads and the size of pending queues of Web applications. The examples explained here refer to the Web applications in which the Web server integration functionality is used.

(1) Example of setting the Web applications used in the description

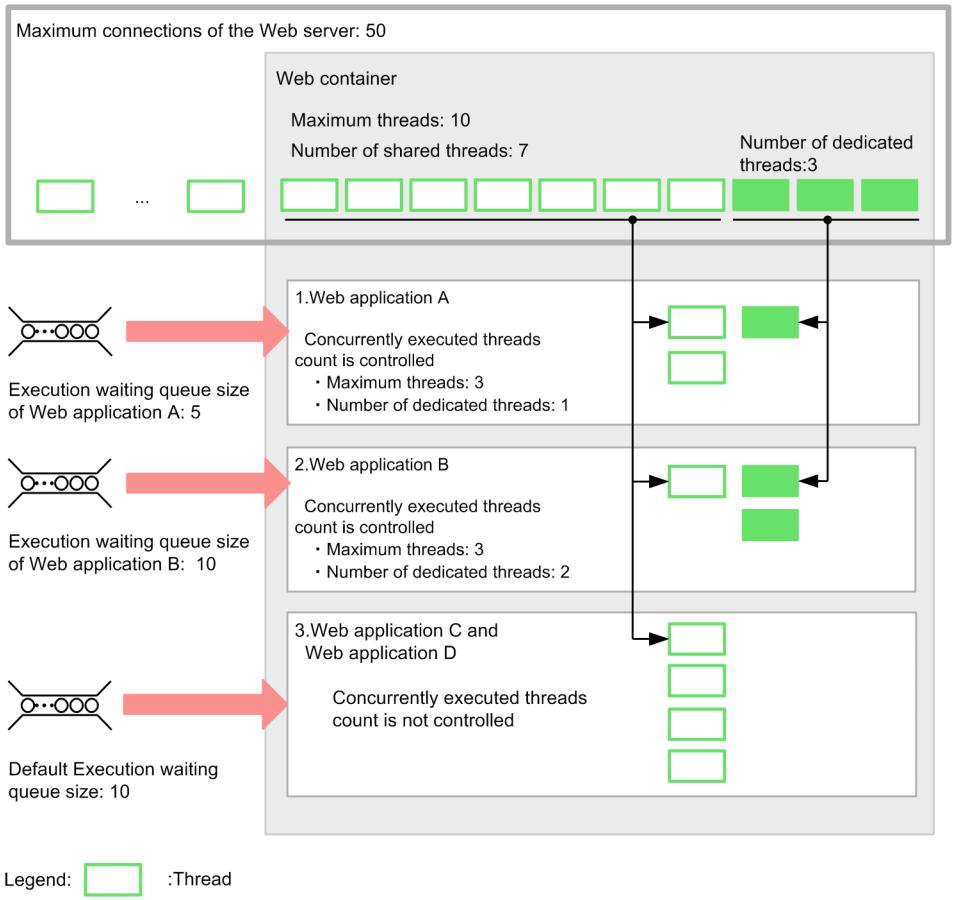
In this example, the setting to control the number of concurrently executing threads in Web application is specified in two Web applications out of the four Web applications deployed on the Web container. The settings are shown below:

- Maximum number of concurrent connections in the Web server: 50
- Maximum number of concurrently executing threads of Web container: 10
- Default size of the pending queue: 10
- Specifying the setting to control the number of concurrently executing threads in the Web application
  This is assumed that the following settings are specified in Web applications A and B respectively.
  In the Web applications C and D, the settings for controlling the number of concurrently executing threads in the Web applications are not specified.

Web application name	Maximum number of concurrently executing threads	Number of dedicated threads	Size of pending queue of Web application
Web application A	3	1	5
Web application B	3	2	10

The following figure illustrates the example in which the Web application with the setting to control the number of concurrently executing threads for each Web application, and Web applications without the setting exist together on a Web container:

Figure 2-21: Example of Web application settings



(2) Number of threads available in each Web application

The maximum number of concurrently executing threads, the number of dedicated threads, and the size of the pending queue that you can use in the case of the setup in figure 2-21 are illustrated below for each Web application. Note that the serial number of the description corresponds to the serial number mentioned in the figure.

1. Web application A
   - Maximum number of concurrently executing threads and the number of dedicated threads

     Since the maximum number of concurrently executing threads and the number of dedicated threads is set in Web application A, you can use the threads up to the respective set values.

     You can concurrently execute maximum of 3 threads in the Web application A. Out of the three threads you can secure minimum of one thread for the number of dedicated threads in Web application A.
   - Size of the pending queue of the Web application

     The size of the pending queue of the Web application is specified in the Web application A. If three threads are used concurrently in the Web application A, a maximum of five requests to the Web application A are accumulated in the queue.
2. Web application B
   - Maximum number of concurrently executing threads and the number of dedicated threads

     Since the maximum number of concurrently executing threads and the number of dedicated threads is set in Web application B, you can use the threads up to the respective set values.

     You can concurrently execute maximum of 3 threads in the Web application B. Out of the three threads, you can secure minimum of two threads for the number of dedicated threads in Web application B.
   - Size of the pending queue of the Web application

The size of the pending queue of the Web application is specified in the Web application B. If three threads are used concurrently in the Web application B, a maximum of ten requests to the Web application B are accumulated in the queue.

3. Web application C and Web application D

   The setting to control the concurrently executing threads in the Web application is not specified in Web application C and Web application D.

   Perform the operation as follows:

   - Maximum number of concurrently executing threads

     In Web application C and Web application D, use the number of shared threads of the Web container. In this case, since the number of shared threads of the Web container is seven, you can use maximum up to seven threads together in Web application C and Web application D.

     Note that the number of threads is shared between Web application C and Web application D.

     In Web application 3 and Web application 4, since the setting to control the number of concurrently executing threads is not specified, there are no dedicated threads. When the access is centralized in Web application A and Web application B, and no threads are available, the processing of Web application C and Web application D is pending.

   - Size of default pending queue

     When the processing of the Web application C or the Web application D is pending, the requests to these Web applications are accumulated in the pending queue. The size of the pending queue is not specified for the Web application C and the Web application D, and therefore, the requests are accumulated in the default pending queue. A maximum of ten requests are accumulated in the pending queue.

     Reference note

     In a Web container, the threads are even used for error processing of static contents and requests. The number of threads used for these objectives can be calculated from the following expression:

     Number-of-processing-threads-of-Web-server − (*Maximum-number-of-concurrently-executing-threads-of-Web-container* + *Total-size-of-the-pending-queues*[#])

     # The total size of the pending queues refers to the value obtained by adding the size of the pending queues of the Web container, Web application A and Web application B, in this figure.

     Therefore, in the case of this figure, the value will be `50 - (10 + (10 + 5 + 10) )` implying that 15 threads are used for error processing of static contents and requests.

2.17.7 Notes on controlling the number of concurrently executing threads in the Web application

When you control the number of concurrently executing threads in the Web application, note the following points:

- When the number of threads exceeds the maximum number of concurrently executing threads in Web container

  Since only the requests equal to the minimum number of threads secured in the Web application are implemented when all of the following conditions are satisfied, the thread count may temporarily exceed the maximum number of concurrently executing threads in the Web container.

  - If the value set for the number of concurrently executing threads in the Web container is completely used when the access is centralized.
  - If a Web application in which the number of dedicated threads is set, is deployed
  - If a request arrives in the deployed Web application, before completion of the request processing threads that are running

- Access to be controlled

  Apart from the access to servlets and JSPs, the number of concurrently executing threads is also to be controlled when the static contents are accessed.

- Notes for using the error page customization functionality

  When using the error page customization functionality with `<error-page>` tag used in `web.xml`, the number of concurrently executing threads is not controlled in processing of the request that is transferred when an error occurs. There are no restrictions as a result of controlling the number of concurrently executing threads.

- About the threads

  The dedicated threads are not generated as exclusive threads of the Web application. The minimum number of threads secured when a Web application is executed is set here for the thread count.

  Note that the threads for request processing is re-used in the entire Web container.

- Maximum number of concurrently executing threads in Web application

  Even if a value greater than the maximum number of concurrently executing threads in Web container is set as the maximum number of concurrently executing threads in Web application, you can deploy the Web application on a J2EE server. Note that the maximum number of concurrently executing threads in Web container is, however used for the maximum number of concurrently executing threads in deployed Web application.

- When the number of shared threads in Web application is 0 or less

  If you deploy a following type of Web application, the number of shared threads in Web application may become 0 or less. Note that if the number of shared threads in the Web application is 0 or less, deployment of the Web application fails. You can consider the following cases in which the number of shared threads is 0 or less:

  - When a Web application with setting to control the number of concurrently executing threads in URL group is already deployed, and an attempt is made to deploy another Web application with the number of dedicated threads set
  - When the number of dedicated threads, and the setting to control the number of concurrently executing threads in URL group is specified in a Web application to be deployed, and the number of shared threads in Web application is already 0 or less during setup

2.18 Controlling the number of concurrently executing threads in the URL group

This section describes the settings for controlling the number of concurrently executing threads in the URL group.

Note that when you control the number of threads in the URL group, you also need to simultaneously specify the setting to control the threads in the Web container and the Web application. Also see *2.16 Controlling the number of concurrently executing threads in the Web container* and *2.17 Controlling the number of concurrently executing threads in the Web application*.

The following table describes the organization of this section.

Table 2-55: Organization of this section (Controlling the number of concurrently executing threads in the URL group)

Category	Title	Reference
Description	Mechanism of controlling the number of concurrently executing threads (URL group)	*2.18.1*
	Mapping of URL patterns	*2.18.2*
	Parameters required for controlling the number of concurrently executing threads (URL group)	*2.18.3*
	Guidelines for setting the number of concurrently executing threads (URL group)	*2.18.4*
Implementation	Definition in `cosminexus.xml`	*2.18.5*
Settings	Execution environment settings (Web application settings)	*2.18.6*
	Examples of setting the number of concurrently executing threads and the size of a pending queue (URL group)	*2.18.7*

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

2.18.1 Mechanism of controlling the number of concurrently executing threads (URL Group)

If you want to control the number of concurrently executing threads more carefully than the Web application, control the number of concurrently executing threads in the URL group.

If a business logic (servlets and JavaBeans) that takes long processing time is present in a Web application, almost all the concurrently executing threads in the Web application are used, and the processing of other business logics may be in waiting state. In such a case, by specifying the setting to control the number of concurrently executing threads in the URL group, you can execute the business logics of the Web application without affecting the other processing.

Note that the URL that you can control with the number of concurrently executing threads indicates the request URI defined in RFC 2616. The URL that can control the number of concurrently executing threads in URL group does not include the query characters of request URI. An example is shown below.

- HTTP requests that can be controlled: /webapp/index.html
- HTTP requests that cannot be controlled: http: //localhost/webapp/index.html?id=0001

For guidelines on setting the number of concurrently executing threads in the URL group, see *8.3.4 Controlling the number of concurrently executing threads in a Web application* in the *uCosminexus Application Server System Design Guide*.

2.18.2 Mapping of URL patterns

This subsection explains the order of mapping between URL specified in a URL that can be controlled by controlling the number of concurrently executed threads and a URL pattern and the request URL. This subsection also explains the case in which the URL pattern is set for the welcome file.

(1) URL that can be controlled by controlling the number of concurrently executed threads

A URL that can be controlled by controlling the number of concurrently executed threads in a URL group indicates the request URI defined in RFC 2616. The URL that can be controlled by controlling the number of concurrently executed threads in a URL group does not include the schema, host, port, and query string of the request URI.

The following is an example of a URL that can be controlled by controlling the number of concurrently executed threads in a URL group:

The request URI of the received HTTP request:

```
http://localhost/webapp/index.html?id=0001
```

The part used for controlling the number of concurrently executed threads in a URL group:

```
/webapp/index.html
```

(2) Mapping order

The request URL is mapped in the following order of 1. to 3. Note that the mapping order is same as the order applicable to servlet mapping in the Servlet specifications.

1. Exact match

   If the request URL and the URL pattern match completely, the matching URL pattern is applied.

2. Prefix match

   The URL pattern in which the request URL and the prefix are matching and the request URL matches with the longest string, is applied.

3. Extension match

   If the request URL and the extension match, the matching URL pattern is applied.

Note that if the request URL does not match with the above-mentioned 1. to 3., the number of concurrently executing threads is not controlled in the URL group. In such a request URL, the number of concurrently executing threads is controlled in the Web application.

Examples of URL mapping are explained using the following URL patterns:

Table 2-56: Examples of URL pattern

URL pattern	URL group corresponding to the URL pattern
`/foo/bar`	Control1
`/foo/*`	Control2
`/foo/bar/*`	Control3
`*.do`	Control4

Mapping example 1: When the request URL is `/foo/bar`

Since the request URL completely matches with the URL pattern of Control1, the request URL is distributed to the Control1.

Mapping example 2: When the request URL is `/foo/bb`

Since the request URL does not match completely with any URL pattern, the request URL is distributed to Control2 in which the prefix is matching.

Mapping example 3: When the request URL is `/foo/aa.do`

In this case, the following locations are matching in Control2 and Control4:

- Prefix `/foo` matches with Control2.
- Extension `.do` matches with Control4.

In the mapping order, since prefix match is given priority over extension match, the request URL is distributed to Control2.

Mapping example 4: When the request URL is `/foo/bar/`

In this case, the prefix is matching in Control2 and Control3 respectively.

- Prefix `/foo` matches with Control2.
- Prefix `/foo/bar` matches with Control3.

The request URL is distributed to Control3 in which it matches with the longer string.

Mapping example 5: When the request URL is `/foo/bar/action.do`

In this case, the following locations are matching in Control2, Control3 and Control4:

- Prefix `/foo` matches with Control2.
- Prefix `/foo/bar` matches with Control3.
- Extension `.do` matches with Control4.

In the mapping order, since prefix match is given priority over extension match, and the URL pattern that matches the longer string is given priority, the request URL is distributed to Control3.

Mapping example 6: When the request URL is `/context/fo`

Since there is no corresponding URL pattern, the mapping of request URL is treated as controlling the concurrently executing threads in the Web application.

Mapping example 7: When the request URL is `/action.do`

Since the extension matches with Control4, the request URL is distributed to Control4.

Mapping example 8: When the request URL is `/boo/action.do`

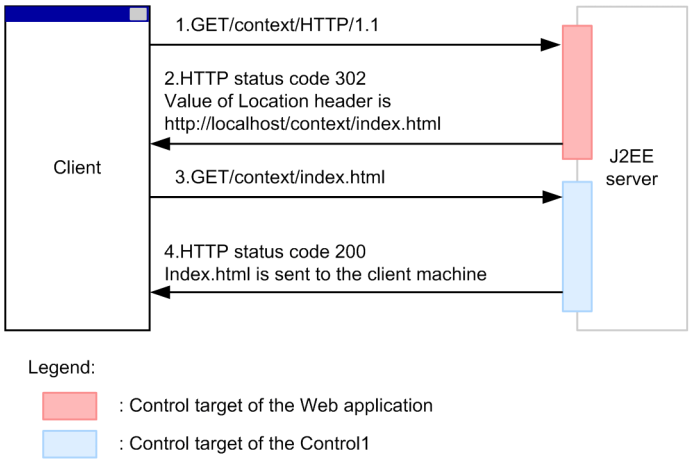
Since the extension matches with Control4, the request URL is distributed to Control4.

(3) Flow of processing when URL pattern is set in the welcome file

The following figure illustrates the flow of processing when the URL pattern is set in the welcome file. In the example illustrated in the following figure, the context root of Web application is `context`, and `/index.html` is set as the `web.xml` for welcome file. The URL pattern and the definition name for controlling the number of concurrently executing threads in the URL group is as follows:

- URL pattern: `/index.html`
- Definition name for controlling the number of concurrently executing threads in the URL group: `Control1`

Figure 2-22: Example of setting a URL pattern in the welcome file



The flow in the figure is as follows:

1. The client sends requests to `http: //localhost/context/`.
2. The J2EE server returns the HTTP status code 302, so that the client can re-access the welcome file set as `web.xml`. `http: //localhost/context/index.html` is included in Location header.
3. The client that receives the HTTP response sends requests to the value of the Location header (`http: //localhost/context/index.html`).

4. Since the request of the corresponding Web application is `/index.html`, the request is controlled by 'controlling the number of concurrently executing threads in the URL group' functionality of Control1. After request processing is complete, index.html is send to the client as response.

2.18.3 Parameters required for controlling the number of concurrently executing threads (URL group)

This subsection explains the settings to control the thread count in the URL group. You need to specify the following settings to control the thread count in the URL group:

1. Specifying the setting to control the number of concurrently executing threads in the Web container

   Set the maximum number of concurrently executing threads in the Web container. Note that the maximum number of concurrently executing threads set here is shared in all the Web applications deployed on the Web container.

   For details on specifying the setting to control the number of concurrently executing threads in the Web container, see *2.16 Controlling the number of concurrently executing threads in the Web container*.

2. Specifying the setting to control the number of concurrently executing threads in the Web application

   Set the number of concurrently executing threads and the number of dedicated threads in the Web application. Note that when setting the number of concurrently executing threads in the URL group, it is mandatory to set the number of dedicated threads in the Web application.

   For details on specifying the setting to control the number of concurrently executing threads in the Web application, see *2.17 Controlling the number of concurrently executing threads in the Web application*.

3. Specifying the setting to control the number of concurrently executing threads in the URL group

   To control the number of concurrently executing threads in the URL group, set the following parameters in the URL group of the Web application in which the setting to control the number of concurrently executing threads in Web application is specified:

   - Definition name of the control of number of concurrently executing threads in URL group

     Set the name of the URL group that is the unit to control the number of concurrently executing threads.

   - Maximum number of concurrently executing threads

     Set the maximum number of threads that you can concurrently execute in the URL group.

   - Number of dedicated threads

     Set up the number of dedicated threads in the URL group.

   - Size of the pending queue of the URL group

     Set the size of a pending queue in the URL group.

   - URL pattern

     Specify the URL pattern to distribute the request URL in which the number of threads is to be controlled.

The detailed settings to control the number of concurrently executing threads in the URL group are explained below:

(1) Maximum number of concurrently executing threads in URL group

The thread count used in the URL group refers to the thread count of the Web application to which the URL group belongs. Consequently, if one thread is used in the URL group, it implies that one thread is executed even in the Web application that includes the URL group.

Note that for a request URL in which the maximum number of concurrently executing threads of URL group is not set, the number of shared threads in the Web application is used. The number of shared threads in the Web application is as follows:

Number of shared threads in Web application unit =

Maximum number of concurrently executing threads in Web application unit[#] - Total number of dedicated threads in URL group

The smaller value in following 1. and 2., is applied as the maximum number of concurrently executing threads:

1. Number of shared threads in Web container unit
2. Value set as the maximum number of concurrently executing threads in Web application

In such a case, the number of shared threads in Web application needs to be at least 1. If the number of shared threads is 0 or less, an error occurs. For details, see *2.17.7 Notes on controlling the number of concurrently executing threads in the Web application*.

(2) Number of dedicated threads in URL group

Set the number of dedicated threads in URL group to execute specific business logic without being affected by other business logics in the Web application.

Specify the number of dedicated threads in URL group within the range of the number of dedicated threads set in the Web application. As a result, if the number of dedicated threads is not set in the Web application that includes the URL group, you cannot set the number of dedicated threads even in the URL group.

If the number of requests to the request URL in which the setting to control the number of concurrently executing threads in the URL group is specified exceeds the number of dedicated threads in URL group, and if the value is less than the maximum number of concurrently executing threads in the URL group, the requests are processed by using the number of shared threads in the Web application. As a result, if the number of shared threads in the Web application is less, be careful that the maximum number of concurrently executing threads in URL group will be even lesser than the set value.

(3) Pending queue in URL group

The pending queue in the URL group refers to the queue in which requests enter when the number of concurrently executing threads in URL group reaches the upper limit. The pending queue in URL group is created for each URL group when the setting to control the number of concurrently executing threads in the URL group is specified. Set the size of this pending queue.

Requests enter a pending queue when the number of concurrently executing threads in a URL group reaches the upper limit, and the pending queue of the URL group has some space. The requests in the pending queue of the URL group are extracted sequentially from the pending queue, and executed after the processing of the request is complete. If the number of concurrently executing threads in the URL group reaches the upper limit and there is no space in the pending queue of the URL group, an error occurs, and then HTTP status code 503 is returned to the client.

Note that the requests entering the pending queue of the URL group do not enter the default pending queue and the pending queue of the Web application. The pending queue of the Web application is used for the requests that do not correspond to the request URL set for controlling the number of concurrently executing threads in the URL group.

(4) Setting the URL patterns

Set the URL patterns for distributing the request URL. You can specify the URL patterns for the servlet mapping in Servlet specifications. You can specify the following URL patterns:

- A string beginning with "/"
  Example: `/index.jsp`
- A string beginning with "/" and ending with "/*"
  Example: `/test/*`
- A string beginning with "*."
  Example: `*.do`

For details on the order of mapping the request URL with the URL pattern, see *2.18.2 Mapping of URL patterns*.

To control the number of concurrently executing threads in the URL group, you must simultaneously specify settings to control the number of concurrently executing threads in the Web applications and the settings to control the number of threads in the Web Container. The following table lists the parameters specified for controlling the number of concurrently executing threads in the URL group:

Table 2-57: Parameters specified for controlling the number of threads in the URL group

Set parameters	Set units		
	Web Container	Web application	URL group
Whether to control the number of concurrently executing threads	--	Y	--
Maximum number of concurrently executing threads	Y	Y	Y
Number of dedicated threads	--	Y	Y
Size of a pending queue	--	Y	Y
Default size of a pending queue	--	Y	--
Definition name of the control of number of concurrently executing threads in URL group	--	--	Y
URL pattern to be controlled	--	--	Y

Legend:
Y: Specified
--: Not applicable

The settings for controlling the number of concurrently executing threads in the URL group are described here. Use the server management commands to specify settings for controlling the number of concurrently executing threads in the URL group. For details on the parameters set in the Web Container, see *2.16.2 Execution environment settings (J2EE server settings)* and for details on the settings for each Web application parameter, see *2.17.5 Execution environment settings*.

You can set the followings in the server management commands:

- **Definition name of the control of number of concurrently executing threads in URL group**
  Specifies the URL group name in which the number of concurrently executing threads is controlled. The group name must be unique in the Web application.
  The characters that can be used are as follows:
  - Alphanumeric characters (A-Z, a-z, 0-9)
  - One-byte hyphen (-)
  - One-byte underscore (_)

  The length of the string is 1 to 64.
- **Maximum number of concurrently executing threads in each URL group**
  Specify the maximum number of threads that you can concurrently execute in a URL group. The specifiable range is as follows:
  Setup range for Maximum number of concurrently executing threads
  `1` ≤ *Maximum-number-of-concurrently-executing-threads (URL-group)* ≤ *Maximum-number-of-concurrently-executing-threads (Web-application)*
- **Number of dedicated threads in the URL group**
  Specify the minimum number of threads to be secured in the URL group (number of dedicated threads). The specifiable range is as follows:
  Setup range for the number of dedicated threads
  `0` ≤ *number-of-dedicated-threads (URL-group)* ≤ *Maximum-number-of-concurrently-executing-threads (URL-group)*
  Furthermore, the number of dedicated threads in the URL group must be less than the number of dedicated threads in the Web application.

Also, the sum of the number of dedicated threads set in the URL group in the Web application must fulfill the following conditions. Note that these conditions depend on the values of the maximum number of concurrently executing threads and the number of dedicated threads in Web application.

In the Web application settings, when *Maximum-number-of-concurrently-executing-threads* ≠ *number-of-dedicated-threads*

Number-of-dedicated-threads (Web-application) ≥ *Sum-of-number-of-dedicated-threads (URL-group)*

In the Web application settings, when *Maximum-number-of-concurrently-executing-threads = number-of-dedicated-threads*

Number-of-dedicated-threads (Web-application) > *Sum-of-number-of-dedicated-threads (URL-group)*

Specify `0` when you do not want to set the number of dedicated threads.

- **Size of the pending queue of the URL group**

  When the number of executing threads reaches the maximum number, the request is accumulated in the queue. Specify the pending queue size at this time for each URL group. The specifiable range is as follows:

  Setup range for the size of the pending queue of the URL group

  `0` ≤ *pending-queue-size (URL-group)* ≤ `2,147,483,647`

  If you specify `0`, the pending queue of the URL group will not be used. At this time, if the number of concurrently executing threads reaches the upper limit, the requests will result in an error.

- **URL pattern to be controlled**

  Specify the URL for which you want to control the number of concurrently executing threads. Specify a unique name in the Web application for the URL pattern. Also, specify the URL below the Web application context.

When using the server management commands, set the number of concurrently executing threads in the `<urlgroup-thread-control>` tag under the `<thread-control>` tag of the WAR property file.

- Specify the definition name for controlling the number of concurrently executing threads in the `<urlgroup-thread-control-name>` tag.
- Specify the maximum number of concurrently executing threads in the URL group in the `<urlgroup-thread-control-max-threads>` tag.
- Specify the number of dedicated threads in the `<urlgroup-thread-control-exclusive-threads>` tag.
- Specify the size of the pending queue of the URL group in the `<urlgroup-thread-control-queue-size>` tag.
- In the `<urlgroup-thread-control-mapping>` tag, specify the URL pattern you want to control enclosed in the `<url-pattern>` tag.

Acquire the property file using the `cjgetappprop` command of server management commands, and after editing the property file, apply the edited contents using the `cjsetappprop` command. For the server management commands, see *3. Basic Operation of Server Management Commands* in the *uCosminexus Application Server Application Setup Guide*.

2.18.4 Guidelines for setting the number of concurrently executing threads (URL group)

This subsection provides guidelines for specifying the settings to control the number of concurrently executing threads in the URL group.

- When specifying the settings to control the number of concurrently executing threads in the URL group, set up the followings beforehand:
  - Maximum number of concurrently executing threads of Web container
  - Maximum number of concurrently executing threads, number of dedicated threads, pending queue size for each Web application
- Follow this order to determine the value you want to set:

1. Set the maximum number of concurrently executing threads in the URL group.
2. Set the number of dedicated threads in the URL group.
3. Set the size of a pending queue in the URL group.
4. Set the URL pattern.

- In the maximum number of concurrently executing threads in the URL group, set an appropriate value for the business logic present in the J2EE application.

  For example, if the processing of specific business logic is heavy, by setting an upper limit to the number of concurrently executing threads of that business logic, you can prevent the business logic from occupying the entire Web application capacity, even if the access to the business logic is centralized.
- Set the number of dedicated threads in URL group to execute specific business logic without being affected by other business logics in the Web application.
- Set the pending queue size in the URL group to set an upper limit for the flow to specific business logic. The guidelines for setting the pending queue in the URL group is the same as that for the default pending queue and the pending queue in the Web application.

2.18.5 Definition in cosminexus.xml

This subsection describes the definitions in `cosminexus.xml` required in the application development environment.

Specify the definition for controlling the number of concurrently executing threads in the URL group in the `<war>` tag of `cosminexus.xml`.

The following table lists the definitions in `cosminexus.xml` for controlling the number of concurrently executing threads in the URL group.

Table 2-58: Definitions in cosminexus.xml for controlling the number of concurrently executing threads in the URL group

Tag to be specified	Setting contents
`<urlgroup-thread-control-name>` tag	Specifies the definition name for controlling the number of concurrently executing threads.
`<urlgroup-thread-control-max-threads>` tag	Specifies the Maximum number of concurrently executing threads in the URL group.
`<urlgroup-thread-control-exclusive-threads>` tag	Specifies the number of dedicated threads.
`<urlgroup-thread-control-queue-size>` tag	Specifies the size of the pending queue of the URL group.
`<urlgroup-thread-control-mapping>` tag	Specifies the URL pattern you want to control enclosed in the `<url-pattern>` tag.

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*.

2.18.6 Execution environment settings (Web application settings)

To control the number of concurrently executing threads in the URL group, you must set up the Web application.

Reference the Web application settings only when you want to set or change the properties of the Web applications that do not contain `cosminexus.xml`.

Implement the Web application settings in the execution environment by using the server management commands and property files. Use the WAR property file for the definition for controlling the number of concurrently executing threads in the URL group.

The tags specified in the WAR property file correspond to `cosminexus.xml`. For details on the definitions in `cosminexus.xml`, see *2.18.5 Definition in cosminexus.xml*.

2.18.7 Example of setting the number of concurrently executing threads and the size of a pending queue (URL Group)

This subsection describes the examples of setting up the number of concurrently executing threads and the size of the pending queue of the URL group. The examples explained here reference to the Web applications in which the Web server integration functionality is used.

(1) Example of setting the Web applications used in the description

In this example, out of the two Web applications deployed on the Web container, the setting to control the number of concurrently executing threads in the Web application as well as in the URL group is specified in one Web application. The settings are shown below:

- Maximum number of concurrent connections in the Web server: 40
- Maximum number of concurrently executing threads of Web container: 8
- Default size of the pending queue: 5
- Specifying the setting to control the number of concurrently executing threads in the Web application
  Set the following contents in Web application A. Note that in Web application B, the setting is not specified to control the number of concurrently executing threads in the Web application.

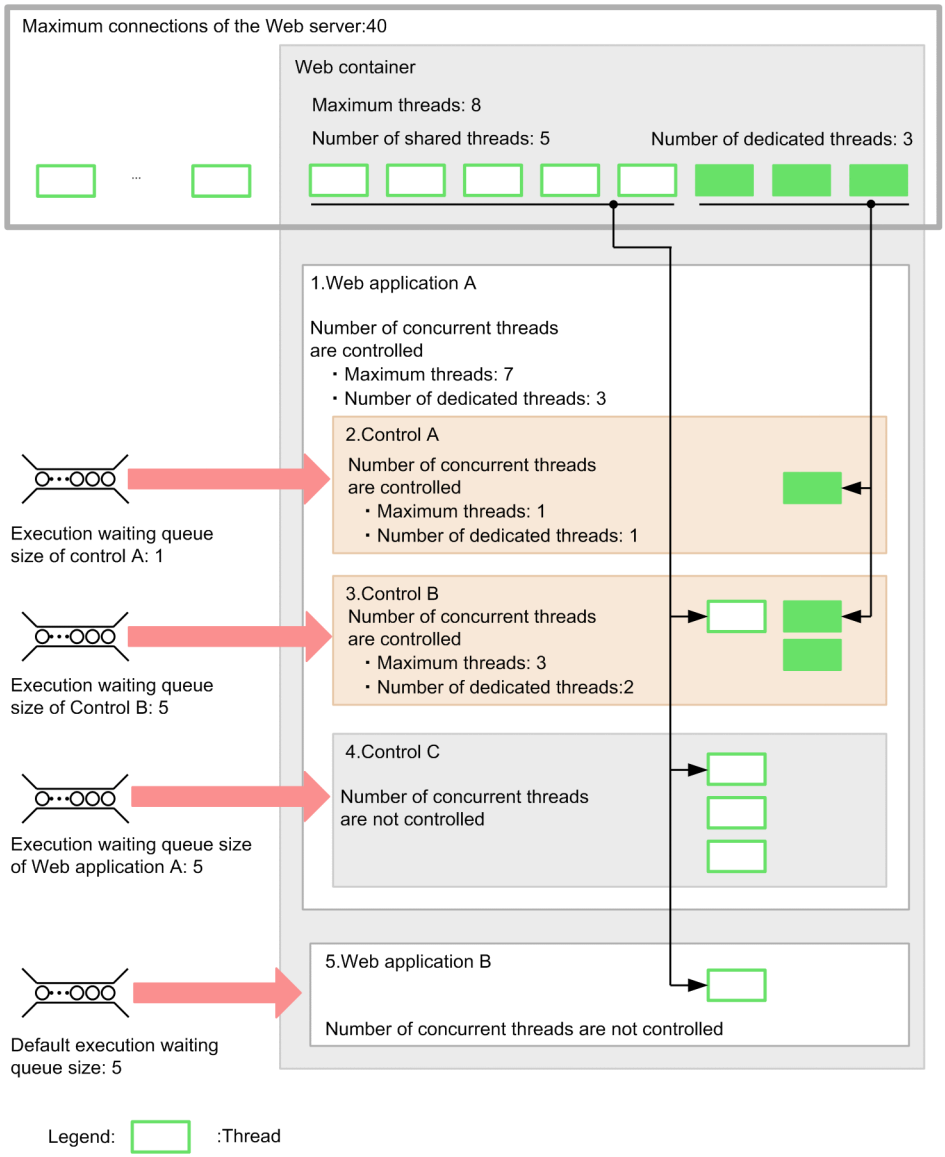
Web application name	Maximum number of concurrently executing threads	Number of dedicated threads	Size of pending queue of Web application
Web application A	7	3	5

- Specifying the setting to control the number of concurrently executing threads in the URL group
  In Web application A, the setting of the following URL groups is specified. In Control C, the settings for controlling the number of concurrently executing threads in URL group are not specified.

URL group name	URL pattern	Maximum number of concurrently executing threads	Number of dedicated threads	Size of pending queue of Web application
Control A	`/health_check.jsp`	1	1	1
Control B	`/create_pdf`	3	2	5

The following figure shows the example in which the thread count is controlled in the URL group:

Figure 2-23: Example of settings in the URL group



(2) Number of threads available in each Web application

The maximum number of concurrently executing threads, the number of dedicated threads, and the size of the pending queue that you can use in the case of the setup in figure 2-23 are illustrated below for each Web application or URL group. Note that the serial number of the description corresponds to the serial number mentioned in the figure.

1. Web application A

   The setting to control the number of concurrently executing threads is specified in the Web application A. In the business logics (Control A and Control B) of Web application A, the setting to control the number of concurrently executing threads in the URL group is specified.

   The thread count of the Web application A is explained below:

   - Maximum number of concurrently executing threads and the number of dedicated threads

     Since the maximum number of concurrently executing threads and the number of dedicated threads is set, you can use the threads up to the respective set values.

     You can concurrently execute maximum of 7 threads in the Web application A. Out of the seven threads you can secure minimum of three thread for the number of dedicated threads in Web application A.

   - Number of shared threads

Calculate the number of shared threads that you can use in the entire Web application A by the following expression: Maximum number of concurrently executing threads in Web application A - Total number of dedicated threads. In this case, the number of shared threads will be 7 - 3, that is 4.

- Size of the pending queue of the Web application

  The size of the pending queue of the Web application is specified in the Web application A. If seven threads are used concurrently in the entire Web application A, a maximum of five requests are accumulated in the pending queue. Note that this queue is used in the requests to the business logic Control C in the Web application, in which the settings to control the number of concurrently executing threads in the URL group is not specified.

2. Control A (requests to `/health_check.jsp`)
   - Maximum number of concurrently executing threads and the number of dedicated threads

     Since the maximum number of concurrently executing threads and the number of dedicated threads is set in Control A, you can use the threads up to the respective set values.

     In Control A, the maximum number of concurrently executable threads is one. This one thread also acts as the minimum number of dedicated threads that you can secure in Control A. Note that the number of dedicated threads in Control A is one of the dedicated threads in Web application A.
   - Size of the pending queue of the URL group

     The size of the pending queue of the URL group is specified in Control A. If one thread is in use in Control A, maximum one request is accumulated in the pending queue of the URL group.
3. Control B (requests to `/create_pdf`)
   - Maximum number of concurrently executing threads and the number of dedicated threads

     Since the maximum number of concurrently executing threads and the number of dedicated threads is set in Control B, you can use the threads up to the respective set values.

     In Control B, the maximum number of concurrently executable threads is three. Out of the three threads, you can secure minimum of two threads as the dedicated threads in Control B. Note that, , out of the dedicated threads in Web application 1, two threads serve as the dedicated threads in Control2.
   - Size of the pending queue of the URL group

     The size of the pending queue of the URL group is specified in Control B. If three threads are in use in Control B, maximum five requests are accumulated in the pending queue of the URL group.
4. Control C process

   The setting to control the number of concurrently executing threads is not specified for requests to business logic Control C in Web application A.

   Perform the operation as follows:
   - Maximum number of concurrently executing threads

     The number of shared threads in Web application A is the maximum number of concurrently executing threads. Since the number of shared threads in Web application A is four, the maximum number of concurrently executing threads of the Control C processing is four.

     Web application 1 does not contain any dedicated threads since the setting to control the number of concurrently executing threads is not specified. Therefore, if access to Control A or B is centralized and if the threads that can be used in the Web application A are lost, the processing of the Control C business logic remains pending.
   - Size of the pending queue of the Web application

     When the processing of business logic Control C is pending, the requests to the processing are accumulated in the queue. Since the size of the pending queue of the URL group is not specified in the business logic Control C, the requests are accumulated in the pending queue of the Web application A. A maximum of five requests are accumulated in the pending queue.
5. Web application B

   The settings to control the number of concurrently executing threads in Web application are not specified in Web application B.

   Perform the operation as follows:
   - Maximum number of concurrently executing threads

     The number of shared threads in Web container is used as the maximum number of concurrently executing threads. You can calculate the number of shared threads in Web container by the following expression:

Maximum number of concurrently executing threads in Web container - Number of dedicated threads in Web application A

In this case, it will be 8 - 3, that is the maximum number of concurrently executing threads in Web application B will be 5.

- Size of default pending queue

  When requests are pending in the Web application B, the requests to the Web application B are accumulated in the pending queue. Since the size of the pending queue of the Web application is not specified in the Web application B, the requests are accumulated in the default pending queue. A maximum of five requests are accumulated in the pending queue.

Reference note

In a Web container, the threads are even used for error processing of static contents and requests. The number of threads used for these objectives can be calculated from the following expression:

Number-of-processing-threads-of-Web-server − (*Maximum-number-of-concurrently-executing-threads-of-Web-container* + *Total-size-of-the-pending-queues*[#])

The total size of the pending queues refers to the value obtained by adding the size of the pending queues of the Web container, Web application A, Control A, and Control B, in this figure.

Therefore, in case of figure 3-16, the value will be `40 - (8 + (5 + 5 + 1 + 5))`, that is the number of threads used for error processing of static contents and requests is 16.

2.19 Dynamic change in the number of concurrently executing threads

This section describes the settings for dynamically changing the number of concurrently executing threads.

The following table describes the organization of this section.

Table 2-59: Organization of this section (Dynamically changing the number of concurrently executing threads)

Category	Title	Reference
Description	Overview of dynamically changing the number of concurrently executing threads	*2.19.1*
	Flow of dynamically changing the number of concurrently executing threads	*2.19.2*
	Operations of a Web application when the number of concurrently executing threads are changed dynamically	*2.19.3*
Notes	Precautions related to dynamically changing the number of concurrently executing threads	*2.19.4*

Note:

There is no specific description of *Implementation*, *Settings*, and *Operations* for this functionality.

2.19.1 Overview of dynamically changing the number of concurrently executing threads

This subsection provides an overview of dynamically changing the number of concurrently executing threads.

(1) Usage of dynamically changing the number of concurrently executing threads

In a system built with Application Server, you can dynamically change the maximum number of concurrently executing threads, number of dedicated threads, and pending queue size of each Web application without stopping the service. You can change the maximum number of concurrently executing threads, number of dedicated threads, and pending queue size of each Web application with the management command (`mngsvrutil`) .

If the maximum number of concurrently executing threads of each Web application is changed, the following aspects can be supported:

- Performance tuning in the operational status of each Web application

  You can tune the performance while providing services to a client.
- Changing the maximum number of concurrently executing threads of each temporary Web application corresponding to the access status

  You can temporarily increase or decrease the maximum number of concurrently executing threads of a specific Web application corresponding to the access status.
- Changing the maximum number of concurrently executing threads of each planned Web application according to a time zone

  You can systematically increase or decrease the maximum number of concurrently executing threads of a Web application according to a time zone.

Note that the items set here become invalid when you stop the service and the settings are not saved in J2EE server. Further, with this method you can change only the information related to a Web application. When you change the Web container settings, you have to restart the J2EE server to apply the changes.

When you want to change the maximum number of concurrently executing threads of each Web container or when you want to permanently change the maximum number of concurrently executing threads of a Web application, follow the procedure similar to the one used when building a system.

Important note

When the maximum number of concurrently executing threads of a Web application is changed, the control operation for these threads of each URL group is affected depending on the relation between the maximum number of concurrently executing threads and the number of dedicated threads of each URL group. For details, see *2.19.1(4) Effect on the controlling of the number of concurrently executing threads in the URL group*.

(2) An example of setting change

This subsection introduces an example of setting change for a specific Web application, in the case you want to improve the throughput and reduce the number of requests resulting in an error. In this example, the maximum number of concurrently executing threads, the number of dedicated threads, and the size of the pending queue of a Web application are increased. Note that you cannot change the maximum number of concurrently executing threads of Web container, and the maximum number of concurrently executing threads of URL group during dynamic change in the number of concurrently executing threads.

Table 2-60: An example of the dynamic change in the number of concurrently executing threads

Parameter		Setting before change	Setting after change
Maximum number of concurrently executing threads of Web container		10	--
Web application setup	Maximum number of concurrently executing threads	7	8
	Number of dedicated threads	4	5
	Size of a pending queue	8	10

Legend:
--: The setting cannot be changed

(3) Operation after setting change

A change in the number of concurrently executing threads is applied immediately. You need to pay attention in the following operations, immediately after making changes:

When the maximum number of concurrently executing threads is changed

- When the maximum number of concurrently executing threads is increased

  The requests that change to executable state from among the pending requests of the Web application are executed immediately.
- When the maximum number of concurrently executing threads is reduced, since all the threads specified in the maximum number of concurrently executing threads are used.

  The requests that exceed the maximum number of concurrently executing threads after change are executed concurrently on a temporary basis.

When the number of dedicated threads is changed

- When the number of dedicated threads is increased, since all the threads specified in the maximum number of concurrently executing threads of Web container are used

  If there are pending requests in a Web application in which the number of dedicated threads is increased, as many requests as the number of dedicated threads are executed immediately. In such cases, however, the requests that exceed the maximum number of concurrently executing threads of Web container are executed concurrently on a temporary basis.
- When the number of dedicated threads is reduced

  When the number of dedicated threads is reduced, the number of threads shared in all the Web applications increases. In such cases, the requests in the pending state that change to executable state by increasing the number of threads shared in all Web applications, are executed immediately.

When the size of the pending queue of the Web application is changed

- When the size of a pending queue is reduced, since requests are pending up to the upper limit of the queue in a pending queue of Web application

Error HTTP 503 is returned for the requests that exceed the size of the pending queue.

(4) Effect on the controlling of the number of concurrently executing threads in the URL group

When you dynamically change the number of concurrently executing threads of the Web application in which the settings to control the number of concurrently executing threads in URL group is specified, the setting of the number of concurrently executing threads in URL group may be affected. The setting is affected due to the following changes:

When the maximum number of concurrently executing threads in Web application is reduced

When the following condition is satisfied by reducing the maximum number of concurrently executing threads in the Web application, the maximum number of concurrently executing threads in URL group is temporarily used as the number of concurrently executing threads in Web application.

Condition in which the maximum number of concurrently executing threads in URL group is changed

Maximum-number-of-concurrently-executing-threads-in-URL-group > *Maximum-number-of-concurrently-executing-threads-in-Web-application*

The setting of the maximum number of concurrently executing threads in URL group is however not changed. If the maximum number of concurrently executing threads in Web application reduces up to the number of threads set during dynamic change, and the maximum number of concurrently executing threads in URL group falls below the maximum number of concurrently executing threads in Web application, the maximum number of concurrently executing threads in URL group operates as per the setting.

Note that due to of this change, in order to continue the processing of running requests, the requests that exceed the maximum number of concurrently executing threads of Web application after change are executed concurrently on a temporary basis.

When the number of dedicated threads in Web application is reduced

If the number of dedicated threads in Web application is reduced, the number of dedicated threads set in all the URL groups of the Web application is no longer available. The conditions in which you cannot use the number of dedicated threads are explained below. Note that these conditions depend on the relationship between the maximum number of concurrently executing threads and the number of dedicated threads in Web application.

When the maximum number of concurrently executing threads in Web application = Number of dedicated threads in Web application

When the following expression is fulfilled, you cannot use the number of dedicated threads set in the URL group.

Number-of-dedicated-threads-in-Web-application ≤ *Total-number-of-dedicated-threads-in-URL-group*

When the maximum number of concurrently executing threads in Web application ≠ Number of dedicated threads in Web application

When the following expression is fulfilled, you cannot use the number of dedicated threads set in the URL group.

Number-of-dedicated-threads-in-Web-application < *Total-number-of-dedicated-threads-in-URL-group*

2.19.2 Flow of dynamically changing the number of concurrently executing threads

The preparations and procedures for dynamically changing the maximum number of concurrently executing threads of a Web application are described below:

Preparation

Perform the dynamic change in the maximum number of concurrently executing threads of a Web application when the J2EE applications including the J2EE server and the Web application, are started.

For starting a J2EE server, see *4.1.24 Starting the system (when using CUI)* in the *uCosminexus Application Server System Setup and Operation Guide*. For starting a system including the startup of a J2EE application, see *4.1.29 Setting and starting the business application (when using CUI)* in the *uCosminexus Application Server System Setup and Operation Guide*.

Procedure

For dynamically changing the maximum number of concurrently executing threads of a Web application:

1. **Monitor the operational status of a Web application, and confirm that it is necessary to change the maximum number of concurrently executing threads (see *(1)*)**
   Perform this task using the management command.
2. **Change the maximum number of concurrently executing threads of the Web application when deemed necessary (see *(2)*)**
   Perform this task using the management command.
3. **Check the operational status of the Web application and confirm the improvement (see *(3)*)**
   Perform this task using the management command.

(1) Confirming the operational status of a Web application

Confirm the operational status of a running Web application. You can confirm the operational status of a Web application using the management command (`mngsvrutil`) . After confirming the operational status, consider whether to change the maximum number of concurrently executing threads in cases such as described below:

- When the number of pending threads is too large in number as compared to the number of active threads in a situation when this condition is not assumed
- When the current value of the number of pending requests of a Web application is approaching the maximum value of the number of pending requests of the Web application, in the case when this condition is not assumed
- When requests are overflowing from the pending queue of the Web application, in a situation when this condition is not assumed

  When you monitor and determine that it is necessary to change the maximum number of concurrently executing threads permanently, rather than changing dynamically stop the Web application and reset the maximum number of concurrently executing threads. For details on the settings for controlling the maximum number of concurrently executing threads in the Web container when the Web application is not running, see *2.16 Controlling the number of concurrently executing threads in the Web Container*, *2.17 Controlling the number of concurrently executing threads in the Web application*, and *2.18 Controlling the number of concurrently executing threads in the URL group*.

To check the operational status of the Web application, specify and execute subcommand `get` in the `mngsvrutil` command.

The execution format and an example are described below. For details on the `mngsvrutil` command, see *mngsvrutil (Management command of Management Server)* in the *uCosminexus Application Server Command Reference Guide*.

Execution format

```
mngsvrutil -m Management-Server-host-name [:port-number] -u Management-user-ID -p
Management-password -t host-name -k host get webApps
```

Execution example

```
mngsvrutil -m mnghost -u user01 -p pw1 -t host01 -k host get webApps
```

The execution result of the command is output in a standard output or file.

From the statistics of a running Web application, you can check the information to be referred when changing the maximum number of concurrently executing threads of a Web application with the following header information items. Note that N seconds is the sampling time that is set by management.

Table 2-61: Information to be referred when changing the maximum number of concurrently executing threads of a Web application

Header information	Contents
contextRoot	Context root of a Web application
exclusiveThreadCountUpperBound	Number of dedicated threads of a Web application
activeThreadCountUpperBound	Maximum number of concurrently executing threads of a Web application
waitingRequestCountUpperBound	Pending queue size of a Web application

Header information	Contents
currentThreadCountUpperBound	Upper value of the number of threads of a Web application that can be executed concurrently
activeThreadCount	Current value of the number of active threads
activeThreadCountPeak	N seconds peak of the number of active threads
activeThreadCountAverage	N seconds average value of the number of active threads
activeThreadCountHightWaterMark	Maximum value of the number of active threads
activeThreadCountLowWaterMark	Minimum value of the number of active threads
waitingRequestCount	Current value of the number of pending requests of a Web application
waitingRequestCountPeak	N seconds peak of the number of pending requests of a Web application
waitingRequestCountAverage	N seconds average value of the number of pending requests of a Web application
waitingRequestCountHighWaterMark	Maximum value of the number of pending requests of a Web application
waitingRequestCountLowWaterMark	Minimum value of the number of pending requests of a Web application
overflowRequestCount	Number of requests overflowing from the pending queue of a Web application

(2) Changing the settings of the maximum number of concurrently executing threads of a Web application

Change the following items of a Web application, the operational status of which is confirmed, as required:

- Maximum number of concurrently executing threads
- Number of dedicated threads
- Size of a pending queue

You can change these items with management command (`mngsvrutil`) . The value set here is applied until the Web application stops.

! Important note

Do not deploy and undeploy a J2EE application when dynamically changing (when the sub command change is specified in the `mngsvrutil` command) the maximum number of concurrently executing threads of a Web application.

For dynamically changing the maximum number of concurrently executing threads of a Web application, specify and execute the sub command change in the `mngsvrutil` command.

The following is the execution format. For details on the `mngsvrutil` command, see *mngsvrutil (Management command of Management Server)* in the *uCosminexus Application Server Command Reference Guide*.

Execution format

```
mngsvrutil -m Host-name-of-Management-Server [:port-number] -u Management-user-ID -
p Management-password -t host-name -k host change webAppThreadCtrl Context-root-of-
the-Web-application Maximum-number-of-concurrently-executing-threads, number-of-
dedicated-threads, pending-queue-size-of-Web-application
```

The following is an execution example. In this example, the settings are changed as shown in the following table. Note that the name of the Web application is WebAP1.

Table 2-62: Example of settings for dynamically changing the maximum number of concurrently executing threads of a Web application (WebAP1)

Setting target	Settings	Setting before change	Setting after change
Web container	Maximum number of concurrently executing threads	10	10 (cannot be changed)

Setting target	Settings	Setting before change	Setting after change
Web application (WebAP1)	Maximum number of concurrently executing threads	7	8
	Number of dedicated threads	4	5
	Size of a pending queue	8	10

Execution example

```
mngsvrutil -m mnghost -u user01 -p pw1 -t host01 -k host change
webAppThreadCtrl "WebAP1" 8, 5, 10
```

The set contents are applied immediately after executing the command.

2.19.3 Operations of a Web application when the number of concurrently executing threads are changed dynamically

This subsection describes the operations of a Web application when the maximum number of concurrently executing threads of a Web application is changed dynamically.

(1) Operations when the maximum number of concurrently executing threads is changed

When the maximum number of concurrently executing threads is changed, the Web application runs as follows:

When the maximum number of concurrently executing threads is increased

Among the pending requests of the Web application, the executable requests are executed immediately.

For example, when the number of maximum concurrently executing threads is changed from 7 to 8 due to the changes in the settings, if there are any requests in the pending queue of the Web application, one request from the pending queue is immediately executed.

When the maximum number of concurrently executing threads is decreased

The maximum number of concurrently executable threads is decreased.

However, if you attempt to change the settings when all the threads of the maximum number of concurrently executing threads are being used, the threads exceeding the maximum number of concurrently executing threads are executed temporarily, since the number of running threads cannot be decreased.

For example, when the maximum number of concurrently executing threads is changed from 8 to 7 by changing the settings, and if 8 threads are in use at that time, temporarily the 8th thread that exceeds the maximum number of concurrently executing threads after change in settings will be executed.

If one of the active threads ends, the number of threads is decreased and thereafter threads are executed according to the set value, i.e. maximum 7 threads are executed concurrently as per the settings.

(2) Operations when the number of dedicated threads is changed

When the number of dedicated threads is changed, a Web application runs as follows:

When the number of dedicated threads is increased

Among the pending requests of the Web application, the executable requests are executed immediately with the increase in the number of dedicated threads of the corresponding Web application.

Moreover, when access is in peak and all the threads set in the maximum number of concurrently executing threads of each Web container are in use, and if the number of dedicated threads of a specific Web application is increased, the requests that are in the pending queue of that Web application are executed immediately. As a result, the number of threads exceeding the maximum number of concurrently executing threads of each Web container is executed temporarily.

For example, if the maximum number of concurrently executing threads of each Web container is 10, and the number of threads used in WebAP1 and WebAP2 of the Web application is 7 and 3 respectively, if the number of dedicated threads of WebAP1 is changed to 8, 11 threads are executed temporarily in each Web container.

When the number of dedicated threads is decreased

On decreasing the number of dedicated threads of a specific Web application, the number of shared threads in each Web container is increased. Among the requests of the Web application, URL group, and default pending queue, if there are any threads that can be executed, they are executed immediately on increasing the number of shared threads from the requests in the pending queue.

(3) Operations when the size of the pending queue of a Web application is changed

When the size of the pending queue of a Web application is changed, the Web application runs as follows:

When the size of the pending queue of the Web application is increased

The size of the pending queue of the Web application is increased immediately.

When the size of the pending queue of the Web application is decreased

When the size of the pending queue of the Web application is changed to a value smaller than the number of requests waiting in the pending queue of the Web application, the request exceeding the pending queue size is returned as an HTTP 503 error.

2.19.4 Precautions related to dynamically changing the number of concurrently executing threads

- You can dynamically change the setting of the number of concurrently executing threads in Web application. You cannot dynamically change the number of concurrently executing threads in Web container, and the number of concurrently executing threads in URL group.
- The information about the dynamically changed number of concurrently executing threads is not saved in the J2EE server. Note that the changed values are invalid when the service is stopped.
- If the number of shared threads in Web application is 0 or less due to dynamic change of the number of concurrently executing threads in the Web application, you cannot use the number of dedicated threads set in all the URL groups of the Web application.

2.20 Error page customization

When the client accesses a non-existent resource, and a servlet in which an exception occurred, the Web container returns an error status code. An error page corresponding to the error status code returned from the Web container is displayed in the client. In an application server, instead of the error pages displayed in the client, pages created by the user can be displayed in the client. This is called *error page customization*.

The methods to customize the error pages include: Customization with the `<error-page>` tag of `web.xml` specified in Servlet specifications and customization with the Web server functionality. You can also use the error page customization with the in-process HTTP server.

For details on the error page customization when the Web server functionality is used, see *4.8 Error page customization with the Web server integration functionality*. For details on error page customization with the in-process HTTP server, see *5.15 Error page customization (In-process HTTP server)*.

2.21 Caching the static contents

You can cache the static contents that were accessed once, on the memory. By caching the once accessed static contents on the memory, and by returning a response from the cache to the browser when the static contents are accessed for second time and then onwards, you can shorten the response time of the static contents.

With a Web container, you can control the cache of the static contents by setting an upper limit for the memory size used for the cache in the Web application, and an upper limit for the file size of the static contents to be cached.

This section describes the static contents cache.

The following table describes the organization of this section.

Table 2-63: Organization of this section (static contents cache)

Category	Title	Reference
Description	Controlling the cache of static contents	*2.21.1*
Implementation	Definition in the DD (Settings for each Web application)	*2.21.2*
Settings	Execution environment settings	*2.21.3*

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

2.21.1 Controlling the cache of static contents

With a Web container, you can control the cache of the static contents by setting an upper limit for the memory size used for the cache in the Web application, and an upper limit for the file size of the static contents to be cached.

The cache of the static contents is controlled by the following two methods:

- **Controlling the cache of static contents in the Web container**
  This is the method to control the cache of static contents in the Web container. In the Web container, set an upper-limit value for the memory size to be cached in the Web application, and also an upper-limit value for the file size of the static contents allowed to be cached. The set upper-limit value of the memory size used in the cache of Web application unit, and the upper-limit value of the file size of static contents is applied to all the Web applications deployed on the Web container.
- **Controlling the cache of static contents in the Web application**
  This is the method to control the cache of static contents in the Web application. In the Web application, set an upper-limit value for the memory size to be cached, and an upper-limit value for the file size allowed to be cached.
  If control in Web application as well as control in Web container are set, the setting for control in Web application is given priority.

Note that if the memory size to be cached in the Web application unit exceeds the upper-limit value, or the file size of static contents exceeds the upper-limit value, caching is not performed on the memory, but a response is returned from the file system to the browser each time.

Specify the settings for caching the static contents at the following locations, for each range:

- Controlling the cache of static contents in the Web container
  Specify as a property of the J2EE server.
- Controlling the cache of static contents in the Web application
  Set as an attribute (property) of Web application.

! Important note

The functionality for caching the static contents is disabled, when the reload functionality of J2EE application is enabled.

2.21.2 Definition in the DD (Settings for each Web application)

Specify whether you want to use the static contents cache functionality, memory size of static contents permitting cache, and upper limit for the file size for each Web application.

This subsection describes the definitions in the DD required in the application development environment.

Specify the definition of static contents cache in each Web application in the <param-name> tag that exists in the <web-app><context-param> tag of web.xml.

The following table lists the definitions of the static contents cache in the DD:

Table 2-64: Definitions of the static contents cache in the DD

Items	Parameters specified in <param-name> tag	Set contents in <param-value> tag
Enabling or disabling the static contents cache functionality	com.hitachi.software.web.static_content.cache.enabled	Specifies the enabling or disabling of the static contents cache functionality.
Memory size in each Web application	com.hitachi.software.web.static_content.cache.size	Specifies the size that can be cached in the memory in bytes if the static contents cache functionality is enabled. • If the total size of the cache exceeds the specified value in the Web application, the Web application that has not been accessed for the longest time is deleted from the cache. The deletion of the cache is repeated until the total cache size becomes less than the set value. • In Web applications where the memory size is not specified, the value specified in its property is used. However, in Web applications where the memory size is specified, the value specified in its property is not used. • Specifies an integer value from 0 to 2147483647. If 0 is specified, restrictions are not set for the cacheable memory size for each Web application. • If an invalid value is set in this property and if the value is smaller than the value specified in the file size that permits cache, the default value is used. • If a null character string or blank character is set in this property, the default value is used.
File size permitting cache	com.hitachi.software.web.static_content.cache.filesize.threshold	Specifies the cacheable file size in bytes if the static contents cache functionality is enabled. • A file with size exceeding the specified value is not cached. • In Web applications where the file size is not specified, the value specified in its property is used. However, in Web applications where the file size is specified, the value specified in its property is not used. • Specifies an integer value from 0 to 2147483647. If 0 is specified, restrictions are not set for the cacheable file size. • If an invalid value is set in this property and if the value is greater than the value specified in the memory size for each Web application, the default value is used. • If a null character string or blank character is set in this property, the default value is used.

Important note

The following parameters are used in the static contents cache functionality and therefore, cannot be used optionally in the <context-param> tag of the DD:

- com.hitachi.software.web.static_content.cache.enabled
- com.hitachi.software.web.static_content.cache.size
- com.hitachi.software.web.static_content.cache.filesize.threshold

An example of the DD definition is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC '-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN'
'http: //java.sun.com/dtd/web-app_2_3.dtd'>

<web-app>
 <context-param>
 <param-name>
 com.hitachi.software.web.static_content.cache.enabled
 </param-name>-
 <param-value>true</param-value>
 </context-param>

 <context-param>
 <param-name>
 com.hitachi.software.web.static_content.cache.size
 </param-name>
 <param-value>5242880</param-value>
 </context-param>

 <context-param>
 <param-name>
 com.hitachi.software.web.static_content.cache.filesize.threshold
 </param-name>
 <param-value>102400</param-value>
 </context-param>
</web-app>
```

The above definition example defines the following contents:

- The static contents cache functionality is enabled.
- The memory size of the Web application is set to 5 MB.
- The upper limit of the file size that permits cache is set to 100 KB.

  Note that if an invalid value or null character is specified in the DD, the settings in the Web container (Easy Setup definition file settings) are used.

2.21.3 Execution environment settings

To define the static contents cache for each Web container, you must set up the J2EE server.

To define the static contents cache for each Web application, you must set up the Web applications. Reference these settings only when you want to set or change the properties of Web applications that do not contain cosminexus.xml.

Tip

Whether you specify settings in the Web container or in the Web application, the settings in the Web application are given priority.

(1) J2EE server settings (settings for each Web container)

Implement the J2EE server settings in the Easy Setup definition file. Specify the definition of the static contents cache for the Web container in the <configuration> tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file.

The following table lists the definitions of static contents cache in the Easy Setup definition file:

Table 2-65: Definitions of the static contents cache in the Easy Setup definition file

Items	Parameter to be specified	Setting contents
Enabling and disabling of the static contents cache functionality	`webserver.static_content.cache.enabled`	Specifies the enabling, disabling, or forced disabling of the static contents cache.
Memory size in each Web application	`webserver.static_content.cache.size`	Specifies the memory size in each Web application for which static contents cache is permitted.
File size permitting cache	`webserver.static_content.cache.filesize.threshold`	Specifies the upper limit for the file size in the Web application for which static contents cache is permitted.

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

(2) Web application settings (Settings for each Web application)

Implement the Web application settings in the execution environment by using the server management commands and property files. Use the WAR property file to define the static contents cache.

The tags specified in the WAR property file correspond to the DD. For details on definitions in the DD, see *2.21.2 Definition in the DD (Settings for each Web application)*.

2.22 URI decode functionality

You can use the URI decode functionality in the Web server integration and the in-process HTTP server.

This section describes the URI decode functionality.

The following table describes the organization of this section:

Table 2-66: Organization of this section (URI decode functionality)

Category	Title	Reference
Description	Overview of URI decode functionality	*2.22.1*
Settings	Execution environment settings (J2EE server settings)	*2.22.2*
Notes	Precautions for using the URI decode functionality	*2 22.3*

Note:

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

2.22.1 Overview of URI decode functionality

The URI decode functionality is used for decoding the URL-encoded strings included in the servlet path of request URIs and in the additional path information of Application Server. However, the context path is not decoded.

To execute a Web application that does not use decoded URIs, you must not use the URI decode functionality or you must manage at the Web application machine.

The following is the description of "Servlet APIs affected when URI decode functionality is used", "Functionality using decoded strings", "Character code used for decoding", and "Execution procedure for decoding and normalizing character strings":

(1) Servlet APIs affected when using the URI decode functionality

For using the URI decode functionality, a decoded URI is considered as a return value in the following methods of the `javax.servlet.http.HttpServletRequest` interface:

- `getPathInfo` method
- `getPathTranslated` method
- `getServletPath` method

However, in the `getRequestURI` and `getRequestURL` methods, a non-decoded URL is considered as a return value.

(2) Functionality using decoded strings

For using the URI decode functionality, the decoded strings are used in the following processes:

- Matching with URL pattern of servlets and JSPs
- Matching with default mapping
- Matching with static contents
- Matching with URL pattern of filter
- Matching with the `<error-page>` tag of `web.xml` or with the `errPage` attribute of the page directive of JSPs
- Matching with URL pattern for restricting access
- Determining URL for login authentication
- Forward and include request
- Matching with URL pattern for HTTP response compression filter

- Matching with URL pattern to control the number of concurrently executed threads in the URL group

However, the context path is not decoded and is handled as the original string, so the value "404 Not Found" is considered as a return value, when the context path does not match with the context root.

The matching for the decoded character string is not performed in the following functionality of Application Server:

- Error page customization functionality of the in-process HTTP server
- Request distribution functionality by redirecting the in-process HTTP server

(3) Character code used for decoding

For using the URI decode functionality, the character code used for decoding is UTF-8.

(4) Execution procedure for decoding and normalizing character strings

URLs used in the matching processes after decoding are normalized in the request URIs sent from clients.

2.22.2 Execution environment settings (J2EE server settings)

To use the URI decode functionality, you must set up a J2EE server.

Implement the J2EE server settings in the Easy Setup definition file. Specify the definition for the URI decode functionality in `webserver.http.request.uri_decode.enabled` within the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file. With this parameter, specify whether to use the URI decode functionality or not.

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

2.22.3 Precautions for using the URI decode functionality

This subsection describes the precautions for using the URI decode functionality.

(1) Execution procedure for decoding and normalizing character strings

The URIs, normalized after decoding, are used for matching the URL patterns with the servlet path.

The URIs, normalized without decoding, are used for matching the context root with the context path.

(2) Attributes of request

The decoded values are also stored in the attributes that are added in requests during the forward or include processing. The following table describes whether the stored values are decoded for each attribute specified in requests during the forward or include processing:

Processing	Attribute	Decoding the stored value
Forward	`javax.servlet.forward.request_uri`	N
	`javax.servlet.forward.context_path`	N
	`javax.servlet.forward.servlet_path`	Y
	`javax.servlet.forward.path_info`	Y
	`javax.servlet.forward.query_string`	N
Include	`javax.servlet.include.request_uri`	N
	`javax.servlet.include.context_path`	N

Processing	Attribute	Decoding the stored value
Include	`javax.servlet.include.servlet_path`	Y
	`javax.servlet.include.path_info`	Y
	`javax.servlet.include.query_string`	N

Legend:
Y: Decoded
N: Not decoded

For details on each attribute and the values stored in each attribute, see *Servlet specifications*.

(3) Inheriting HTTP session

The context path is not decoded and is handled as the original string, so the HTTP session is inherited.

2.23 Version setup functionality of Web applications

This section describes the version setup functionality of Web applications.

The following table describes the organization of this section:

Table 2-67: Organization of this section (Version setup functionality of Web applications)

Category	Title	Reference
Description	Overview of the version setup functionality of Web applications	*2.23.1*
	Compiling and executing JSP files and tag files	*2.23.2*
Settings	Execution environment settings	*2.23.3*
Notes	Precautions for using the version setup functionality of Web applications	*2.23.4*

Note

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

2.23.1 Overview of the version setup functionality of Web applications

To execute Web applications in Application Server, the servlets and JSPs that are compliant with the version of the Web application defined in `web.xml` are executed.

The version setup functionality of Web applications is used for specifying versions, when executing Web applications. You can use this functionality to execute the Web application compliant with the specified version without changing the version of the Web application defined in `web.xml`.

The following are the differences when you use or do not use the version setup functionality of Web applications:

When using the version setup functionality of Web applications

You need not change the version of the Web application defined in `web.xml` to execute servlets and JSPs compliant with the new version, for executing the Web application created in the earlier version.

If version of the Web application to be executed is specified, a syntax check in JSP compilation and operations of servlet APIs are changed simultaneously, when you change `web.xml`. However, you cannot use the functions that must be defined in `web.xml`.

When the version setup functionality of Web applications is not used

For executing the Web application created in earlier version, you must change the version of the Web application defined in `web.xml` to execute servlets and JSPs compliant with the new version.

The following table describes the different operations of Web applications that are defined in `web.xml`, depending on the version specified by the version setup functionality of Web applications:

Table 2-68: Different operations depending on the version specified by the version setup functionality of Web applications

Version defined in web.xml	Version specified by the version setup functionality of Web applications		
	No specifications	2.4	2.5
2.2	Operating as 2.3[#1]	Operating as 2.4	Operating as 2.5
2.3	Operating as 2.3		
2.4	Operating as 2.4		
2.5, 3.0	Operating as 2.5	Operating as 2.5[#2]	

#1

When the version defined in `web.xml` is 2.2 and nothing is set up by the version setup functionality of Web applications, the operations are performed as the Web application of 2.3 version.

#2

When the version defined in `web.xml` is 2.5 and 2.4 version is set up by the version setup functionality of Web applications, the operations are performed as the Web application of 2.5 version.

! Important note

The version setup functionality of Web applications is used to support the migration of applications of the old version. We do not recommend using this functionality for developing new applications.

To develop a new application, specify the `web.xml` of the correct version.

2.23.2 Compiling and executing JSP files and tag files

This subsection describes the operations executed if the version of the JSP specifications is different in the compilation and execution of JSP files and tag files, when you use the version setup functionality of Web applications. This subsection separately describes the cases "When the JSP pre-compilation functionality is used" and "When the JSP pre-compilation functionality is not used".

(1) When the JSP pre-compilation functionality is used

When compiling the JSP files using the JSP pre-compilation functionality, the class file generated from the JSP file and the version information file both are created. The version information file is output when the JSP pre-compilation functionality is executed and the version of the JSP file is coded.

In the following two cases, versions of JSPs are different:

- When the version information file and the version of the JSP specifications, corresponding to the Web application version for executing the Web application, are different
- When the version of the class files generated from JSP files using the JSP pre-compilation functionality and its corresponding version of the JSP specifications are different

The following table describes the operations of Web applications for above two cases:

Table 2-69: When the version information file and the version of the JSP specifications are different

Timing for changing file		Operation of Web application
Before starting Web applications		Case 1
After starting Web applications	For enabling reload	Case 2
	For disabling reload	

Case 1

If the JSP pre-compilation command is executed before starting a Web application, the message KDJE39522-E is output, when an attempt to start the Web application fails.

Case 2

Operations are same as the case 1 after re-starting the J2EE server or Web applications.

Table 2-70: When versions of class files and the JSP specifications are different

Timing for changing file		Operation of Web application
Before starting Web applications		Case 3
After starting Web applications	For enabling reload	Case 4

Timing for changing file		Operation of Web application
After starting Web applications	For disabling reload	Case 5

Case 3

- In JSP files where <load-on-startup> is specified in web.xml

  If true is specified in the WAR property file or the <start-notify-error> tag of cosminexus.xml, or if the tag specifications are omitted, the message KDJE39298-E will output, when an attempt to start the Web application fails.

  If false is specified in the <start-notify-error> tag of cosminexus.xml, the message KDJE39298-E will output when you start the Web application, but the Web application will start successfully. However, during the request, the message KDJE39298-E will output, and the error code 500 (Internal Server Error) will returned.

- In JSP files in which <load-on-startup> is not specified in web.xml

  During the initial request, the message KDJE39298-E will output, and the error code 500 (Internal Server Error) will returned.

Case 4

- In the executed JSP file

  Operations are same as the case 3 after re-starting the J2EE server or Web application.

- In the non-executed JSP file

  Operations are same as the case 3 "In JSP files in which <load-on-startup> is not specified in web.xml".

Case 5

- In the executed JSP file

  When executing reload, the message KDJE39317-E will output resulting in the failure of JSP reload (error code 500 (Internal Server Error) returns from the corresponding request).

- In the non-executed JSP file

  Operations are the same as in case 4 "In the non-executed JSP file".

! Important note

- The cases (case 3, case 4, and case 5), with only different class files, are considered for overwriting the class files with different versions that are compiled by the JSP pre-compilation command. As a result, you cannot use the JSP pre-compilation functionality for starting Web applications.
- You cannot use the JSP pre-compilation functionality for starting Web applications because you cannot update the version information file after starting the Web application (case 2).
- In the case 1, when you start a Web application using the JSP pre-compilation functionality that is used for starting Web applications, an error does not occur because you have recreated the version information file itself. Also, for executing the JSP pre-compilation command before starting Web applications, the command will check the version. Also, when executing the command, a version mismatch might be detected and an error might occur.
- When you compile JSPs by specifying a separate JSP file during JSP pre-compilation, the version of the JSP specifications specified in the existing version information file will be compared with the version of the JSP specifications during the compilation. In such cases, if the versions will be different, the message KDJE39522-E will output and an error will occur. If a separate JSP file is not specified, the version information file will be recreated, and an error will not occur.

(2) When the JSP pre-compilation functionality is not used

When different class files are used for the Web application version during the compilation and execution, the message KDJE39334-I will output, and the JSP files and tag files will be compiled again.

The following table describes the re-compilation operation when the Web application during the compilation and execution are different:

Version for compilation			Version for execution (version of JSP specifications)			
Version of web.xml	File type	Version of TLD	2.2	2.3	2.4	2.5
2.2	JSP file	--	--	--	2.0	2.1
2.3		--	--	N	2.0	2.1
2.4	JSP file	--	1.2#1	1.2	N	2.1
	Tag file	2.0	--	--	N	N#2
2.5, 3.0	JSP file	--	1.2#1	1.2	2.0	N
	Tag file	2.0	--	--	N	N#2
		2.1	--	--	2.0	N

Legend:

--: Not applicable

N: Not re-compiled

1.2: Re-compiled as per the JSP1.2 specifications

2.0: Re-compiled as per the JSP2.0 specifications

2.1: Re-compiled as per the JSP2.1 specifications

#1

If the version of `web.xml` is 2.2, operations are performed as Web application version 2.3, and therefore, re-compilation is done as per the JSP1.2 specifications.

#2

In the JSP2.1 specifications, the JSP specifications compliant with the JSP version defined in the TLD file are determined for tag files. Even if the Web application version is 2.5, the tag files can be executed as per the JSP2.0 specifications, and therefore, the file is not compiled again.

2.23.3 Execution environment settings

To use the version setup functionality of Web applications, you must set up the J2EE server and the JSP pre-compilation commands.

(1) J2EE server settings

Implement the J2EE server settings in the Easy Setup definition file. Specify the executing version for the Web application version setup functionality of Web applications in `webserver.application.lower_version` within the `<configuration>` tag of the logical J2EE server (j2ee-server) within the Easy Setup definition file. In this parameter, specify whether to use the version setup functionality of Web applications.

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

(2) Settings for JSP pre-compilation commands

Implement the JSP pre-compilation settings in the `-lowerversion` option of the `cjjspc` command. For details on the `cjjspc` command, see *cjjspc (JSP pre-compilation)* in the *uCosminexus Application Server Command Reference Guide*.

2.23.4 Precautions for using the version setup functionality of Web applications

This subsection describes the precautions for using the version setup functionality of Web applications.

(1) Using annotations

Even when you use the version setup functionality of Web applications, when specifying the version of Web applications in which annotations can be used, you cannot read the annotation information.

(2) Exporting J2EE applications

When Web applications are exported by enabling the version setup functionality of Web applications, the version specified by the version setup functionality of Web applications is not changed, and the Web applications are exported. The Web applications, with the same version as imported, will be exported.

2.24 Precautions related to the Web container

- The maximum size of the POST data that can be handled in a Web container is 2 GB. Also, the maximum size of the POST data that can be handled is regulated depending on the settings of the Web container, and the settings of the Web server and redirector in front of the Web container.
- The Web container returns an `Eta` (`Entity Tag`) that consists of the file size and the last updated date and time and does not include a unique ID (incode) allocated to the file.

3 Using JSF and JSTL

This chapter describes the details on using JSF and JSTL.

3.1 Organization of this chapter

This chapter describes the details on using JSF and JSTL.

Java Server Faces (JSF) is a Web application framework provided with the standard specification of JavaEE.

Java Server Pages Standard Tag Library (JSTL) is a tag library that has all the tags frequently used in Web applications.

The following table describes the organization of this chapter:

Table 3-1: Organization of this chapter (Using JSF and JSTL)

Classification	Title	Reference
Guide	Overview of JSF and JSTL	*3.2*
	JSF and JSTL functionality	*3.3*
Implementation	Setting up a class path (Setting up a development environment)	*3.4*
	Definitions in a DD	*3.5*
	JSF applications development flow	*3.6*
	Using a log (development inspection log) for debugging	*3.7*
Setup	Setting up an execution environment	*3.8*
Operations	To output and check the troubleshooting information	*3.9*
Notes	Notes on using the JSF and JSTL	*3.10*

3.2 Overview of JSF and JSTL

This section gives an overview of the JSF and JSTL. Note that you can use the Bean Validation functionality from the JSF.

3.2.1 Overview of JSF

(1) About JSF

The *JSF* is a framework defined as a standard specification in JaveEE to develop a user interface for Web applications. The JSF aims at making the development of Web applications more efficient. A Web application developed by using the JSF is called a *JSF application*.

The JSF distinctly differentiates between Model class (business logic) development and View class (input and output windows) development based on the MVC model, thereby simplifying the division of work among JSP developers. Developers can focus on their own work, which in turn improves the development efficiency.

(2) Linkage with Bean Validation

With Application Server, you can use the Bean Validation functionality to simplify the process for validating the values entered in a JSF application.

For details on using Bean Validation, see *10. Using Bean Validation with Application Server* in the *uCosminexus Application Server Common Container Functionality Guide*.

3.2.2 JSTL

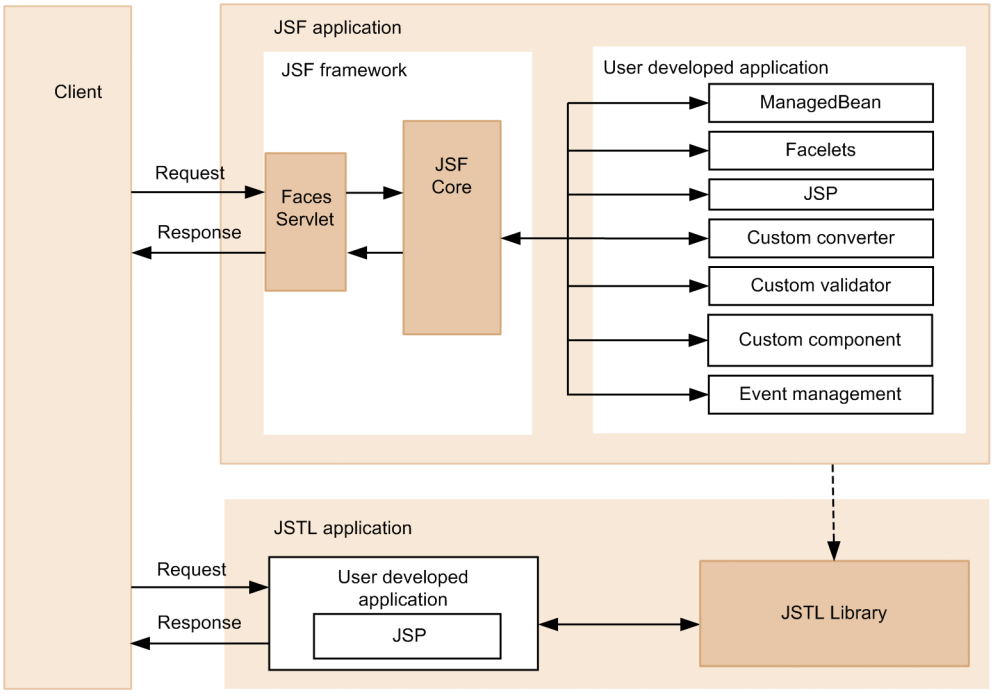
The *JSTL* is a library of tags that are used to express common processes, such as the database access process, and the loop process executed in a Web application. Using the tags provided by the JSTL reduces the time required normally to describe the processes that require coding, thus improving the development efficiency.

3.3 JSF and JSTL functionality

This section describes the details on the JSF and JSTL functionality.

The following figure shows the process flow for a Web application that uses JSF and JSTL:

Figure 3-1: Process flow for a Web application that uses JSF and JSTL



- When you access a Web page that uses a JSF application from the client, a request goes to the FacesServlet servlet that exists in the JSF framework. The JSF application starts the processing with the FacesServlet as the starting point, and the JSFCore of the JSF framework processes user-defined classes or pages. The result of the request processing is returned as a response to the client through the FacesServlet.
- When you access a Web page that uses JSTL from the client, the JSTL library executes processes corresponding to the tags used in the Web page. The result of the process is returned as a response to the client.

3.3.1 JSF functionalities

This section describes the details on JSF functionalities.

(1) Basic functionalities of JSF

When you develop a Web application using JSF, you can use various JSF functionalities such as defining access to the page view, page re-use, changing or validating types of input values from the client, and controlling events in the application.

You use the ManagedBean or the Expression Language functionality to implement most of the functionalities of JSF.

ManagedBean is a JavaBean that defines methods and the data to be used in the JSP and Facelets pages. For details on ManagedBean, see the *JSF specifications*.

The **Expression Language functionality** associates properties and methods defined in the ManagedBean with attributes of the `JSF` tags by coding properties and methods in a prescribed format. The Expression Language functionality is a part of the JSP specifications. For details, see the *JSF specifications*.

(2) Operations of Bean Validation with Application Server

The following table describes functionalities of the Bean Validation that can be used from JSF.

Table 3-2: Functionalities of Bean Validation that can be used from JSF

No.	Functionality		Overview
1	Validation of input values		This functionality uses the specified validation definitions to verify the values set in ManagedBean.
		Group management functionality	This functionality groups validations.
		Message management functionality	This functionality manages the messages that are returned when the validation result is an error.
2	Create custom validator		This functionality creates independent validation processes.

You can use the functionalities of the Bean Validation by specifying annotations. The following table describes annotation classes that can be specified for using the functionalities of the Bean Validation. The table also describes the types of variables that can be specified as well as the operations performed when an invalid type of variable is specified.

Table 3-3: Annotation classes and variable types that can be specified in the Bean Validation

No.	Annotation class	Variable type that can be specified	Behavior when an invalid variable type is specified	Remarks
1	`Null`	You can specify any type of variable	--	--
2	`NotNull`	You can specify any type of variable	--	--
3	`AssertTrue`	• *boolean/java.lang.Boolean*	`javax.validation.UnexpectedTypeException` is thrown.	--
4	`AssertFalse`	• *boolean/java.lang.Boolean*	`javax.validation.UnexpectedTypeException` is thrown.	--
5	`Min`	• *java.math.BigDecimal* • *java.math.BigInteger* • *byte/java.lang.Byte* • *short/java.lang.Short* • *int/java.lang.Integer* • *long/java.lang.Long* • *float/java.lang.Float* • *double/java.lang.Double* • *java.lang.String*	`javax.validation.UnexpectedTypeException` is thrown.	--
6	`Max`	• *java.math.BigDecimal* • *java.math.BigInteger* • *byte/java.lang.Byte* • *short/java.lang.Short* • *int/java.lang.Integer* • *long/java.lang.Long* • *float/java.lang.Float* • *double/java.lang.Double* • *java.lang.String*	`javax.validation.UnexpectedTypeException` is thrown.	--

No.	Annotation class	Variable type that can be specified	Behavior when an invalid variable type is specified	Remarks
7	`DecimalMin`	• *java.math.BigDecimal* • *java.math.BigInteger* • *java.lang.String* • *byte/java.lang.Byte* • *short/java.lang.Short* • *int/java.lang.Integer* • *long/java.lang.Long* • *float/java.lang.Float* • *double/java.lang.Double*	`javax.validation.UnexpectedTypeException` is thrown.	`javax.validation.ValidationException` is thrown if you specify a value that cannot be parsed by `java.math.BigDecimal`, for the `value` attribute.
8	`DecimalMax`	• *java.math.BigDecimal* • *java.math.BigInteger* • *java.lang.String* • *byte/java.lang.Byte* • *short/java.lang.Short* • *int/java.lang.Integer* • *long/java.lang.Long* • *float/java.lang.Float* • *double/java.lang.Double*	`javax.validation.UnexpectedTypeException` is thrown.	`javax.validation.ValidationException` is thrown if you specify a value that cannot be parsed by `java.math.BigDecimal`, for the `value` attribute.
9	`Size`	• *java.lang.String* • *java.util.Collection* • *java.util.Map* • *array*	`javax.validation.UnexpectedTypeException` is thrown.	`javax.validation.ValidationException` is thrown if you specify a negative value for the `max` or `min` attribute. `javax.validation.ValidationException` is thrown if you specify a value greater than max, in min.
10	`Digits`	• *java.math.BigDecimal* • *java.math.BigInteger* • *java.lang.String* • *byte/java.lang.Byte* • *short/java.lang.Short* • *int/java.lang.Integer* • *long/java.lang.Long* • *float/java.lang.Float* • *double/java.lang.Double*	`javax.validation.UnexpectedTypeException` is thrown.	`javax.validation.ValidationException` is thrown if you specify a negative value for the `integer` or `fraction` attribute.
11	`Past`	• *java.util.Date* • *java.util.Calendar*	`javax.validation.UnexpectedTypeException` is thrown.	--
12	`Future`	• *java.util.Date* • *java.util.Calendar*	`javax.validation.UnexpectedTypeException` is thrown.	--
13	`Pattern`	• *java.lang.String*	`javax.validation.UnexpectedTypeException` is thrown.	`javax.validation.ValidationException` is thrown if the value that you have specified for the `regexp` attribute is incorrect as a regular expression.

No.	Annotation class	Variable type that can be specified	Behavior when an invalid variable type is specified	Remarks
13	`Pattern`	• *java.lang.String*	`javax.validation.UnexpectedTypeException` is thrown.	Whether an expression is regular depends on the java.util.regex.Pattern specifications.

Legend:
--: Not Applicable.

(3) JSF operations in Application Server

JSF operations in Application Server are as follows:

- If you use the Expression Language to specify methods that function as `ValueChangeListener`, `ActionListener`, `AjaxBehaviorListener`, and `ComponentSystemEventListener`, and if both, a method with arguments and a method without arguments exist with the same name, the method with arguments is invoked.
- If multiple components are present in the *f:facet* tag, the component coded first is processed in a JSP page, whereas in a Facelets page, all the coded components are processed.
- If the *f:subview* tag is coded outside the *f:view* tag, in JSP pages, the value of the *f:view* is overwritten with the value of *f:subview*. In Facelets pages, the value of *f:view* does not change and *f:view* and *f:subview* are displayed together.
- You must code the *h:head* tag when using the *f:ajax* tag. *f:ajax* does not function unless the *h:head* tag is coded.
- When using the `valueChangeListener` attribute or the *f:valueChangeListener* tag for processing an event, if you specify `ValueExpression` in the `value` attribute of the *h:inputText* tag and `RequestScoped` in the `scope` attribute, the `valueChangeListener` method is executed even if you do not change the input value.
- If the user application registers a Cookie called *csfcfc* and sends repeated requests to the Facelets by using the `Flash` object, an error occurs while reading the data of the `Flash` object and exception (`NullPointerException`) handling is done.
- The retry count of the *ui:repeat* tag will be the value specified in the `size` attribute + 1.
- Attributes of some tags might become invalid due to the browser specifications. We therefore recommend that you check the browser specifications.
- To enable the `required` attribute of tags such as *composite:attribute*, *composite:facet*, *composite:insertFacet*, and *composite:renderFacet* tags, set the value of the context parameter `javax.faces.PROJECT_STAGE` to Development.
- The value you specify in the context parameter `javax.faces.FACELETS_REFRESH_PERIOD` serves as the time from receiving the latest request by the Facelets page until the Facelets file update is confirmed.
- Therefore, in an environment where many requests are sent in a short period of time, the time until the confirmation of the update is extended according to the latest request, and there is a possibility that a state continues where the Facelets file is not updated.
- The Bean Validation functionality is usually enabled in Application Server. For details on how to disable the Bean Validation functionality, see*10.5.1 Procedure of using Bean Validation from JSF* in the *uCosminexus Application Server Common Container Functionality Guide*.
- Specify the attribute as the target for specifying a value in the `f:attribute` tag, in which the character string (having `java.lang.String` type) is treated as the specified value.
- Note that the parameter count of the requests included in the POST data from the client window of the JSF application differs depending on the type of the JSP tag to be used and the usage method. When you want to set a maximum value of the request parameters count in the `webserver.connector.limit.max_parameter_count` key, we recommend that you validate by using the real JSF application on the same environment as the real environment.
- When the resources of `META-INF/resources` within the JAR file of the application are updated, the message KDJE39556-W is output at the time of reload. If you access the resource from a JSF when the message is output, you can acquire the updated resources.

3.3.2 JSTL functionality

In the JSTL, you can execute processes that are used in common in applications by defining tags in pages.

The JSTL includes tags related to setting values, conditional sorting, database access, internationalization, and XML parsing.

3.3.3 Proprietary functionalities of Application Server

The following table describes the proprietary functionalities of Application Server available for JSTL and JSF.

Table 3-4: Proprietary functionalities of Application Server

No.	Functionality	Explanation
1	Log output	This functionality records the execution related information in a log file.
2	Output of the performance analysis trace	This functionality records the trace of the start and end of specific functionalities (methods) in a file. Use this information to analyze the system performance and performance bottlenecks. For details on the output of the performance analysis trace, see *7. Performance Analysis by Using Trace based Performance Analysis* in the *uCosminexus Application Server Maintenance and Migration Guide*.

3.3.4 Collaboration with other functionalities of Application Server

This section describes the collaboration of JSF and JSTL with other functionalities of Application Server.

The following table describes the functionalities that you must consider when using in combination with JSF or JSTL.

Table 3-5: Functionalities that you must consider when using in combination with JSF or JSTL

Item No.	Functionality	Reference
1	Explicit memory management functionality	*7. Controlling full garbage collection by using the explicit memory management functionality* in the *Expansion Guide*
2	Session failover functionality	*5.2 Overview of the session failover functionality* in the *Expansion Guide*
3	Re-deploy and re-load functionality	*13. Format and deployment of J2EE application* in the *Common Container Functionality Guide*
4	JSP pre-compile functionality	*2.5 JSP pre-compilation functionality and maintaining compilation results*
5	Giving optional names to J2EE resources (user specified name space functionality)	*2. Naming control* in the *Common Container Functionality Guide*

Note:
The manual name *uCosminexus Application Server* is omitted.

The sections hereafter describe the collaboration of JSF and JSTL with each functionality.

(1) Explicit memory management functionality

With JSF, the following information and objects generated based on the user-created Facelets files or JSP files are registered in HTTP sessions:

- Information of HTML page windows (View state)
- Objects of the ManagedBean class with the SessionScope defined

You can use the explicit memory management functionality to control the same objects and information, as used in other Web applications.

However, whether the objects and information are registered in HTTP sessions is subject to conditions. Furthermore, not all the information registered in HTTP sessions is necessarily managed with the explicit memory management functionality. The following table describes the conditions when information and objects can be registered in HTTP sessions and whether the registered information or objects can be managed with the explicit memory management functionality.

Table 3-6: Conditions for registering the information and objects in HTTP sessions

Information or object	Condition for registering in HTTP sessions	Whether the explicit memory management functionality can be used for controlling
View information of the UIComponent (information of the View that comprises I/O interface components such as text fields, radio buttons, and submit buttons for enabling the user interaction)	When the value of the JSF standard context parameter (`javax.faces.STATE_SAVING_METHOD`) is "`server`"(default value)	Not used
ManagedBean object	When you specify the `SessionScope` annotation, or "session" in the *managed-bean-scope* in `faces-config.xml`.	Used
Character code to be used in pages	When an HTTP session is generated	Used
Objects registered in the SessionMap	When used in a user application	Used

If you intend to use the explicit memory management functionality, you must calculate the approximate memory size that the JSP application requires for the explicit memory management area. You can calculate the memory size with the following formula:

Memory size used by a JSF application for the explicit memory management area

```
The memory size used for the explicit memory management area in one session
=(A#+1) X 0.4 KB
+ ManagedBean object size (when registering in the HTTP session)
+ Object size when registering in the SessionMap
```

"A" indicates the value specified in the property (`com.sun.faces.numberOfLogicalViews`) that is used for setting the maximum value of a JSF logical page. The default value is `15`.

Notes on using the explicit memory management functionality in a JSF application

HTTP sessions are not discarded (the invalidate method of the `javax.servlet.http.HttpServletRequest` interface is not called) in JSF. You must therefore discard HTTP sessions (call the invalidate method of the `javax.servlet.http.HttpServletRequest` interface) in user applications, or set an adequate session timeout.

(2) Session failover functionality

The JSF can use the session failover functionality in the same way as for other Web applications with objects registered in the HTTP session. You need not perform any JSF specific settings for using this functionality.

Size of objects registered in an HTTP session by a JSF application

If you intend to use the session failover functionality, use the values listed in the following table to estimate the size of objects that the JSF application registers in an HTTP session.

Table 3-7: Size of objects registered in HTTP session by JSF application

Page	Operations that use memory	Memory used
Facelets page	Mandatory objects and all tags	1.3 KB
	Page generation part of one Form tag	0.2 KB[#1]
	If Expression Language is used[#2]	0.8 KB[#3]

Page	Operations that use memory	Memory used
JSP page	Mandatory objects and all the tags	2.2 KB
	Page generation part of one Form tag	2.0 KB[#1]
	If Expression Language is used[#2]	0.8 KB[#3]

#1
The size might vary depending upon how ManagedBean is created or the ID settings of the tag.

#2
You must estimate the size for each individual resource.

#3
The size might vary depending upon how the resources are set up.

Note that the memory size values described in the table are estimated values. Use the information retrieved after executing the application to calculate the actual memory size required for an HTTP session.

Notes on using the session failover functionality in a JSF application

With the session failover functionality, if there is information that cannot be serialized with the serialization, when inheriting attributes of an HTTP session, serialization fails and you cannot save the information. This applies also to JSF applications. Serialization fails in the following cases and you cannot use the session failover functionality:

- If you specify either the `SessionScope` annotation, or "session" in the *managed-bean-scope* tag of `faces-config.xml`, and ManagedBean includes the information that cannot be serialized
- If the information that cannot be serialized is registered in SessionMap

(3) Redeploy and reload functionality

You need not consider any special points for using the redeploy functionality in JSF applications. You can use the functionality in the same way as in other Web applications.

Execution of the reload functionality from a command prompt also requires no special consideration. You can use the functionality in JSF applications in the same way as in other Web applications.

However, the files that are reloaded based on the update detection have the following limitations:

Update Detection of JSP files

The update detection of JSP files is the same as the update detection in other Web applications.

Update Detection of Facelets files

Facelets files are not subject to the update detection.

The following table describes the applicability of the reload functionality when updating files in Facelets.

Table 3-8: The applicability of the reload functionality when updating files in Facelets

Files subject to update detection	Applicability of the reload functionality		
	app	web	jsp
Facelets file	N	N	N
Tag file	Y	Y	Y
ManagedBean compilation result	Y	Y	Y
Static content	N	N	N

Legend:
Y: Applicable
N: Not applicable

Servlets or JSP files that are loaded with a class loader are subject to monitoring and hence the J2EE server detects when files are updated, and then executes the reload functionality. However, a class loader does not load Facelets

files. Accordingly, the J2EE server does not detect whether files have been updated, and as a result does not execute the reload functionality.

To automatically detect and apply the updates made to the Facelets files, specify `javax.faces.FACELETS_REFRESH_PERIOD` as the standard context parameter. Facelets files are monitored for updates at regular intervals as set in the parameter, and any update detected is applied. Note that you can use this functionality only for applications in the exploded archive format.

If you update a Facelets file after executing the page output of the file, the JSF detects updates when the page is accessed next time, and displays the `KDJE59227-I` message.

(4) JSP precompile functionality

You can use the JSP precompile functionality to compile JSP files in a JSF application.

However, you must explicitly specify the tag library and the class library to be used in the `-classpath` option when you use the `cjjspc` command to compile JSP files in the JSF application.

The following example shows how to specify the `-classpath` option.

The following example shows how to specify the `-classpath` option in the `cjjspc` command for compiling JSP files in the JSF application (in Windows):

```
-classpath Cosminexus-application-server-installation-directory /CC/lib/
cjsf.jar; Cosminexus-application-server-installation-directory /CC/lib/
cjsf.jar
```

(5) Assigning an optional name to J2EE resources (user-specified name space functionality)

You cannot assign optional names to J2EE resources in the JSTL.

Specify the relative path from `java:comp/env` in the `datasource` attribute of the *sql:setDataSource* tag in the JSTL.

3.4 Setting up the class path (setting up the development environment)

This section describes how to set the class path required for using JSF and JSTL.

3.4.1 File storage location

The following table describes the storage location of JSF and JSTL libraries that are installed during the Application Server installation.

Table 3-9: Storage location of JSF and JSTL libraries

Type	File overview	File name	Class path
JSF	A library for consolidating interfaces and implementations	`cjsf.jar`	*Cosminexus-application-server-installation-directory* `\CC\lib\cjsf.jar`
JSTL	A library for consolidating interfaces and implementations	`cjstl.jar`	*Cosminexus-application-server-installation-directory* `\CC\lib\cjsf.jar`

Note:
In UNIX, replace the *Cosminexus-application-server-installation-directory* with `/opt/Cosminexus` and `\` with `/`.

3.4.2 Setting up the class path

Set the class path in the `add.class.path` key in the `usrconf.cfg` file (the option definition file for a Java application) of the J2EE Server to run the applications that use JSF and JSTL with Application Server.

The following example is for the Windows environment:

```
add.class.path=Cosminexus-application-server-installation-directory\CC\lib\cjsf.jar
add.class.path=Cosminexus-application-server-installation-directory\CC\lib\cjstl.jar
```

! Important note

Concurrently specifying libraries with different versions in the class path might lead to abnormal behavior of the application. You must specify a library that is compatible with the application version.

For details on settings required to develop a JSF application in the Developer environment, see*Appendix L.1 When using JSF* in the *uCosminexus Application Server Application Development Guide*.

3.5 Definition in the DD

You must set up the context parameters in the DD (`web.xml`) to use a JSF application.

This section describes how to set up the JSF context parameters in the DD.

3.5.1 Standard context parameters

The following table describes the standard context parameters of JSF.

Table 3-10: Standard context parameter of JSF

Parameter name	Data type	Specifiable value	Behavior when an invalid value is specified	Default value	Explanation
`javax.faces.CONFIG_FILES`	`String`	A comma (,) or semicolon (;) delimited list of JSF setup files under the current context root	Ignores the invalid configuration file.	`/WEB-INF/faces-config.xml`	Specifies the path of the JSF setup file used in the application.
`javax.faces.DATETIMECONVERTER_DEFAULT_TIMEZONE_IS_SYSTEM_TIMEZONE`	`Boolean`	`true` or `false`	`false`	`false`	Specifies whether to use GMT in the time zone that is set with the `convertDateTime` tag.
`javax.faces.DECORATORS`	`String`	A semicolon (;) delimited list of class names in the `javax.faces.view.facelets.TagDecoratorlist` type and without the constructor argument	Ignores the specified class	"" (empty)	Specifies the user defined Decorator class.
`javax.faces.DEFAULT_SUFFIX`	`String`	A space delimited list of page extensions.	`.xhtml` `.view` `.xml` `.jsp`	`.xhtml` `.view` `.xml` `.jsp`	Specifies the suffix of the file used as a JSF page.
`javax.faces.DISABLE_FACELET_JSF_VIEWHANDLER`	`Boolean`	`true` or `false`	`false`	`false`	Specifies whether to use the Facelets view handler in the application.
`javax.faces.FACELETS_BUFFER_SIZE`	`int`	1 to 2147483647	`1024`	`1024`	Specifies the buffer size of the stream used while returning a response page to the client.
`javax.faces.FACELETS_LIBRARIES`	`String`	A semicolon (;)delimited list of `Facelets` tag library paths in the application root.	Ignores the library file of the specified tag.	"" (empty)	Specifies the path of the tag library file used in the user-defined Facelets.
`javax.faces.FACELETS_REFRESH_PERIOD`	`int`	-2147483648 to 2147483647	`2`	`2`	Specifies the interval in milliseconds at which the JSF checks the Facelets files for changes when a Facelets page is requested. [#1]
`javax.faces.FACELETS_RESOURCE_RESOLVER`	`String`	A valid java class name that inherits the javax.faces.view.facelet	Ignores the	"" (empty)	Specifies the user defined ResourceResolver class.

Parameter name	Data type	Specifiable value	Behavior when an invalid value is specified	Default value	Explanation
javax.faces.FACELETS_RESOURCE_RESOLVER	String	s. (In ResourceResolver, define a constructor with no arguments or a constructor that has one argument of the ResourceResolver type)	specified class.	"" (empty)	Specifies the user defined ResourceResolver class.
javax.faces.FACELETS_SKIP_COMMENTS	Boolean	true or false	false	false	Specifies whether to output the comments described in the Facelets file to the response page.
javax.faces.FACELETS_VIEW_MAPPINGS	String	A semicolon (;) delimited list of strings that either starts or ends with "*" is considered as a valid value.	Ignores the specified string.	"" (empty)	Specifies a file name pattern used for recognizing Facelets files.
javax.faces.FULL_STATE_SAVING_VIEW_IDS	String	A comma (,) separated list of strings that indicate view ID.	Ignores the specified string.	"" (empty)	Specifies the ID of the view for which you want to save the entire state. You can no longer use the method for saving the state partially in a view specified using this parameter. #2
javax.faces.INTERPRET_EMPTY_STRING_SUBMITTED_VALUES_AS_NULL	Boolean	true or false	false	false	Specifies whether to convert the submitted value (if the value is empty) to null in the JSF internally.
javax.faces.LIFECYCLE_ID	String	Java ID name	DEFAULT	No warning message is output, when you call the JSF application. However, the IllegalArgumentException handling is done when the FacesServlet starts.	Specifies the user-defined life cycle ID.
javax.faces.PARTIAL_STATE_SAVING	Boolean	true or false	true	true	Specifies whether you can use a method to partially save the view state in the application.
javax.faces.PROJECT_STAGE	String	Production, Development, UnitTest, or SystemTest	Production	Production	Specifies the value according to the software development phase.
javax.faces.SEPARATOR_CHAR	Character	Any identifiable string that can be used for the web.xml parsing	First character of the string	:	Specifies a character to separate the Id attribute of tags output to the response page.

Parameter name	Data type	Specifiable value	Behavior when an invalid value is specified	Default value	Explanation
javax.faces.STATE_SAVING_METHOD	String	The client or the server	server	server	Specifies how to save the state of the view.
javax.faces.VALIDATE_EMPTY_FIELDS	String	auto, true, or false	false	auto	Specifies whether to validate a submitted value, if the value is empty or null.
javax.faces.validator.DISABLE_DEFAULT_BEAN_VALIDATOR	Boolean	true or false	false	false	Specifies whether to disable the use of Bean Validation in an application.#3

#1

You cannot use this parameter for applications in the archive format, because the files in the archive format cannot be modified. You can, however, use this parameter for applications in the exploded archive format.

#2

Although no warning message is output if you do not specify forward slash (/) at the beginning of an ID string, you cannot save the entire state. You can save only the partial state. We therefore recommend that you start a string with a forward slash (/).

#3

For details on the operations of each setup value, see *10.5.1 Procedure of using Bean Validation from JSF* in the *uCosminexus Application Server Common Container Functionality Guide*.

3.5.2 Proprietary context parameters of Application Server

The following table describes proprietary context parameters of Application Server.

Table 3-11: Proprietary context parameters of Application Server

Parameter name	Date type	Specifiable value	Behavior when an invalid value is specified	Default value	Explanation
com.sun.faces.clientStateTimeout	Long	-9223372036854775808 to 9223372036854775807	Timeout does not work.	Timeout does not work.	Specifies the timeout value for the view state. If the time consumed between the present view and the next view exceeds the timeout value, the next view is not displayed, and javax.faces.application.ViewExpiredException is thrown. This parameter is enabled if the value of javax.faces.STATE_SAVING_METHOD is "client". If this parameter is not set, the timeout does not occur. The parameter value + 1 is considered as the valid timeout value. For example, if you set the parameter value as 0, the timeout occurs in 1 minute. If you specify a negative value, the view is definitely disabled after submit.
com.sun.faces.disableUnicodeEscaping#1	String	auto, true, or false	false	auto	Determines the output method for non-ASCII characters of the response page to be returned to the client. There are two types of output methods of non-

Parameter name	Date type	Specifiable value	Behavior when an invalid value is specified	Default value	Explanation
com.sun.faces.disableUnicodeEscaping[#1]	String	auto, true, or false	false	auto	ASCII characters, either output the characters as it is or output as the character reference. The encoding set in the response page has an impact on this functionality. The extent of the impact of this functionality differs for JSP and Facelets respectively. In JSP, characters specified in JSF tags (HTML tags or core tags) are targeted. While in Facelets, all the characters described in a page are targeted.
com.sun.faces.numberOfLogicalViews	int	0 to 2147483647	15	15	Specifies the count of logical views. [#2] This parameter is enabled if the value of javax.faces.STATE_SAVING_METHOD is "server". If you access a view that exceeds the value set in this parameter, the logical view that is not referenced for the longest period of time is replaced with the currently accessed logical view. The result page does not appear even if you submit the replaced view. If you set 0 in this parameter, the view is definitely disabled after you perform submit. If a negative value is set, the behavior differs for JSP and Facelets respectively. In JSP, although an exception occurs, the view is displayed and you can also submit the view. Note that you cannot save the view state. In Facelets, an exception occurs and the view is not displayed.
com.sun.faces.numberOfViewsInSession	int	0 to 2147483647	15	15	Specifies the number of view states that you can register in a logical view. [#2] This parameter is enabled if the value of javax.faces.STATE_SAVING_METHOD is "server". If you submit the same view for a number of times more than the value set in this parameter, the state of the view that is not referenced for the longest period of time is replaced with the state of the currently accessed view. The result page does not appear even if you submit the view that has the replaced state. If you set 0 in this parameter, the view is definitely disabled after you perform submit. If you set a negative value, the behavior differs for JSP and the Facelets. In JSP, although an exception occurs, the view is displayed and you can also

Parameter name	Date type	Specifiable value	Behavior when an invalid value is specified	Default value	Explanation
com.sun.faces.numberOfViewsInSession	int	0 to 2147483647	15	15	submit the view. Note that you cannot save the view state. In Facelets, an exception occurs and the view is not displayed.

#1

The following table describes the mapping between the values set for the com.sum.faces.disableUnicode.Escaping parameter and the output of the response page.

Table 3-12: Mapping between the values set for the com.sum.faces.disableUnicode.Escaping parameter and the output of the response page

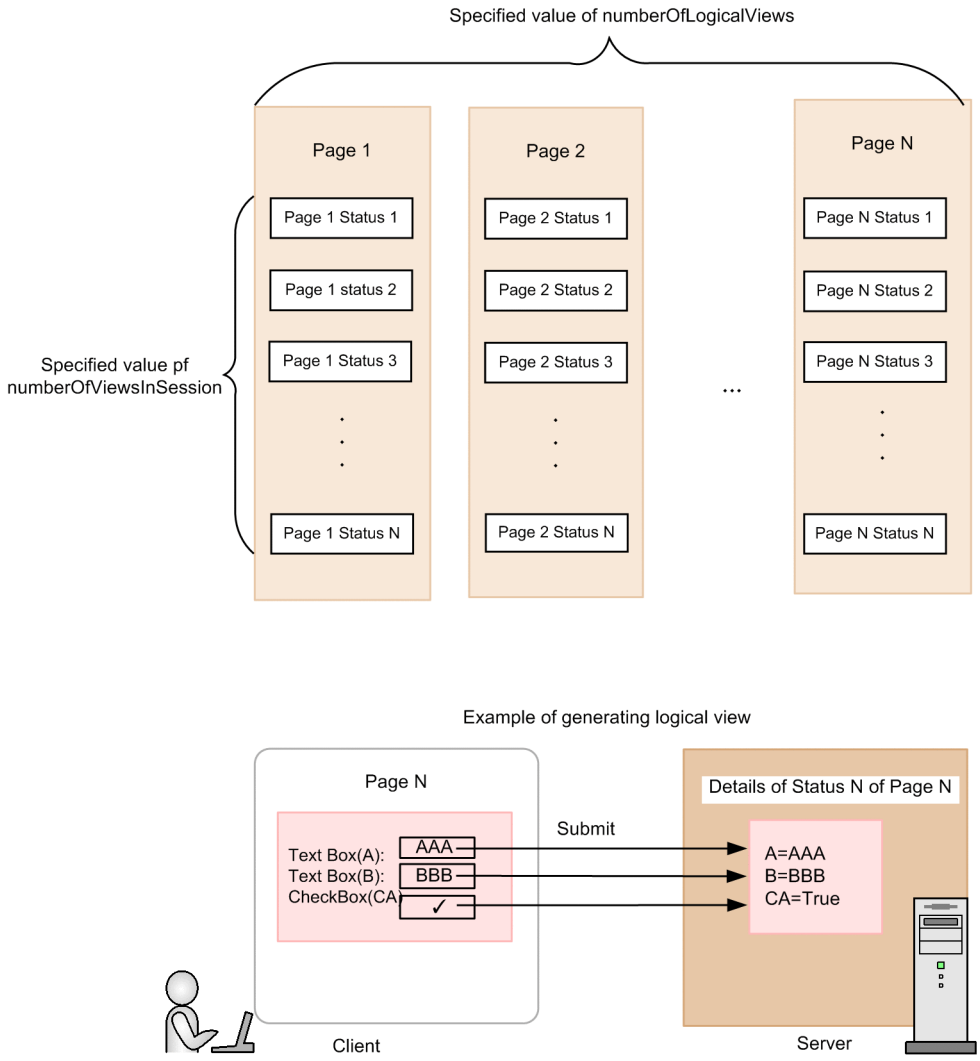
Value set for the com.sun.faces.disableUnicodeEscaping	Response page encoding is in UTF, or ISO-8859-1	Response page encoding is other than UTF, or ISO-8859-1
true	Character	Character
false	Character reference	Character reference
auto	Character	Character reference

#2

The *Logical view* indicates response pages saved by the JSF. A logical view is used to identify the views accessed so far by the client. A logical view is created when the client accesses a view. The logical view also holds the view states. A view state is incremented by one every time you submit the same view.

The following figure shows the relationship of the view and the values specified for numberOfLogicalViews and numberOfViewsInSession.

Figure 3-2: 2Relationship of the view and the values specified for numberOfLogicalViews and numberOfViewsInSession



The number of states specified for `numberOfViewsInSession` is stored for each logical view included in the number of logical views specified in `numberOfLogicalViews`.

When a client machine submits a value to the server machine, the server machine saves a new state of the view. Note that a state indicates the information about the UI components defined in a logical view.

In JSF, the window information (View state) of Facelets files created by the user or HTML pages created based on JSP files and the objects of the `ManagedBean` class that define SessionScope, are registered in the HTTP session.

3.5.3 Servlet settings

The servlets settings are defined in `web.xml`. A `web.xml` file has different definitions for different versions of servlet.

(1) For Servlet2.5

You must define the following tags in `web.xml` to run JSF applications:

- *servlet*

Use the *servlet* tag to register the `FacesServlet` class as a servlet. Make the settings in the `web.xml` as follows:

```
<servlet>
<servlet-name>FacesServlet</servlet-name>
<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
</servlet>
```

- *servlet-mapping*

You must define the `servlet-mapping` element in `web.xml`.

Make the settings in `web.xml` as follows:

```
<servlet-mapping>
<servlet-name>FacesServlet</servlet-name>
<url-pattern>/faces/*</url-pattern>
</servlet-mapping>
```

(2) For Servlet3.0

For Servlet3.0, processes such as registration of the `FacesServlet` class or definition of the URL mapping are done by default. You need not define any settings in `web.xml` and the creation of `web.xml` is also optional.

The following points describe how the program behaves depending upon whether you register the `FacesServlet` class and define the URL mapping in `web.xml`.

Condition 1:

Conditions

When either of the following conditions is fulfilled:

- You create `web.xml` but do not register the `FacesServlet` class and do not define the URL mapping
- You do not create web.xml

Behavior

FacesServlet is automatically initialized and is mapped to the following default URL:

- `/faces/*`
- `*.jsf`
- `*.faces`

The user accesses the FacesServlet by using the default URL.

Condition 2:

Condition

You create `web.xml` and register the FacesServlet class, and define the URL mapping.

Behavior

The user accesses the FacesServlet according to the contents defined in `web.xml`. In such cases, the default settings described in Condition 1 above are not used.

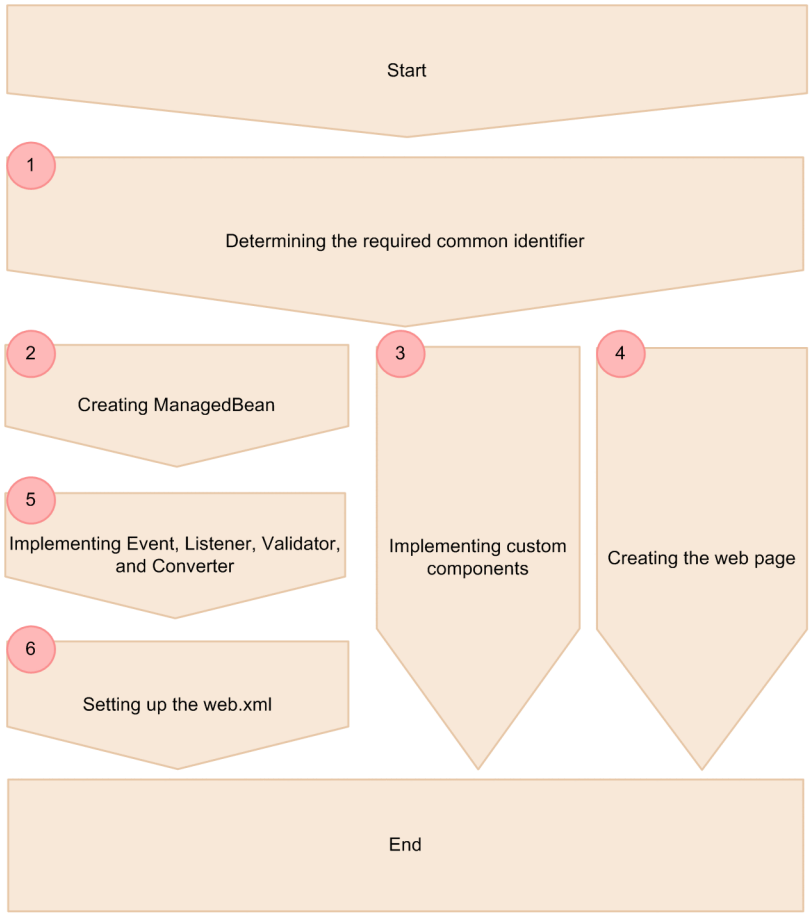
3.6 JSF applications development flow

This section describes the JSF applications development flow.

3.6.1 Procedure for developing JSF applications

The following figure shows the flow of developing a JSF application. Develop the application by performing the tasks in order of the numbering in the following figure. In step 1 of the figure, determine common identifiers necessary for creating items described from steps 2 to 6.

Figure 3-3: Flow of the process for developing a JSF application



The procedure described above covers the development procedure up to the step for creating a JSF application. To actually execute the developed JSF application, you must create an EAR file including the files described above, and deploy the JSF application on the Application Server machine.

3.6.2 Procedure for using the Bean Validation from JSF

This section describes the procedure for using the Bean Validation from JSF.

(1) Prerequisite

You must satisfy the following conditions for using the Bean Validation verification functionality in JSF:

- Create Web pages in Facelets.
- Define annotations of the Bean Validation in ManagedBean.

(2) Validation process

The validation process of the Bean Validation differs according to the values set in the context parameter of JSF. The behavior of the Bean Validation differs according to the values set in `javax.faces.validator.DISABLE_DEFAULT_BEAN_VALIDATOR` and depending upon whether the *f:validateBean* tag is specified for the ManagedBean that defines the annotation of the Bean Validation.

Table 3-13: Relation of the values set in javax.faces.validator.DISABLE_DEFAULT_BEAN_VALIDATOR and the validation process

Value set in javax.faces.validator.DISABLE_DEFAULT_BEAN_VALIDATOR	If the `<f:validateBean>` tag is specified (when the `disabled` attribute is not specified)	If validation is done by the Bean Validation
`true`	Yes	Yes
	No	No
`false`	Yes	Yes
	No	Yes
If no value is specified (default)	Same as in the case of false	Same as in the case of false

For details on how to use the *f:validateBean* tag, see *Standard specifications of JSF*.

For details on how to define a validation for ManagedBean variables, see *Standard specifications of Bean Validation*.

(3) Implementation example

The following is an example of implementation done for using the Bean Validation from JSF.

The following is an example of implementing a Facelets page that registers the information which requires validation.

```
  <f:view>
    <h:form>
        Enter ID<br/>
        <h:inputText id="IDBox" value="#{personalData.id}" /><br/>
        <h:message for="IDBox"/><br/>
        <br/>
      <f:validateBean disabled="true">
        Enter name (validation will not be performed) <br/>
        <h:inputText id="NameBox" value="#{personalData.name}" /><br/>
      </f:validateBean >
        <br/>
        Enter age <br/>
        <h:inputText id="AgeBox" value="#{personalData.age}" /><br/>
        <h:message for="AgeBox"/><br/>
        <br/>
      <br/>
    :
    :
    </h:form>
  </f:view>
```

! Important note

Although validation is defined for variables of the corresponding ManagedBean in the *inputText* tag in `<f:validateBean disabled="true">`, the validation is not performed.

Next is an implementation example of the validation definition for the ManagedBean that stores the data, which requires validation.

```
@ManagedBean(name="personalData")
@SessionScoped
public class PersonalData
{
    @Size(min=8,max=12, message="Enter a string between 8 to 12 characters inclusive.")
    private String id = "";

    @Size(min=1,message="Enter name.")
    private String name = "";
```

```
    @Max(value = 150, message="Confirm if the correct age is entered.")
    @Min(value = 0, message="Age must be 0 or above.")
    private int age = 0;
   :
setter/gettter method
   :
}
```

3.7 Using log (development investigation log) for debugging

You can use the development investigation log for debugging during the JSP application development.

You can use the development investigation log to investigate the cause or to understand more details on the behavior of the concerned functionality if an error occurs or a bug is detected during the development of a JSF application.

The JSF application outputs the development related messages to the development investigation log. While debugging the JSF application, you can identify the messages to be noted, with the respective class names that are output in messages. The following table describes the components that output the messages and the respective class names that are output.

Table 3-14: Class names output in the development investigation log of the JSF application and the Bean Validation

Component name	Class name output	Output content
JSF	`javax.faces.~` `com.sun.faces.~`	• Error messages • Messages reporting incorrect settings • Exception stack trace • Messages about the execution status of functionalities • Messages about the internal status

Note that the development investigation log is not output if the default settings are used. You need to change the settings as and when required.

For details such as the settings required to output the development investigation log, the output destination of log, the output format, and the log level, see *24.4 Messages output in the development check log* in the *uCosminexus Application Server Messages*.

Also, for details on the development investigation log of Bean Validation used in the JSF application, see *10.6 Using log (development investigation log) for debugging* in the *uCosminexus Application Server Common Container Functionality Guide*.

3.8 Setting up the execution environment

The execution environment for applications that use JSF and JSTL needs to be set up in the same way as the development environment. You must set libraries in the J2EE server class path.

For details on the class path to be set up, see *3.4 Setting up the class path (setting up the development environment)*.

3.9 To output and check the troubleshooting information

If an error occurs in the application that uses JSF and JSTL, reference the log and take the required action. For details, see *4. Output Destinations and Output Methods of Data Required for Troubleshooting* and *5. Problem Analysis* in the *uCosminexus Application Server Maintenance and Migration Guide*.

3.10 Notes on using JSF and JSTL

This section describes the points to consider when using JSF and JSTL.

- Values set in context parameters are handled according to the following rules:
  - Boolean type properties are case-sensitive.
  - All context parameters have their respective default values. If an incorrect value or a value outside the specified range is set, a default value is used and the incorrect context parameter is output to a message.
- You must consider the following points regarding the library version while setting up the class path of JSF or JSTL:
  - A library of any other version must not be specified in the class path along with the library version that you actually want to use. The application might behave abnormally.
  - You must specify a library in the class path that is compatible with the application version.

4 Web Server Integration

This chapter describes the functionality related to Web server integration.

4.1 Organization of this chapter

Application Server provides the redirector functionality for Web server integration. A *redirector* refers to a library provided in the Web container. By registering the redirector in the Web server, you will be able to process a specific requests, from the HTTP requests for the Web server, in the specified Web container and distribute and process the requests in multiple Web containers.

The following table lists the functionality and the corresponding reference sections for each functionality related to the Web server integration:

Table 4－1: Functionality and the corresponding reference sections of each functionality related to Web server integration

Functionality	Reference
Distributing requests with the Web server (Redirector)	*4.2*
Distributing requests by URL pattern	*4.3*
Distributing requests by the round-robin format	*4.4*
Distributing requests by the POST data size	*4.5*
Communication timeout (Web server integration)	*4.6*
Specifying the IP address (Web server integration)	*4.7*
Error page customization (Web server integration)#	*4.8*
Viewing the top page by specifying the domain name	*4.9*
Notification of Gateway Information to a Web Container	*4.10*

#

Functionality available only when you use Cosminexus HTTP Server as the Web server. You cannot use this functionality in the case of using Microsoft IIS.

Furthermore, in the case of Web server integration, you can also use the SSL-based authentication and the data encryption functionality of Cosminexus HTTP Server or the SSL-based authentication and the data encryption functionality of Microsoft IIS.

For SSL-based authentication and data encryption in Cosminexus HTTP Server, see *7.2.5 SSL setup with Cosminexus HTTP Server* in the *uCosminexus Application Server Security Management Guide*. For SSL-based authentication and data encryption in Microsoft IIS, see *7.2.6 Microsoft IIS setup (in Web redirector environments)* in the *uCosminexus Application Server Security Management Guide*. You can use this functionality only in the Web redirector environment.

Note that the Web server integration functionality provided in Application Server includes a functionality wherein the functions unique to the Application Server are extended beyond the functions defined in J2EE, and also those provided as functions unique to Application Server. For details about whether the functionality is unique to Application Server, see *1.2 Functionality corresponding to the purpose of the system*.

Environment settings required for Web server integration

The following environment settings are required for the Web server integration:

- **When Smart Composer is used**

  See *Appendix B Precautions related to Cosminexus HTTP Server Settings* and set up the environment of Cosminexus HTTP Server.

- **When Smart Composer is not used**

  Set up the environment of one of the following Web servers:

- Cosminexus HTTP Server (*Appendix B*)
- Microsoft IIS (*Appendix C*)

4.2 Distributing requests with the Web server (Redirector)

This section explains the distribution of requests with the redirector.

You can use this functionality only when you use the Web server integration functionality. To distribute requests, you must define distribution for the host on which the Web server or the redirector is running.

The following table describes the organization of this section:

Table 4-2: Organization of this section (Distributing requests with the Web server (redirector))

Category	Title	Reference
Description	Mechanism of request distribution with the redirector	*4.2.1*
	User-defined file for setting the request distribution method (When the Smart Composer functionality is used)	*4.2.2*
	User-defined file for setting the request distribution method (When the Smart Composer functionality is not used)	*4.2.3*
Notes	Points to be considered during Web server integration	*4.2.4*

Note:
There is no description of *Implementation*, *Settings*, and *Operations* for this functionality.

Note that the required definitions differ according to the type of the Web server used. Also, if the type of the Web server used is Cosminexus HTTP Server, the definitions also differ according to the type of the functionality used for the system building. The following table lists the types of Web servers and the definitions to be used:

Table 4-3: Web server types and the definitions to be used

Type of the Web server	Type of system building function	Definitions
Cosminexus HTTP Server	Smart Composer functionality	• Definition of Easy Setup definition file • Definition of `workers.properties` • Definition of `mod_jk.conf`
	Other than Smart Composer functionality	• Definition of `workers.properties` • Definition of `mod_jk.conf`
Microsoft IIS	--	• Definition of `workers.properties` • Definition of `uriworkermap.properties` • Definition of `isapi_redirect.conf`

Legend:
--: Not applicable

For details on distributing requests by redirection when using the in-process HTTP server, see *5.7 Request distribution with the redirector*.

4.2.1 Mechanism of request distribution with the Redirector

If you use the redirector, from among the HTTP requests sent to the Web server, specific requests can be processed in the specified Web container, and requests can be processed by distributing to multiple Web containers.

In the case of distributing requests with the redirector, use the Web container execution process, called the *worker process*# that runs behind the Web server. A worker process is used to process requests including servlets and JSPs, via the redirector. Data exchange between the Web server and a worker process is based on TCP/IP and is executed through a specific port number set by the user. To specify the redirector settings, use the setup unit that abstracts the

Web container called *worker*. The worker includes a worker indicating a stand-alone J2EE server and a worker indicating a J2EE server in a cluster configuration. The worker that forwards requests to the J2EE server is called a *forwarding worker*. A forwarding worker is the `ajp13` type worker.

#

A worker process actually acts as a J2EE server.

(1) Patterns to transfer the requests

The patterns to transfer requests from the redirector to the worker process are as follows:

- Transfer from one Web server to one worker process
- Transfer from one Web server to multiple worker processes

Note that the mechanism of request distribution is not affected even if the Web server and the worker processes are present on the same machine or on different machines.

The following figures show the patterns to transfer requests from the redirector to the worker process:

Figure 4-1: Transfer from one Web server to one worker process

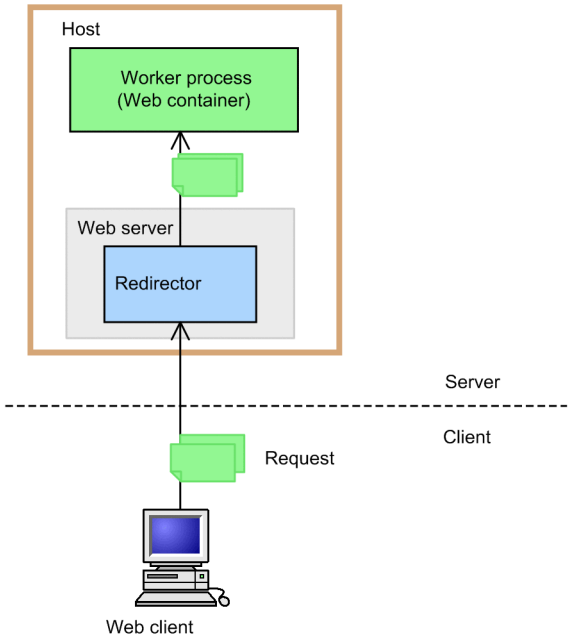
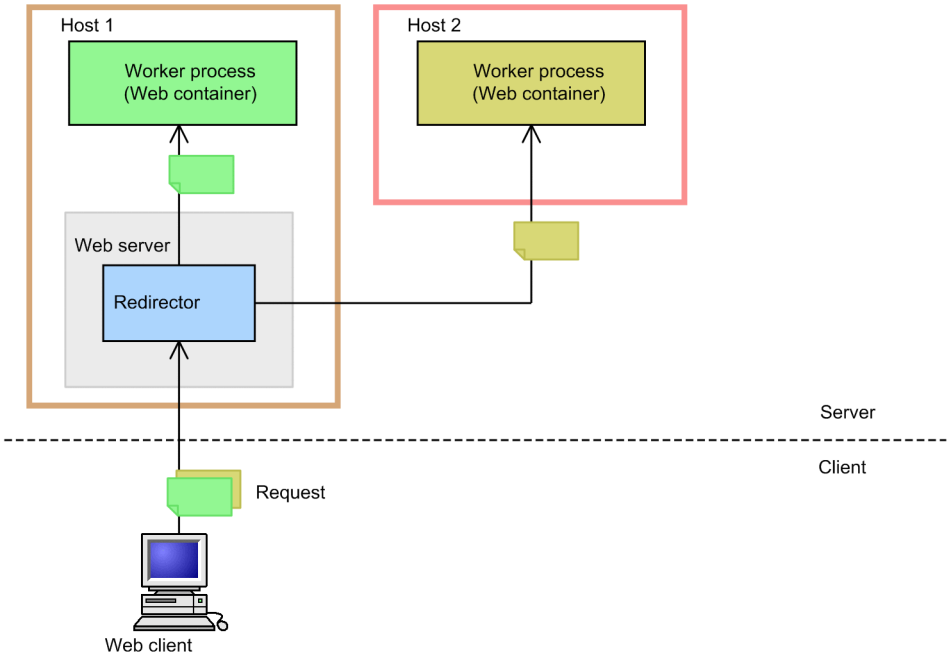


Figure 4-2: Transfer from one Web server to multiple worker processes



To distribute the requests to multiple Web containers, define the worker processes of multiple Web containers as the distribution destinations, in the redirector registered in the Web server.

(2) Request distribution method

The methods to distribute requests with the redirector include:

- **Distributing by URL pattern**
  Use this method when you want to execute a specific processing in a single Web container, and when you want to distribute a process to multiple Web containers.
  You can use this method when there is one worker process, as well as when there are multiple worker processes.
- **Distributing in round-robin format with a load balancer**
  Use this method when you want to distribute a process to multiple Web containers.
- **Distributing by the POST data size**
  Use this method when you want to distribute a process to multiple Web containers. You can specify this distribution method only when the Web server used is Cosminexus HTTP Server.
  Note that you cannot use this distribution method when the following functionality are used:
  - Session failover functionality
  - Distributing requests by the round robin format

To create a worker process, define the following attributes in a file (`workers.properties`) called the *worker definition file*:

- Worker name
- Worker type
- Host name or IP address of the Web server on which the worker is running
- Port number received by the worker

The following workers are already defined in a standard `workers.properties` file. When the Web server and the Web container are operated on the same host, you need not change these parameters.

Worker attributes	Parameter
Worker name	worker1
Worker type	ajp13
Host name	Localhost
Port number	8007

For details on how to define a worker process, see *9.5 workers.properties (Worker definition file)* in the *uCosminexus Application Server Definition Reference Guide*.

4.2.2 User-defined file for setting the request distribution method (When the Smart Composer functionality is used)

To distribute requests, edit the following user-defined files in a text editor and specify the worker, mapping between the URL pattern and worker, and the redirector operations.

- **Easy Setup definition file**

  Specify the worker definition, the worker-wise parameters, and the mapping between the URL pattern and workers. Use this file to set up request distribution by URL pattern.
- `workers.properties` **(Worker definition file)**

  Specify the worker definition and the worker-wise parameters. Use this file to set up request distribution by the round robin format and the request distribution by the POST data size.
- `mod_jk.conf` **(Redirector action definition file)**

  Specify the mapping between the URL pattern and workers and specify the redirector operations, such as which URL patterns will be forwarded to the Web container with the requests sent to Cosminexus HTTP Server. Use this file to set up request distribution by the round robin format and the request distribution by the POST data size.

For details on the Easy Setup definition file, see *4.6 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*. For details on the Worker definition file, see *9.5 workers.properties (Worker definition file)* in the *uCosminexus Application Server Definition Reference Guide*. For details on the Redirector Operation definition file, see *9.2 isapi_redirect.conf (Redirector Operation definition file for Microsoft IIS)* in the *uCosminexus Application Server Definition Reference Guide*.

4.2.3 User-defined file for setting the request distribution method (When the Smart Composer functionality is not used)

To distribute requests, edit the following user-defined files in a text editor and specify workers, mapping between the URL pattern and workers, and the redirector operations.

The files to be setup depend on the used Web server. The common user-defined files and the user-defined files for each Web server are separately described here. For details on the user-defined file `httpsd.conf`, see the *uCosminexus Application Server HTTP Server User Guide*. For details on the other user-defined files, see the *uCosminexus Application Server Definition Reference Guide*

(1) Common user-defined files

The common user-defined files for Cosminexus HTTP Server and Microsoft IIS are as follows:

- `workers.properties` **(worker definition file)**

  Specify the worker definitions and the parameters for each worker.

  The storage location of this file is as follows:

  - In Windows

    Cosminexus-installation-directory`\CC\web\redirector\workers.properties`
  - In UNIX

/opt/Cosminexus/CC/web/redirector/workers.properties

- usrconf.properties **(user property file)**
  Set the communication timeout when the Web container receives a request from the redirector.
  The storage location of this file is as follows:
  - In Windows
    Cosminexus-installation-directory\CC\server\usrconf\ejbs*server-name*\usrconf.properties
  - In UNIX
    /opt/Cosminexus/CC/server/usrconf/ejb/server-name/usrconf.properties

(2) User-defined files for Cosminexus HTTP Server

The user-defined files for Cosminexus HTTP Server are as follows:

- mod_jk.conf **(redirector action definition file)**
  Specify the redirector operations for Cosminexus HTTP Server.
  The storage location of this file is as follows:
  - In Windows
    Cosminexus-installation-directory\CC\web\redirector\mod_jk.conf
  - In UNIX
    /opt/Cosminexus/CC/web/redirector/mod_jk.conf
- httpsd.conf **(Cosminexus HTTP Server Definition file)**
  Specify the directive (parameter that defines the execution environment of the Web server) for defining the operating environment of **Cosminexus HTTP Server**.
  The storage location of this file is as follows:
  - In Windows
    Cosminexus-installation-directory\httpsd\conf\httpsd.conf
  - In UNIX
    /opt/hitachi/httpsd/conf/httpsd.conf

(3) User-defined files for Microsoft IIS

The user-defined files for Microsoft IIS are as follows:

- uriworkermap.properties **(mapping definition file)**
  Specifies the mapping between the URL pattern and worker in Microsoft IIS.
  The storage location of this file is as follows:
  Cosminexus-installation-directory\CC\web\redirector\uriworkermap.properties
- isapi_redirect.conf **(redirector action definition file)**
  Specify the redirector operations for Microsoft IIS.
  The storage location of this file is as follows:
  Cosminexus-installation-directory\CC\web\redirector\isapi_redirect.conf

(4) Notes

The followings are the notes related to the user-defined files:

- During the overwrite installation, the user-defined file is not overwritten.
- For the processing of the worker definition file and the Web server definition file when performing the upgrade installation, see *10. Migrating from Application Server of an Old Version* in the *uCosminexus Application Server Maintenance and Migration Guide*.
- The maximum number of characters in one line of the redirector definition file is 1,023. Specify settings within this character count.

- In the following user-defined files, if several parameters are specified with the same name, the value of the parameter specified first is used for operations:
  - `isapi_redirect.conf`
  - `workers.properties`
  - `uriworkermap.properties`

4.2.4 Points to be considered during Web server integration

This subsection explains the points to be considered when integrating with the Web server.

(1) Upper limit value of the request headers and response headers that can be sent and received by the Web container

When integrating with the Web server, an upper limit is set on the size of the request headers and the response headers that can be sent and received by the Web container. The respective upper-limit value is 16 KB. Note that you cannot send and receive the headers exceeding 16 KB.

(2) Points to be considered when using Cosminexus HTTP Server

In a system built by the application server, you cannot use the virtual host functionality of Cosminexus HTTP Server. If you want to invoke multiple Cosminexus HTTP Servers on the same host, use Management Server.

(3) Points to be considered when using the Microsoft IIS

The points to be considered when using the Microsoft IIS are explained below:

- When multiple Web sites are built by Microsoft IIS, you cannot integrate simultaneously with these Web sites. When you are building multiple Web sites, set a redirector in each Web site.
- Change the URL information for the requests transferred to the Web container with the redirector for Microsoft IIS. Use the changed request URL in the ISAPI filter.

  Consequently, you cannot acquire the request URL received first by the Microsoft IIS with the ISAPI filter executed after the redirector for Microsoft IIS. If you want to acquire the request URL received by Microsoft IIS with the ISAPI filter, you need to set a higher priority order for the ISAPI filter as compared to the redirector for Microsoft IIS. Note that when you need to change the priority order of the redirector for Microsoft IIS to `Medium` or `Low` in order to adjust the priority order of the ISAPI filter, specify the priority order with the `filter_priority` key of the action definition file of the redirector for Microsoft IIS. For the `filter_priority` key, see *9.2 isapi_redirect.conf (Redirector Operation definition file for Microsoft IIS)* in the *uCosminexus Application Server Definition Reference Guide*.
- When integrating with Microsoft IIS, even if you specify the following HTTP request headers in the Web client, you cannot receive these request headers in a Web application:
  - `tomcaturl`
  - `tomcatquery`
  - `tomcatworker`
  - `tomcattranslate`

  These HTTP request headers are used in the redirector.
- When integrating with Microsoft IIS, you cannot specify settings for distributing requests by the POST data size.

4.3 Distributing requests by URL pattern

This section explains the distribution of requests by the URL patterns.

You can distribute the requests by the URL patterns included in an HTTP request. Consequently, only a specific processing can be executed in the Web container, and a processing can be distributed to multiple Web containers depending on the contents of the processing.

The following table describes the organization of this section.

Table 4-4: Organization of this section (Distributing requests by the URL pattern)

Category	Title	Reference
Description	Overview of distributing requests by URL pattern	*4.3.1*
	Types of URL patterns and priority of applicable patterns	*4.3.2*
Settings	Execution environment settings (When the Smart Composer functionality is used)	*4.3.3*
	Execution environment settings (When the Smart Composer functionality is not used)	*4.3.4*

Note:

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

4.3.1 Overview of distributing requests by URL pattern

Define the requests transferred to the Web container according to the mapping between the URL pattern and the worker process. The redirector can switch the Web containers that transfer the requests, depending on the set URL pattern.

For example, you can define 'Process the HTTP request containing the URL "/examples/" in a Web container', and 'Process the HTTP request containing the URL "/examples1/" in Web container A, and the HTTP request containing the URL "/examples2/" in Web container B'.

The following figures show the examples of request distribution with the redirector:

Figure 4-3: Request distribution with the redirector (when distributing and transferring specific requests to the Web container)

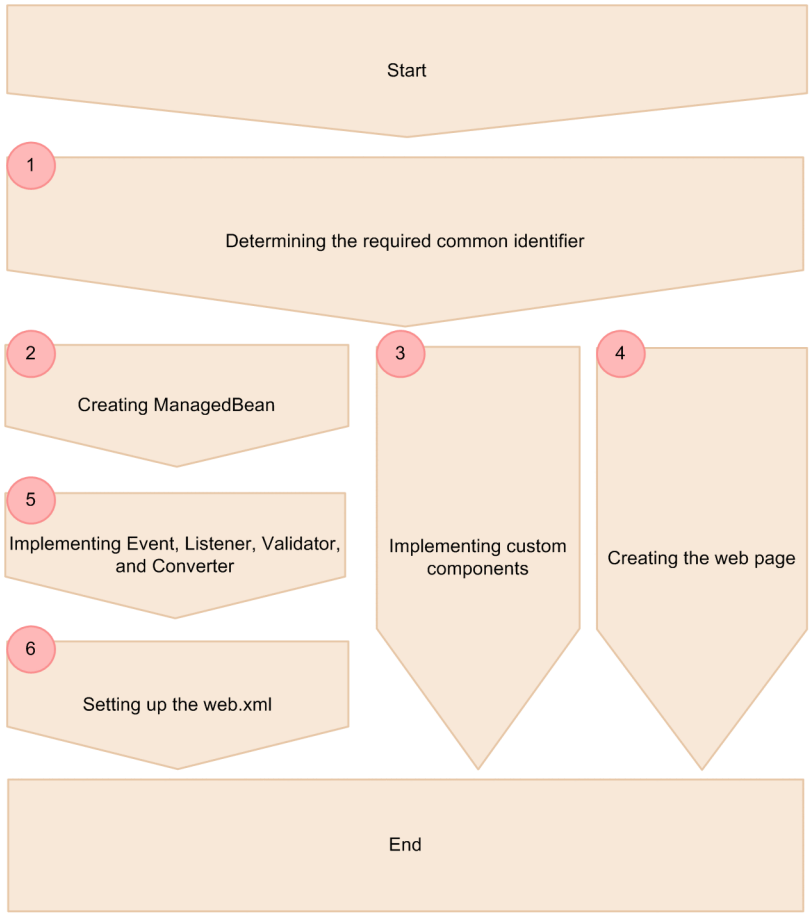
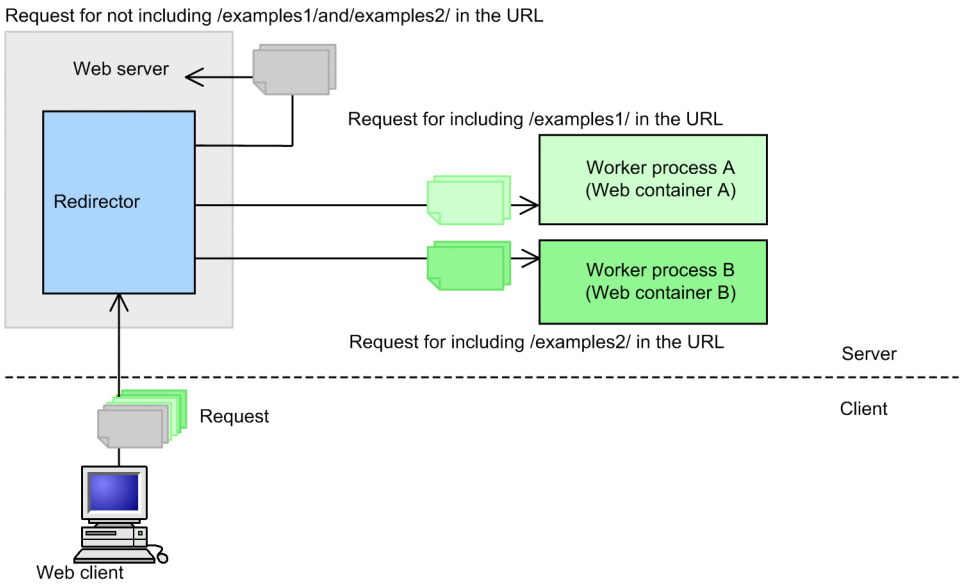


Figure 4-4: Request distribution with the redirector (when distributing and transferring requests to multiple Web containers)



4.3.2 Types of URL patterns and priority of applicable patterns

This section describes the types of URL patterns that you can specify for URL mapping of the Redirector and the priority of applicable patterns.

(1) Types of URL patterns

You can specify the following four types of URL patterns for the URL mapping of the redirector:

- **Complete path specification**

  This is a completely matching pattern.

  URL format:

  /path

  When specifying only the route, use "/".

  Characters you can specify in */path*:

  Specify a string having at least one of the following characters:

  Single-byte alphanumeric characters, "/", "*", "-", ".", "_", "~", "!", "$", "&", "'", "(", ")", "+", ",", "=", ":", "@"

  Example:

  When the URL pattern is "/examples/jsp/index.jsp", and the URL is "/examples/jsp/index.jsp", it indicates a match.

- **Path specification**

  In this pattern, the paths are matching.

  URLformat:

  */path/**

  When specifying all requests, use "/*".

  Characters you can specify in */path*:

  Specify a string having at least one of the following characters:

  Single-byte alphanumeric characters, "/", "*", "-", ".", "_", "~", "!", "$", "&", "'", "(", ")", "+", ",", "=", ":", "@"

  Example:

  When the URL pattern is "/examples/*", and the URL is "/examples/jsp/index.jsp", it indicates a match.

- **Extension specification**

  In this pattern, the extensions are matching. This pattern is applicable to all the hierarchies below the specified path.

  URL format:

  /path/.extension*

  When specifying all the paths, use "/**extension*".

  Characters you can specify in *path* and *extension*:

  Specify a string having at least one of the following characters:

  Single-byte alphanumeric characters, "/", "*", "-", ".", "_", "~", "!", "$", "&", "'", "(", ")", "+", ",", "=", ":", "@"

  Example:

  When the URL pattern is "/examples/*.jsp", and the URL is "/examples/jsp/index.jsp", it indicates a match.

- **Suffix specification**

  In this pattern, suffixes are matching. This pattern is applicable to all the hierarchies below the specified path.

  URL format:

  */path/*suffix*

  When specifying all the paths, use "/*<*suffix*>".

  Characters you can specify in *path* and *suffix*:

  Specify a string having at least one of the following characters:

Single byte alphanumeric characters, "/", "*", "-", ".", "_", "~", "!", "$", "&", "'", "(", ")", "+", ",", "=", ":", "@"

Example:

When the URL pattern is "/examples/servlet/*Servlet", and the URL is "/examples/servlet/HelloServlet", it indicates a match.

! Important note

The following are the notes on specifying a URL pattern:

- A URL pattern must not begin with anything but "/". In Windows, if you specify a character other than "/", the KDJE41012-E message is output and the mapping is ignored. In any other OS, a message is displayed and Cosminexus HTTP Server fails to start.
- A "*", when used as a wildcard, cannot be specified before the "/*" in a URL pattern. If you specify anything other than "/" just before the first "*" in a URL pattern, the URL pattern is treated as a "Complete path specification" and "/*" is not treated as a wildcard even if it is a part of the URL.
- Do not describe multiple mappings of the same URL pattern. The behavior in case you specify multiple mappings, is as follows.

  The former URL pattern mapping is used in the "Complete path specification" and the "Path specification". The latter mapping is used in the case of "Extension specification" and "Suffix specification".
- You must use only valid values in *path*, *extension*, and *suffix*. If you use an invalid character in a URL pattern, some types of characters might not be forwarded to the Web container.
- The string length of *extension* or *suffix* must be at least one character long. If the length is shorter than one character, the "Extension specification" outputs the KDJE41041-W message, and the mapping is ignored. In the Suffix specification, the value that you specify is treated as a URL pattern of the "Path specification".

---

(2) Priority of applicable patterns

Among the mapping to these four URL patterns, the URL pattern with the highest priority is 'Complete path specification'. When the URL does not match with 'Complete path specification', path matching is judged in the following order, and the applicable URL pattern is decided:

1. When the URL does not match with 'Complete path specification'

   The longest matching URL pattern from among 'Path specification', 'Extension specification', and 'Suffix specification' is applied. Longest match refers to the longest matching URL from the beginning ("/") until the high order path of "*".

   The URL in which the following two mappings are defined is illustrated below as an example:

   Mapping definition:

   /examples/* worker1

   /examples/jsp/* worker2

   In this case, when the URL is "/examples/jsp/index.jsp", the mapping of worker2 is applied, and when the URL is "/examples/test/index.jsp", the mapping of worker1 is applied.

2. In addition to the conditions of 1., when multiple longest matching 'Path specification', 'Extension specification', and 'Suffix specification' URL patterns are present

   'Extension specification' or 'Suffix specification' is given priority over 'Path specification'.

   The URL in which the following two mappings are defined is illustrated below as an example:

   Mapping definition:

   /examples/jsp/* worker1

   /examples/jsp/*.jsp worker2

   In this case, when the URL is "/examples/jsp/index.jsp", the mapping of worker2 is applied, and when the URL is "/examples/jsp/test.html", the mapping of worker1 is applied.

3. In addition to the conditions of 1. and 2., when multiple longest matching 'Extension specification 'and 'Suffix specification' URL patterns are present

   The URL pattern specified later is given priority.

   The URL in which the following two mappings are defined is illustrated below as an example:

Mapping definition:

/examples/*.jsp worker1

/examples/*jsp worker2

When URL is specified in this order, the mapping of worker2 is applied when the URL is "/examples/jsp/index.jsp".

! Important note

You must note the following points when judging the priority of applicable patterns:

- If a request URL includes a query (string after a "?" mark in the URL), the query part is not used when comparing with the URL pattern.

  Example:

  If the request URL is "/examples/jsp/index.jsp?query=foo", the URL used for comparison is "/examples/jsp/index.jsp".
- If a request URL includes a parameter (string from a semicolon (;)), the parameter part is not used when comparing with the URL pattern.

  Example:

  If the request URL is "/examples/jsp/index.jsp;jsessionid=0000", the URL used for comparison is "/examples/jsp/index.jsp".
- A request URL path is first normalized and then the URL is compared with the URL pattern to judge whether both the URLs match.

  Example:

  If a request URL is "/examples/../examples/./jsp//index.jsp", the URL used for comparison is "/examples/jsp/index.jsp".
- A URL pattern is never normalized. Therefore, a URL pattern that includes "./" or "../" does not match with the request URL.
- In Windows, an extension of a URL pattern specified in the "Extension specification" is not case sensitive.

4.3.3 Execution environment settings (When the Smart Composer functionality is used)

Distributing requests by the URL patterns included in the HTTP requests enables you to execute only the specific processes in the Web container and to distribute requests to multiple Web containers according to the processing contents. Note that, when you use 'Distributing requests by URL pattern' method, as a principle, the requests are distributed in the Web application. Define the URL pattern as an operation of the redirector.

(1) Setup procedure

To set the distribution of requests by the URL pattern:

1. Define the workers and mapping between the URL pattern and workers in the Easy Setup definition file.

   Specify the list of worker names, worker types (set `ajp13`), port number, and host name.

   If mapping is already defined, delete or replace the definition.
2. Set up the Web server environment and restart the Web server.

   For details on the Web server settings, see *Appendix B Precautions related to Cosminexus HTTP Server Settings*.

(2) Settings in the Easy Setup definition file

Specify the definition for distributing requests by the URL pattern in the `<configuration>` tag of the logical Web server (web-server) in the Easy Setup definition file.

The following table lists the definitions in the Easy Setup definition file for distributing requests by the URL pattern:

Table 4-5: Definitions in the Easy Setup definition file for distributing requests by the URL pattern

Category	Parameter	Description
Worker definitions	`worker.list`	Specifies a list of one or multiple worker names.
	`worker.`*worker-name*`.host`	Specifies the worker host name or IP address.
	`worker.`*worker-name*`.port`	Specifies the worker port number.
	`worker.`*worker-name*`.type`	Specifies the worker type (`ajp13`, `lb`, or `post_size_lb`).
	`Worker.`*worker-name*`.cachesize`	Specifies the number of worker connections that are reused in the redirector. This parameter can only be specified in Windows.
	`worker.`*worker-name*`.receive_timeout`	Specifies the communication timeout value.
	`worker.`*worker-name*`.delegate_error_code`	Specifies the error status code used when the creation of the error page is entrusted to the Web server.
Definition of mapping between the URL pattern and worker	`JkMount`	Specifies some combination of workers specified in the URL pattern and `worker.list`.

Note:

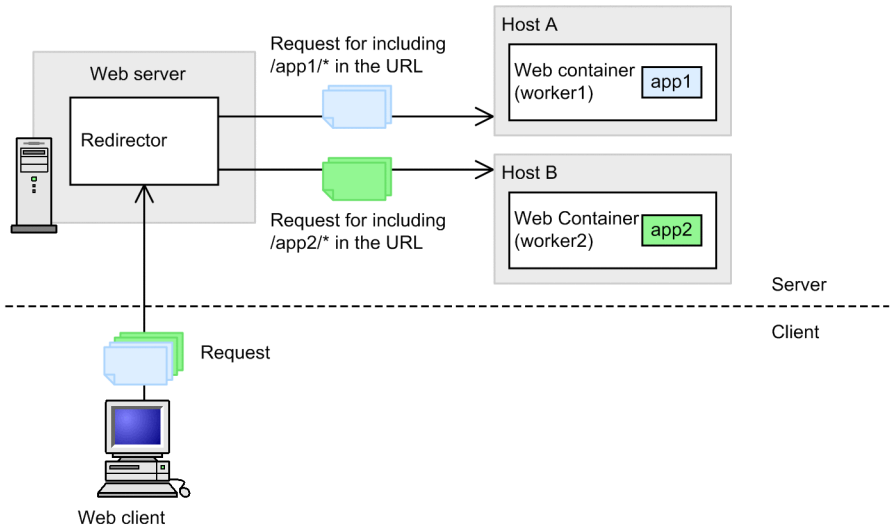
Define the name of the workers specified in the `worker.list` parameter in *worker-name*.

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

(3) Example settings

The following figure shows the distribution of requests by the URL patterns.

Figure 4-5: Example of distribution of requests by the URL patterns



In this example, the Web application `app1` is deployed on host A and the Web application `app2` is deployed on host B. By including the name of the Web application you want to process in the URL pattern of the request, the request with the URL pattern `/app1/*` can be processed on the host A and the request with the URL pattern `/app2/*` can be processed on the host B. The worker name of the host A is `worker1` and the worker name of the host B is `worker2`.

An example of the Easy Setup definition file is described below. To distribute the requests to multiple Web containers, specify the worker processes of multiple Web containers as the distribution destinations, in the redirector registered in the Web server. Also, the URL pattern `/app1/*` are mapped with `worker1` and the URL pattern `/app2/*` are mapped with `worker2`.

Note that to implement this configuration, you must distribute requests to multiple Web systems.

Example of Easy Setup definition file

```
...
<param>
 <param-name>JkMount</param-name>
 <param-value>/app1/* worker1</param-value>
 <param-value>/app2/* worker2</param-value>
</param>
<param>
 <param-name>worker.list</param-name>
 <param-value>worker1, worker2</param-value>
</param>
<param>
 <param-name>worker.worker1.port</param-name>
 <param-value>8007</param-value>
</param>
<param>
 <param-name>worker.worker1.host</param-name>
 <param-value>hostA</param-value>
</param>
<param>
 <param-name>worker.worker1.type</param-name>
 <param-value>ajp13</param-value>
</param>
<param>
 <param-name>worker.worker1.cachesize</param-name>
 <param-value>64</param-value>
</param>
<param>
 <param-name>worker.worker2.port</param-name>
 <param-value>8007</param-value>
</param>
<param>
 <param-name>worker.worker2.host</param-name>
 <param-value>hostB</param-value>
</param>
<param>
 <param-name>worker.worker2.type</param-name>
 <param-value>ajp13</param-value>
</param>
<param>
 <param-name>worker.worker2.cachesize</param-name>
 <param-value>64</param-value>
</param>
...
```

4.3.4 Execution environment settings (When the Smart Composer functionality is not used)

Distributing requests by the URL patterns included in the HTTP requests enables you to execute only the specific processes in the Web container and to distribute requests to multiple Web containers according to the processing contents. Note that, when you use 'Distributing requests by URL pattern' method, as a principle, the requests are distributed in the Web application. Define the URL pattern as an operation of the redirector.

(1) Setup procedure

To set the distribution of requests by the URL pattern:

1. **Define the worker in workers.properties.**
   Specify the list of worker names, worker types (set `ajp13`), port number, and host name.

The default value is defined in `workers.properties` that is provided by default. To use the default definition defined as a comment, delete the hash mark (#) at the beginning of the applicable line.

For details on `workers.properties` (worker definition file), see *9.5 workers.properties (worker definition file)* in the *uCosminexus Application Server Definition Reference Guide*.

2. When using Cosminexus HTTP Server, define the mapping between the URL pattern and worker in `mod_jk.conf`. When using Microsoft IIS, define the mapping between the URL pattern and worker in uriworkermap.properties.

   If mapping is already defined, delete or replace the definition.

   For details on `mod_jk.conf` (redirector operation definition file for HTTP Server), see *9.3 mod_jk.conf (redirector operation definition file for HTTP Server)* in the *uCosminexus Application Server Definition Reference Guide*.

   For details on `uriworkermap.properties` (mapping definition file for Microsoft IIS), see *9.4 uriworkermap.properties (mapping definition file for Microsoft IIS)* in the *uCosminexus Application Server Definition Reference Guide*.

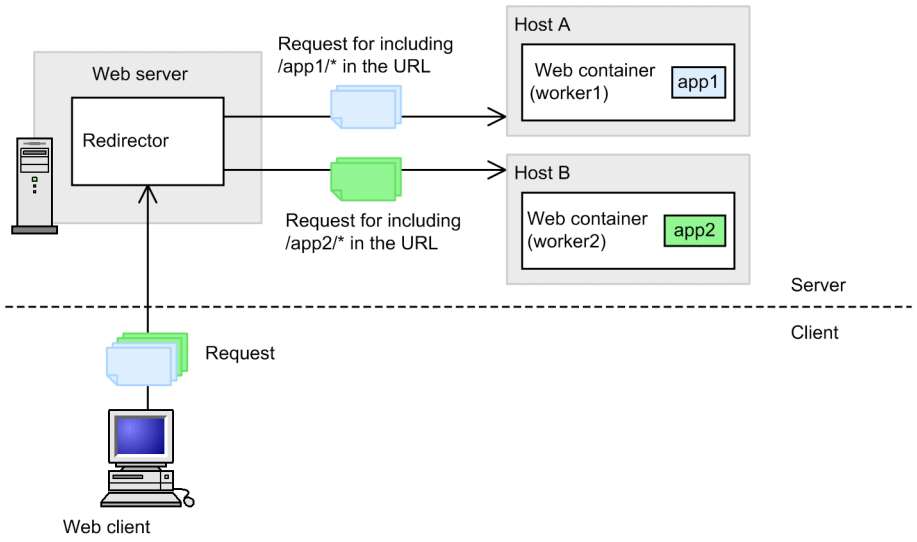
3. Set up the Web server environment and restart the Web server.

   For details on the Web server settings, see *Appendix B Precautions related to Cosminexus HTTP Server Settings* or *Appendix C Microsoft IIS Settings*.

(2) Example settings

The following figure shows the distribution of requests by the URL patterns.

Figure 4-6: Example of distribution of requests by the URL patterns



In this example, the Web application `app1` is deployed on host A and the Web application `app2` is deployed on host B. By including the name of the Web application you want to process in the URL pattern of the request, the request with the URL pattern `/app1/*` can be processed on the host A and the request with the URL pattern `/app2/*` can be processed on the host B. The worker name of host A is `worker1` and the worker name of host B is `worker2`.

An example of the `workers.properties` file is shown below. To distribute the requests to multiple Web containers, specify the worker processes of multiple Web containers as the distribution destinations, in the redirector registered in the Web server.

Example of workers.properties (In Windows)

```
worker.list=worker1, worker2

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13
worker.worker1.cachesize=64

worker.worker2.port=8007
```

```
worker.worker2.host=hostB
worker.worker2.type=ajp13
worker.worker2.cachesize=64
```

Example of workers.properties (In UNIX)

```
worker.list=worker1, worker2

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13

worker.worker2.port=8007
worker.worker2.host=hostB
worker.worker2.type=ajp13
```

Examples of the `mod_jk.conf` and `uriworkermap.properties` files are shown below. Here, the URL pattern `/app1/*` are mapped with `worker1` and the URL pattern `/app2/*` are mapped with `worker2`.

Example of mod_jk.conf (In Cosminexus HTTP Server)

```
JkMount /app1/* worker1
JkMount /app2/* worker2
```

Example of uriworkermap.properties (In Microsoft IIS)

```
/app1/*=worker1
/app2/*=worker2
```

4.4 Distributing requests by the round-robin format

This section explains the distribution of requests by the round-robin format.

The following table describes the organization of this section.

Table 4-6: Organization of this section (Distributing requests by the round-robin format)

Category	Title	Reference
Description	Overview of distributing requests by the round-robin format	*4.4.1*
	Examples of Request Distribution in the Round-robin Format	*4.4.2*
	Defining request distribution in the round robin format	*4.4.3*
Settings	Execution environment settings (When the Smart Composer functionality is used)	*4.4.4*
	Execution environment settings (When the Smart Composer functionality is not used)	*4.4.5*
Notes	Precautions related to request distribution in the round-robin format	*4.4.6*

Note:
There is no specific description of *Implementation* and *Operations* for this functionality.

When Web containers are deployed with a cluster configuration, by using the redirector, requests are distributed in round-robin format to the respective Web containers. By referencing the session ID appended to each request, the redirector distributes the requests so that the requests from the same Web client are always transferred to the same Web container.

If the processing efficiency of the Web containers to which requests are distributed is different, you can adjust the proportion of load on each host by defining load parameters. When distributing requests in the round-robin format, as a prerequisite, you must deploy the same Web application on each Web container performing the distribution processing.

4.4.1 Overview of distributing requests by the round-robin format

For distribution of requests in the round-robin format with a cluster configuration, use a worker definition called *load balancer*. The list of worker processes that act as the distribution destinations is defined in the load balancer. Based on this definition, requests are distributed to the worker processes in the round-robin format.

An HTTP request is distributed. The HTTP requests belonging to the same session, however, are distributed to the same worker as the previous distribution destination.

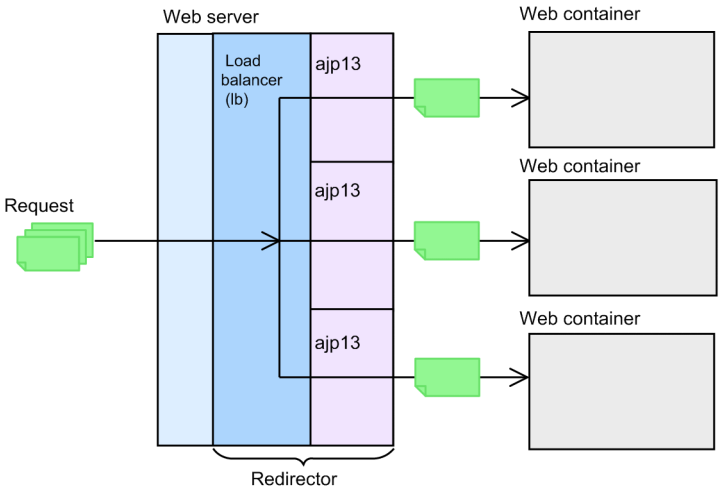
Reference note

During the Web server integration, if you specify request distribution with the round-robin format in the redirector settings, the worker name will be added in the session ID of `HttpSession`. Also, regardless of whether the settings for adding the server ID are specified, the server ID will not be added in the session ID of `HttpSession`.

4.4.2 Examples of request distribution in the round-robin format

The following figure shows an example of request distribution with the load balancer.

Figure 4-7: Example of request distribution with the load balancer



4.4.3 Defining request distribution in the round robin format

The following load balancer is already defined in a standard `workers.properties` file.

```
#worker.list=loadbalancer1

#worker.loadbalancer1.type=lb
#worker.loadbalancer1.balanced_workers=worker1, worker2
```

Set the type of the worker in worker.loadbalancer1.type. Set the name of the worker process to which the request is to be distributed in `worker.loadbalancer1.balanced_workers`. In `workers.properties`, `lb` and `worker1, worker2` are defined respectively as loadbalancer1.

This definition is, however, described as a comment. Consequently, when using the above-mentioned load balancer, delete the '#' (hash mark) present at the beginning of the corresponding rows of `workers.properties`.

You can define the ratio of request distribution in `lbfactor` parameter of each worker definition that is the distribution target. Larger the `lbfactor`, larger is the ratio of requests transferred to the worker process.

For example, when requests are distributed to two worker processes, namely worker1 and worker2, the ratio of request distribution is defined as follows in the `lbfactor` parameter of the worker definition:

- `lbfactor` parameter of worker1: 2.0
- `lbfactor` parameter of worker2: 1.0

In this case, worker1 is in charge of double the number of Web clients of worker2.

4.4.4 Execution environment settings (When the Smart Composer functionality is used)

By defining the list of workers that act as the distribution destinations in the load balancer, requests are distributed to the workers in the round-robin format.

If you set a load balancing value in each worker that acts as the distribution destination and define the request distribution ratio, you can adjust the proportion of load on each host. Since the redirector distributes requests with the round-robin format for each HTTP request at this ratio, higher the ratio for a worker the greater will be the proportion of forwarded requests. However, the HTTP requests belonging to the same session are distributed to the same worker as the last time.

(1) Setup procedure

To set the distribution of requests by the round-robin format, specify the settings using the following procedure:

1. Define the load balancer and worker in workers.properties.

   Definitions for the load balancer

   Specify the list of worker names, worker types (specify `lb`), and list of workers for load balancing.

   Definitions for each worker

   Specify the worker types (specify `ajp13`), port number, host name, and the load balancing value.

   The default value is defined in `workers.properties` that is provided by default. To use the default definition defined as a comment, delete the hash mark (#) at the beginning of the applicable line.

   For details on `workers.properties` (worker definition file), see *9.5 workers.properties (worker definition file)* in the *uCosminexus Application Server Definition Reference Guide*.

2. Define the mapping between the URL pattern and worker in mod_jk.conf.

   If mapping is already defined, delete or replace the definition.

   For details on `mod_jk.conf` (redirector operation definition file for HTTP Server), see *9.3 mod_jk.conf (redirector operation definition file for HTTP Server)* in the *uCosminexus Application Server Definition Reference Guide*.

3. Set up the Web server environment and restart the Web server.

   For details on the Web server settings, see *Appendix B Precautions related to Cosminexus HTTP Server Settings*.

Notes

- When you use load balancing with the redirector and if a failure is detected in a worker, the worker is excluded from the choices for redirect destination workers for 60 seconds from the detection of failure. Therefore, even if the failure is recovered, the worker might not be used for a maximum of 60 seconds.
- In UNIX, when the server processes of Cosminexus HTTP Server are generated or destroyed according to the load, multiple server processes are allocated by the worker defined first in `workers.properties`. Also, even if the number of server processes is fixed, the server process to which the request is allocated is uncertain, and therefore, if you specify the same load balancing value, round-robin might not occur. Unless destroyed, the server processes are allowed to increase according to the load, and therefore you must specify a directive in such a way so that the server processes are not generated or destroyed in a short time.

  Specify the `httpsd.conf` directive of Cosminexus HTTP Server in such a way so that the following conditions are fulfilled:

Conditions	Meaning
MaxSpareServers ≧ MaxClients	The server processes increase up to MaxClients and stay resident even after the processing ends.
MaxRequestsPerChild 10000	The HTTP request is processed 10,000 times and then the server process is terminated to refresh the operations (10,000 is the recommended value). Specify an adequately large value for the number of J2EE servers that act as the distribution destinations.
StartServers = MaxClients	You specify this condition to start all the server processes first.

Example specification of directive

```
StartServers 256
MaxClients 256
MaxSpareServers 256
MaxRequestsPerChild 10000
```

(2) Settings in workers.properties and mod_jk.conf

Define the settings for distributing requests by the round-robin format in `workers.properties` and `mod_jk.conf`. The following table lists the keys defined in `workers.properties and mod_jk.conf`:

Table 4-7: Keys defined in workers.properties and mod_jk.conf (When distributing requests by the round-robin format)

Types of files	Key name	Description
workers.properties	worker.list	Specifies a list of one or multiple worker names.
	worker.*worker-name*.host	Specifies the worker host name or IP address.
	worker.*worker-name*.port	Specifies the worker port number.
	worker.*worker-name*.type	Specifies the worker type. Specify lb in the load balancer and ajp13 in the worker for load balancing.
	worker.*worker-name*.balanced_workers	Specifies the list of workers for load balancing.
	worker.*worker-name*.lbfactor	Specifies the load balancing value.
	worker.*worker-name*.cachesize	Specifies the number of worker connections that are reused in the redirector. This key can only be specified in Windows.
	worker.*worker-name*.receive_timeout	Specifies the communication timeout value.
	worker.*worker-name*.delegate_error_code	Specifies the error status code used when the creation of the error page is entrusted to the Web server.
mod_jk.conf	JkMount	Specifies some combination of workers specified in the URL pattern and worker.list.

Note:

In worker-name, define the worker name specified in the worker.list key or the worker.*worker-name*.balanced_workers key.

The following table lists the parameters that you can specify for each worker type:

Table 4-8: Keys that can be specified for each worker type

Key name	Worker type (value specified in worker.*worker-name*.type)	
	Load balancer (Specify lb)	Worker (Specify ajp13)
worker.*worker-name*.host	--	Y
worker.*worker-name*.port	--	Y
worker.*worker-name*.type	Y	Y
worker.*worker-name*.balanced_workers	Y	--
worker.*worker-name*.lbfactor	--	O
worker.*worker-name*.cachesize	--	O
worker.*worker-name*.receive_timeout	--	O
worker.*worker-name*.delegate_error_code	--	O

Legend:

Y: Can be specified

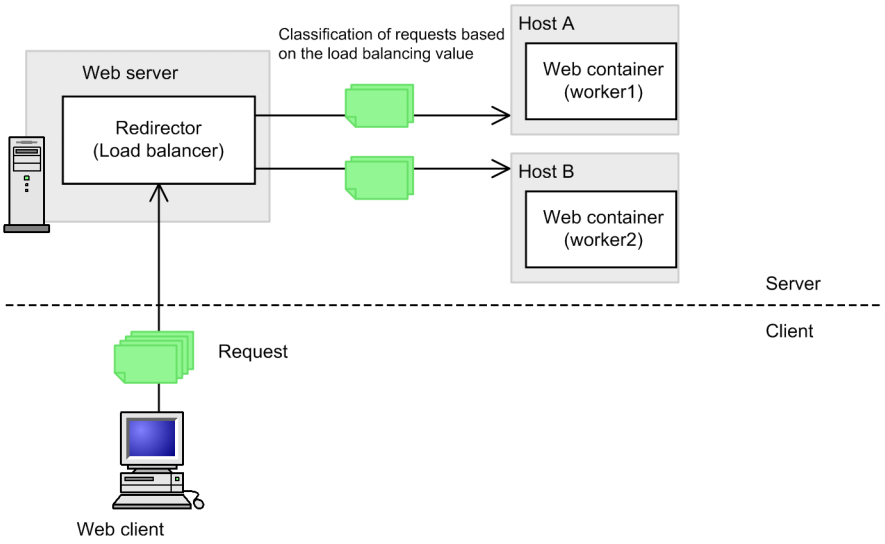
--: Cannot be specified

O: Can be optionally specified

(3) Example settings

The following figure shows the distribution of requests by the round-robin format.

Figure 4-8: Example of distribution of requests by the round-robin format



In this example, the requests under `/examples` are equally distributed on host A and on host B. The worker name in host A is `worker1` and the worker name in host B is `worker2`.

An example of the `workers.properties` file is shown below. The load balancer and worker are defined here. Since the requests are distributed at an equal rate, `1` is specified as the load balancing value for both `worker1` and `worker2`.

Example of workers.properties (In Windows)

```
worker.list=loadbalancer1

worker.loadbalancer1.balanced_workers=worker1, worker2
worker.loadbalancer1.type=lb

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13
worker.worker1.cachesize=64
worker.worker1.lbfactor=1

worker.worker2.port=8007
worker.worker2.host=hostB
worker.worker2.type=ajp13
worker.worker2.cachesize=64
worker.worker2.lbfactor=1
```

Example of workers.properties (In UNIX)

```
worker.list=loadbalancer1

worker.loadbalancer1.balanced_workers=worker1, worker2
worker.loadbalancer1.type=lb

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13
worker.worker1.lbfactor=1

worker.worker2.port=8007
worker.worker2.host=hostB
worker.worker2.type=ajp13
worker.worker2.lbfactor=1
```

An example of the `mod_jk.conf` file is shown below. The load balancer name loadbalancer1 is specified here.

Example of mod_jk.conf

```
JkMount /examples/* loadbalancer1
```

4.4.5 Execution environment settings (When the Smart Composer functionality is not used)

By defining the list of workers that act as the distribution destinations in the load balancer, the requests are distributed to the workers with the round-robin format.

If you specify a load balancing value in each worker that acts as the distribution destination and define the request distribution ratio, you can adjust the proportion of load on each host. Since the redirector distributes requests with the round-robin format for each HTTP request at this ratio, higher the ratio for a worker the greater will be the proportion of forwarded requests. However, the HTTP requests belonging to the same session are distributed to the same worker of the last time.

(1) Setup procedure

To set the distribution of requests by the round-robin format, use the following procedure:

1. Define the load balancer and worker in workers.properties.

   Definitions of load balancer

   Specify the list of worker names, worker types (specify `lb`), and list of workers for load balancing.

   Definitions for each worker

   Specify the worker types (specify `ajp13`), port number, host name, and the load balancing value.

   The default value is defined in `workers.properties` that is provided as standard. To use the default definition defined as a comment, delete the hash mark (#) at the beginning of the applicable line.

   For details on `workers.properties` (worker definition file), see *9.5 workers.properties (worker definition file)* in the *uCosminexus Application Server Definition Reference Guide*.

2. When using Cosminexus HTTP Server, define the mapping between the URL pattern and worker in mod_jk.conf. When using Microsoft IIS, define the mapping between the URL pattern and worker in uriworkermap.properties.

   If a mapping is already defined, delete the definition or replace the mapping.

   For details on `mod_jk.conf` (redirector operation definition file for HTTP Server), see *9.3 mod_jk.conf (redirector operation definition file for HTTP Server)* in the *uCosminexus Application Server Definition Reference Guide*.

   For details on `uriworkermap.properties` (mapping definition file for Microsoft IIS), see *9.4 uriworkermap.properties (mapping definition file for Microsoft IIS)* in the *uCosminexus Application Server Definition Reference Guide*.

3. Set up the Web server environment and restart the Web server.

   For details on the Web server settings, see *Appendix B Precautions related to Cosminexus HTTP Server Settings* or *Appendix C Microsoft IIS Settings*.

Notes

- When you perform load balancing in redirector and if a failure is detected in a worker, that worker is excluded from the choices of the redirect destination workers for the period of 60 seconds after the failure is detected. Therefore, even if the failure is recovered, the worker might not be used for a maximum period of 60 seconds.
- When you use Microsoft IIS and specify multiple worker processes on which the redirector is running, and if two or more workers are set, many requests are allocated to the worker defined initially in `workers.properties` that is the initial redirect destination.

  Also, allocation of a request to a work process is dependent on the Microsoft IIS control and therefore, even if the same load balancing value is specified, there are cases when a round-robin is not performed.

  In such cases, by setting the number of application pools to one, a round robin can be performed even for the first redirect allocation destination.
- In UNIX, when the server processes of Cosminexus HTTP Server are generated or destroyed according to the load, more requests are allocated by the worker defined first in `workers.properties`. Also, even if the number of server processes is fixed, the server process to which the request is allocated is uncertain, and

therefore if you specify the same load balancing value, the round-robin might not occur. Unless the server process is destroyed, the server process is allowed to increase according to the load, so you must specify a directive in such a way so that the server processes are not generated or destroyed in a short time.

Specify the `httpsd.conf` directive of Cosminexus HTTP Server in such a way so that the following conditions are fulfilled:

Conditions	Meaning
MaxSpareServers ≥ MaxClients	The server processes increase up to MaxClients and stay resident even after the processing ends.
MaxRequestsPerChild 10000	The HTTP request is processed 10,000 times and then the server process is terminated to refresh the operations (10,000 is the recommended value). Specify an adequately large value for the number of J2EE servers that act as the distribution destinations.
StartServers = MaxClients	You specify this condition to start all the server processes first.

Example specification of directive

```
StartServers 256
MaxClients 256
MaxSpareServers 256
MaxRequestsPerChild 10000
```

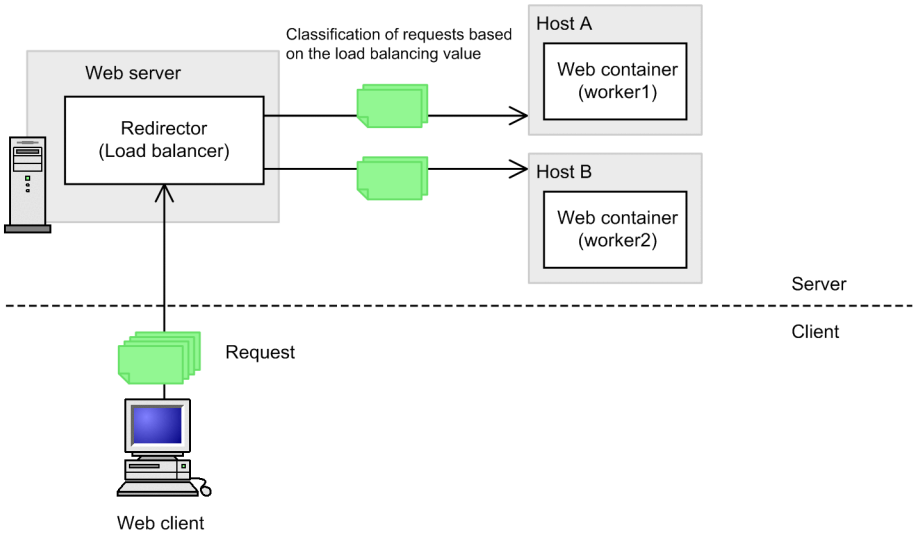
(2) Settings in workers.properties and mod_jk.conf

The settings in `workers.properties` and `mod_jk.conf` are the same settings that are specified when Smart Composer is used. For details, see *4.4.4(2) Settings in workers.properties and mod_jk.conf*.

(3) Example settings

The following figure shows the distribution of requests by the round-robin format.

Figure 4-9: Example of distribution of requests by the round robin format



In this example, the requests under `/examples` are equally distributed on host A and on host B. The worker name in host A is `worker1` and the worker name in host B is `worker2`.

An example of the `workers.properties` file is shown below. The load balancer and worker will be defined here. Since the requests are distributed at an equal rate, `1` is specified as the load balancing value for both `worker1` and `worker2`.

Example of workers.properties (In Windows)

```
worker.list=loadbalancer1

worker.loadbalancer1.balanced_workers=worker1, worker2
worker.loadbalancer1.type=lb

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13
worker.worker1.cachesize=64
worker.worker1.lbfactor=1

worker.worker2.port=8007
worker.worker2.host=hostB
worker.worker2.type=ajp13
worker.worker2.cachesize=64
worker.worker2.lbfactor=1
```

Example of workers.properties (In UNIX)

```
worker.list=loadbalancer1

worker.loadbalancer1.balanced_workers=worker1, worker2
worker.loadbalancer1.type=lb

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13
worker.worker1.lbfactor=1

worker.worker2.port=8007
worker.worker2.host=hostB
worker.worker2.type=ajp13
worker.worker2.lbfactor=1
```

Examples of the `mod_jk.conf` and `uriworkermap.properties` files are shown below. The load balancer name `loadbalancer1` will be specified here.

Example of mod_jk.conf (in Cosminexus HTTP Server)

```
JkMount /examples/* loadbalancer1
```

Example of uriworkermap.properties (In Microsoft IIS)

```
/examples/*=loadbalancer1
```

4.4.6 Precautions related to request distribution in the round-robin format

The precautions related to request distribution in the round-robin format for the redirector are as follows:

- **Sending requests to the Web container when the J2EE application is not running**
  When distributing requests with the round-robin format, requests are sent to the Web container even when the J2EE application is not running. Therefore, you must isolate all the Web containers from the system, and then implement the changes in J2EE applications.
- **Disabling of health check by the load balancer**
  When the load balancer and request distribution with the round-robin format are combined and used, requests are normally forwarded to the Web container by the redirector even if a failure occurs in the J2EE server. As a result, the failure on the J2EE server cannot be detected in the load balancer and the Web container cannot be monitored.

4.5 Distributing requests by the POST data size

This section explains the distribution of requests by the POST data size.

The following table describes the organization of this section.

Table 4-9: Organization of this section (Distributing requests by the POST data size)

Category	Title	Reference
Description	Overview of distributing requests by the POST data size	*4.5.1*
	Examples of Distributing Requests by the POST Data Size	*4.5.2*
	Request distribution conditions	*4.5.3*
	Definition for distributing requests by the POST data size	*4.5.4*
Settings	Execution environment settings (When the Smart Composer functionality is used)	*4.5.5*
	Execution environment settings (When the Smart Composer functionality is not used)	*4.5.6*

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

4.5.1 Overview of distributing requests by the POST data size

When Web containers are deployed with a cluster configuration, by using the redirector, requests are distributed by the POST data size to the respective Web containers. If you use this functionality, you can forward the very long POST data requests with a long processing time to the specific Web containers. As a result, you can avoid the decrease in the throughput of requests other than the very long POST data requests or can avoid the decrease in response time. When distributing requests in this method, as a prerequisite, you must deploy a Web application on each Web container performing the distribution processing. However, the Web applications deployed on each J2EE server is not required to be the same.

For distributing requests by the POST data size in the cluster configuration, use the worker definition called *POST request-distributing worker*. The list of worker processes that act as the distribution destinations is defined in the POST request-distributing worker. Based on this definition, requests are distributed to the worker processes by the POST data size. The worker process that acts as the distribution destination of the POST request-distributing worker is called *POST request-forwarding worker*.

The HTTP requests are distributed to the POST request-forwarding worker.

! Important note

Even when the session is managed with control based on HTTP Cookie or URL rewriting, if distribution by POST data size is specified, the request distribution destination is determined by the POST data size. Therefore, the session ID of the `HttpSession` is not inherited even when the request is from the same client.

For example, if `HttpSession` session ID is generated on J2EE server 1 and the request is forwarded to J2EE server 2, a new `HttpSession` session ID is generated on J2EE server 2. In this case, if J2EE server 1 is accessed again, the `HttpSession` session ID generated on J2EE server 2 is being used in the client, so a new `HttpSession` session ID is generated on J2EE server 1. Therefore, the session of the `HttpSession` is not inherited.

Note that when the `HttpSession` session ID is generated on J2EE server 1 and a request is forwarded to J2EE server 2, in that case if `HttpSession` session ID is not generated on J2EE server 2, the `HttpSession` session ID of J2EE server 1 will be used as it is, when you re-access the J2EE server 1.

4.5.2 Examples of distributing requests by the POST data size

For distributing requests by the POST data size, the value set as the upper limit of the POST data size will differ depending on whether the request forwarded to the POST request-distributing worker can be limited or not.

- **When the request forwarded to the POST request-distributing worker can be limited**

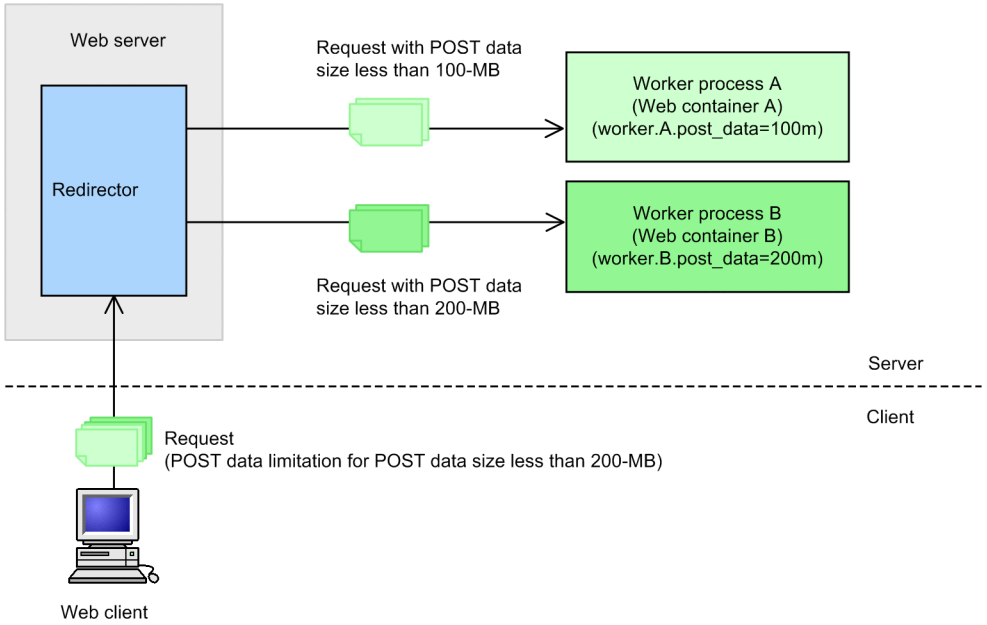
The requests fulfilling the following conditions are forwarded to the POST request-distributing worker:

- The request is a POST data request.
- The POST data size of the request is less than 200 MB.

If the request can be limited, the range of very long POST data that you want to process can also be limited. Set an upper limit for each request-forwarding worker so that a request of a specific POST data size is forwarded to a worker that is processing a very long POST data request.

The following figure shows a distribution example of requests by the POST data size when the requests can be limited:

Figure 4-10: Example of distribution of requests by the POST data size (When the request can be limited)



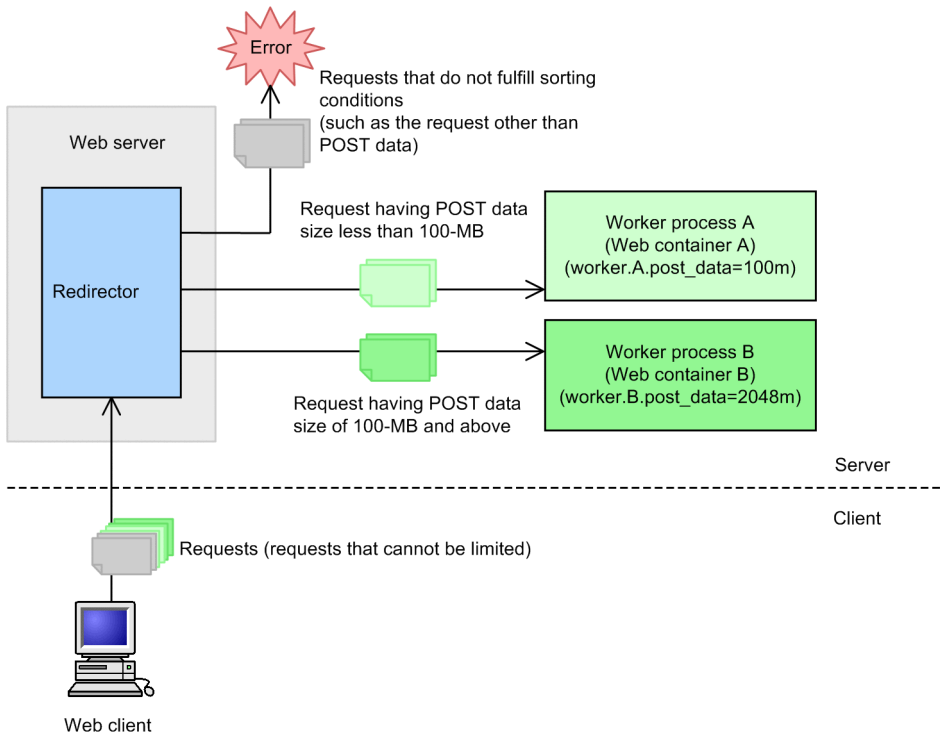
In this figure, two POST request-forwarding workers are prepared. An upper limit is set for the POST data size in such a way so that the requests with POST data size between 100 MB and 200 MB are forwarded to worker process B. If the POST data size of the request is less than the upper limit, the request is distributed to the respective POST request-forwarding worker. If the POST data size of a request is applicable to multiple POST request-forwarding workers, the request is distributed to the POST request-forwarding worker with the smallest POST data size upper limit. For example, a request with POST data size 80 MB is applicable to both workers, but is distributed to worker process A.

- **When the request forwarded to the POST request-distributing worker cannot be limited**

  If the request cannot be limited, set the maximum value as the upper limit for the POST data size of the worker that processes very long POST data.

  The following figure shows a distribution example of requests by the POST data size when the request cannot be limited:

Figure 4-11: Example of distribution of requests by the POST data size (When the request cannot be limited)



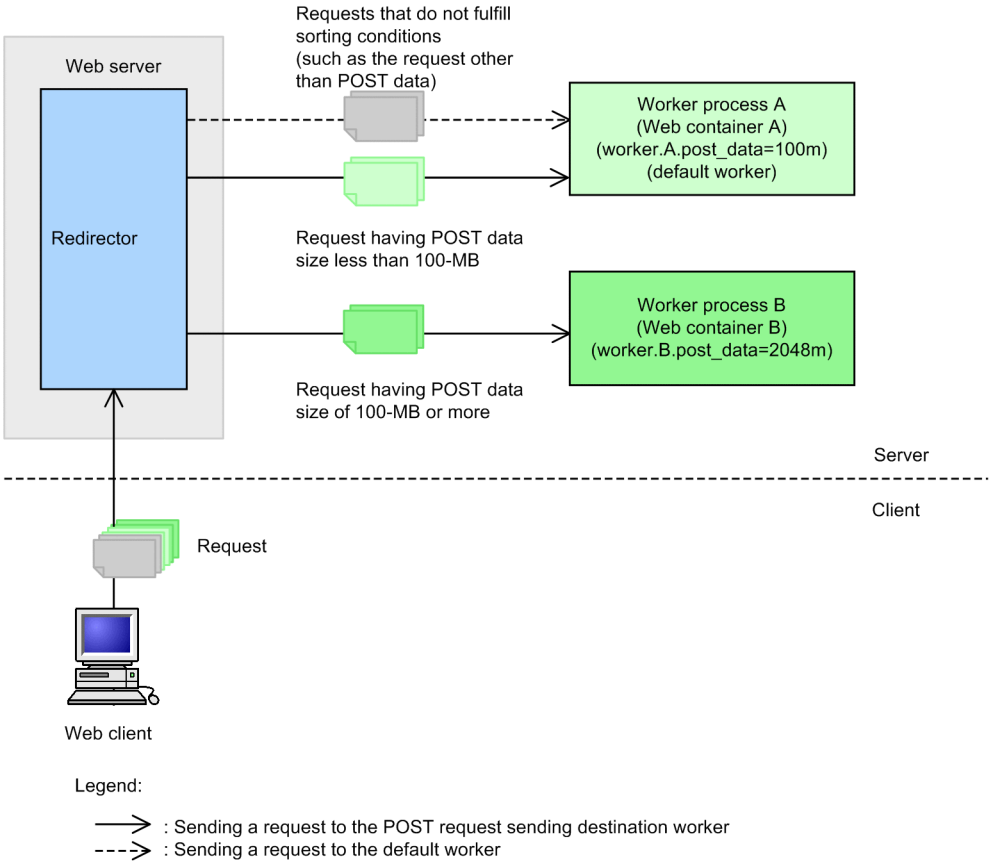
In this figure, 2 POST request-forwarding workers are prepared and upper limit is set for the POST data size in each of the workers. Maximum value is set in the upper limit for the POST data size of worker process B in such a way so that all the requests of very long POST data are processed in worker process B. POST data requests that are more than the upper limit of worker process A (100 MB or more) are forwarded to the worker process B. Note that in this case, if non-POST data requests and requests that cannot reference the POST data size are forwarded, these requests are not distributed by the request-distributing workers, so an error occurs and the redirector returns the error status code 400.

While distributing requests by the POST data size, if requests not fulfilling the request distribution conditions are forwarded to the POST request-distributing worker, and error occurs and the redirector returns the error status code 400. For details on the request distribution conditions, see *4.5.3 Request distribution conditions*.

If you want to process requests that do not fulfill the request distribution conditions, you must set up the worker process to forward that request. The worker process that forwards requests which do not fulfill the request distribution conditions is called a *default worker*. The default worker settings are optional.

The following figure shows an example in which the requests cannot be limited, and the requests not fulfilling the request distribution conditions are forwarded to the default worker:

Figure 4-12: Example of distribution of requests by the POST data size (When the default worker is set)



In this figure, settings are specified in such a way so that the requests that do not fulfill the request distribution conditions are forwarded to the worker process A of the default worker.

4.5.3 Request distribution conditions

The requests distributed to the POST request-forwarding worker must fulfill the following conditions:

Conditions of the requests distributed to the POST request-forwarding worker

- The request method is POST.
- The request has a Content-Length header (body data is not in chunk format).
- The value of the Content-Length header of the request is less than the POST data size set in the worker.

A request that does not fulfill even one of these conditions is distributed to the default worker. If the default worker is not set, an error occurs and an error with error status code 400 is returned.

4.5.4 Definition for distributing requests by the POST data size

The following workers that are used for distributing requests by the POST data size are already defined in a standard `workers.properties` file.

```
#worker.list=postsizelb1
#worker.postsizelb1.type=post_size_lb
#worker.postsizelb1.post_size_workers=worker1, worker2
#worker.postsizelb1.default_worker=worker1
```

Set the type of the worker in `worker.postsizelb1.type`. In `worker.postsizelb1.post_size_workers`, set the worker process name of the POST request-forwarding worker that forms the target of distribution. In `worker.postsizelb1.default_worker`, set the default worker. In `workers.properties`, define `post_size_lb` in the worker type, `worker1` and `worker2` in the POST request-forwarding worker, and `worker1` in the default worker as `postsizelb1`.

This definition is, however, described as a comment. Therefore, when using the POST request-distributing worker of this definition, delete the hash mark (#) at the beginning of the applicable line in `workers.properties`.

Specify the POST data size for distributing the request in the `post_data` parameter of the worker definition in `workers.properties`.

For example, use the `postsizelb1` definition that is provided by default to define the following POST data size in the 2 POST request-forwarding workers named *worker1* and *worker2* respectively:

- `post_data` parameter for worker1: 100m
- `post_data` parameter for worker2: 200m

In this case, the requests of size less than 100 MB are distributed to worker1 and the requests of size between 100 MB to 200 MB are distributed to worker2. If the request-distributing worker distributes a request of size 200 MB or more, the request is forwarded to worker1 that is set as the default worker.

4.5.5 Execution environment settings (When the Smart Composer functionality is used)

By defining the list of workers that act as the distribution destinations in the POST request-distributing worker, requests are distributed to the workers by the POST data size.

Set the upper limit for the POST data size and define the request distribution destination in the POST request-forwarding worker that acts as the distribution destination. As a result, request processing of very long POST data size with a long processing time is distributed to a specific host. The redirector distributes the requests to each HTTP request with the upper limit of the POST data size, so you can avoid the decrease in throughput of requests other than the very long POST data requests and can avoid the decrease in response time. Note that when the upper limit of the POST data size is specified, the value of the POST data size is given priority even if the HTTP request belongs to the same session.

(1) Setup procedure

To specify settings for distributing requests by the POST data size, use the following procedure:

1. Define the POST request-distributing worker and POST request-forwarding worker in workers.properties.

   Definitions for POST request-distributing worker

   Specify the list of worker names, worker types (specify `post_size_lb`), and list of workers for distribution by the POST data size. As needed, set the default worker.

   Definitions for each POST request-forwarding worker

   Specify the worker types (specify `ajp13`), port number, host name, and the upper limit of the POST data size.

   The default value is defined in `workers.properties` that is provided by default. To use the default definition defined as a comment, delete the hash mark (#) at the beginning of the applicable line.

   For details on `workers.properties` (worker definition file), see *9.5 workers.properties (worker definition file)* in the *uCosminexus Application Server Definition Reference Guide*.

2. Define the mapping between the URL pattern and worker in mod_jk.conf.

   If a mapping is defined, delete the definition or replace the mapping.

   For details on `mod_jk.conf` (redirector operation definition file for HTTP Server), see *9.3 mod_jk.conf (redirector operation definition file for HTTP Server)* in the *uCosminexus Application Server Definition Reference Guide*.

3. Set up the Web server environment and restart the Web server.

   For details on the Web server settings, see *Appendix B Precautions related to Cosminexus HTTP Server Settings*.

(2) Settings in workers.properties and mod_jk.conf

Define the settings for distributing requests by the POST data size in `workers.properties` and `mod_jk.conf`. The following table lists the keys defined in `workers.properties` and `mod_jk.conf`.

Table 4-10: Keys defined in workers.properties and mod_jk.conf (When distributing requests by the POST data size)

Types of files	Key name	Description
`workers.properties`	`worker.list`	Specifies a list of one or multiple worker names.
	`worker.`*worker-name*`.host`	Specifies the worker host name or IP address.
	`worker.`*worker-name*`.port`	Specifies the worker port number.
	`worker.`*worker-name*`.type`	Specifies the worker type. Specify `post_size_lb` in the POST request-distributing worker and `ajp13` in the POST request-forwarding worker that forms the target for distribution.
	`worker.`*worker-name*`.post_size_workers`	Specifies the list of workers that form the target of distribution by the POST data size.
	`worker.`*worker-name*`.post_data`	Specifies the upper limit (bytes) of the POST data size for the request.
	`worker.`*worker-name*`.default_worker`	Specifies the worker (default worker) for forwarding the requests if the worker applicable to the request distribution destination is not in the POST request-forwarding worker within the cluster configuration.
	`worker.`*worker-name*`.cachesize`	Specifies the number of worker connections that are reused in the redirector. This key can only be specified in Windows.
	`worker.`*worker-name*`.receive_timeout`	Specifies the communication timeout value.
	`worker.`*worker-name*`.delegate_error_code`	Specifies the error status code used when the creation of the error page is entrusted to the Web server.
`mod_jk.conf`	`JkMount`	Specifies some combination of workers specified in the URL pattern and `worker.list`.

Note:

In worker-name, define the worker name specified in the `worker.list` key or `worker.worker-name.post_size_workers` key.

The following table lists the keys that you can specify for each worker type:

Table 4-11: Keys that can be specified for each worker type

Key name	Worker type (value specified in worker.worker-name.type key)	
	POST request-distributing worker (Specify `post_size_lb`)	POST request-forwarding worker (Specify `ajp13`)
`worker.`*worker-name*`.host`	--	Y
`worker.`*worker-name*`.port`	--	Y
`worker.`*worker-name*`.type`	Y	Y
`worker.`*worker-name*`.post_size_workers`	Y	--
`worker.`*worker-name*`.post_data`	--	Y
`worker.`*worker-name*`.default_worker`	O	--

Key name	Worker type (value specified in worker.worker-name.type key)	
	POST request-distributing worker (Specify `post_size_lb`)	POST request-forwarding worker (Specify `ajp13`)
`worker.`*worker-name*`.cachesize`	--	O
`worker.`*worker-name*`.receive_timeout`	--	O
`worker.`*worker-name*`.delegate_error_code`	--	O

Legend:

Y: Can be specified

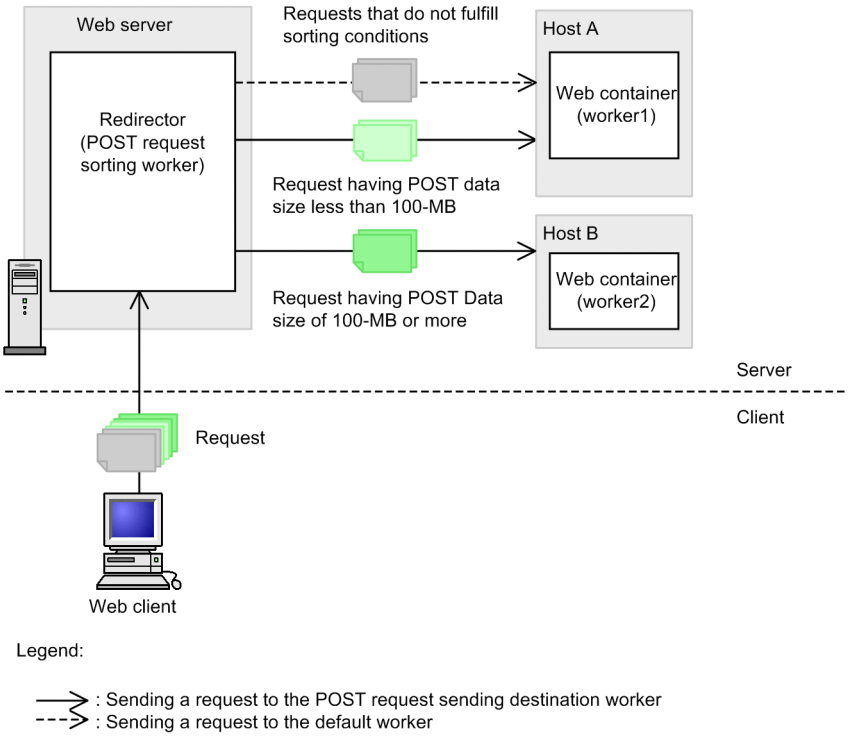
--: Cannot be specified

O: Can be optionally specified

(3) Example settings

The figure below shows the distribution of requests by POST data size. This figure shows an example when the requests cannot be limited:

Figure 4-13: Example of distribution of requests by the POST data size



In this example, among the requests under `/examples`, the requests with POST data size of less than 100 MB are distributed to host A and the requests with POST data size between 100 MB and 200 MB are distributed to host B. The requests that do not fulfill the request distribution conditions are distributed to the host A that is set as the default worker. The worker name of the host A is *worker1* and the worker name of the host B is *worker2*. For details on the request distribution conditions, see *4.5.3 Request distribution conditions*.

An example of the `workers.properties` file is described here. The POST request-distributing worker, POST request-forwarding worker, and default worker is defined here. As the upper limit of POST data size, `100m` is specified for `worker1` and `2048m` is specified for `worker2` (maximum value of the upper limit for POST data size). In the default worker, `worker1` is specified.

Example of `workers.properties` (In Windows)

```
worker.list=postsizelb1

worker.postsizelb1.post_size_workers=worker1, worker2
worker.postsizelb1.type=post_size_lb
worker.postsizelb1.default_worker=worker1

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13
worker.worker1.cachesize=64
worker.worker1.post_data=100m

worker.worker2.port=8007
worker.worker2.host=hostB
worker.worker2.type=ajp13
worker.worker2.cachesize=64
worker.worker2.post_data=2048m
```

Example of workers.properties (In UNIX)

```
worker.list=postsizelb1

worker.postsizelb1.post_size_workers=worker1, worker2
worker.postsizelb1.type=post_size_lb
worker.postsizelb1.default_worker=worker1

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13
worker.worker1.post_data=100m

worker.worker2.port=8007
worker.worker2.host=hostB
worker.worker2.type=ajp13
worker.worker2.post_data=2048m
```

An example of the `mod_jk.conf` file is shown below. POST request-distributing worker `postsizelb1` is specified here.

Example of `mod_jk.conf`

```
JkMount /examples/* postsizelb1
```

Reference note

You can also set the upper limit for the POST data size in the `LimitRequestBody` directive of `httpsd.conf` (Cosminexus HTTP Server definition file). For details on the `LimitRequestBody` directive, see the *uCosminexus Application Server HTTP Server User Guide*.

4.5.6 Execution environment settings (When the Smart Composer functionality is not used)

By defining the list of workers that act as the distribution destinations in the POST request-distributing worker, requests are distributed to the workers by the POST data size.

Set the upper limit for the POST data size and define the request distribution destination in the POST request-forwarding worker that acts as the distribution destination. As a result, request processing of very long POST data size with a long processing time is distributed to a specific host. The redirector distributes the requests to each HTTP request with the upper limit of the POST data size, so you can avoid the decrease in throughput of requests other than the very long POST data requests and can avoid the decrease in the response time. Note that when the upper limit of the POST data size is specified, the value of the POST data size is given priority even if the HTTP request belongs to the same session.

! Important note

When integrating with Microsoft IIS, you cannot specify settings for distributing requests by the POST data size.

(1) Setup procedure

To specify settings for distributing requests by the POST data size, you use the following procedure:

1. Define the POST request-distributing worker and POST request-forwarding worker in workers.properties .

   Definitions for POST request-distributing worker

   Specify the list of worker names, worker types (specify `post_size_lb`), and list of workers for distribution by the POST data size. As needed, set the default worker.

   Definitions for each POST request-forwarding worker

   Specify the worker types (specify `ajp13`), port number, host name, and the upper limit of the POST data size.

   The default value is defined in `workers.properties` that is provided by default. To use the default definition defined as a comment, delete the hash mark (#) at the beginning of the applicable line.

   For details on `workers.properties` (worker definition file), see *9.5 workers.properties (worker definition file)* in the *uCosminexus Application Server Definition Reference Guide*.

2. Define the mapping between the URL pattern and worker in mod_jk.conf.

   If a mapping is already defined, delete the definition or replace the mapping.

   For details on `mod_jk.conf` (redirector operation definition file for HTTP Server), see *9.3 mod_jk.conf (redirector operation definition file for HTTP Server)* in the *uCosminexus Application Server Definition Reference Guide*.

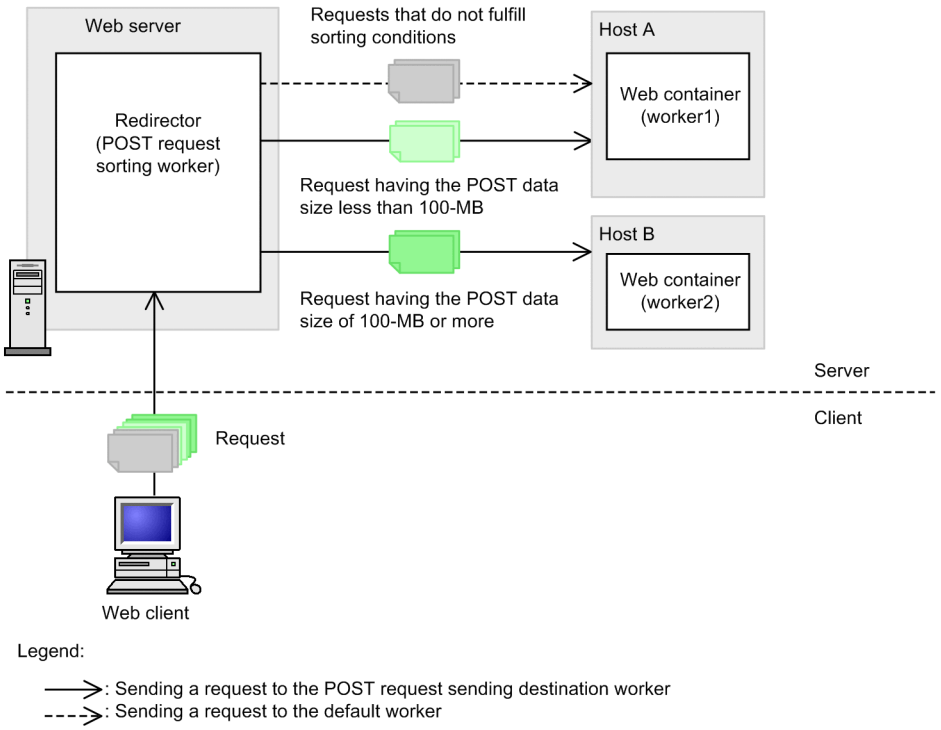
3. Set up the Web server environment and restart the Web server.

   For details on the Web server settings, see , *Appendix B Precautions related to Cosminexus HTTP Server Settings*.

(2) Example settings

The figure below shows the distribution of requests by POST data size. This figure shows an example for the case when request is not limited:

Figure 4－14: Example of distribution of requests by the POST data size



In this example, among the requests under `/examples`, the requests with POST data size of less than 100 MB are distributed to host A and the requests with POST data size between 100 MB and 200 MB are distributed to host B. The requests that do not fulfill the request distribution conditions are distributed to the host A that is set as the default

worker. The worker name of the host A is *worker1* and the worker name of the host B is *worker2*. For details on the request distribution conditions, see *4.5.3 Request distribution conditions*.

An example of the `workers.properties` file is shown below. The POST request-distributing worker, POST request-forwarding worker, and default worker are defined here. As the upper limit of POST data size, `100m` is specified for `worker1` and `2048m` is specified for `worker2` (maximum value of the upper limit for POST data size). In the default worker, `worker1` is specified.

Example of workers.properties (In Windows)

```
worker.list=postsizelb1

worker.postsizelb1.post_size_workers=worker1, worker2
worker.postsizelb1.type=post_size_lb
worker.postsizelb1.default_worker=worker1

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13
worker.worker1.cachesize=64
worker.worker1.post_data=100m

worker.worker2.port=8007
worker.worker2.host=hostB
worker.worker2.type=ajp13
worker.worker2.cachesize=64
worker.worker2.post_data=2048m
```

Example of workers.properties (In UNIX)

```
worker.list=postsizelb1

worker.postsizelb1.post_size_workers=worker1, worker2
worker.postsizelb1.type=post_size_lb
worker.postsizelb1.default_worker=worker1

worker.worker1.port=8007
worker.worker1.host=hostA
worker.worker1.type=ajp13
worker.worker1.post_data=100m

worker.worker2.port=8007
worker.worker2.host=hostB
worker.worker2.type=ajp13
worker.worker2.post_data=2048m
```

An example of the `mod_jk.conf` file is shown below. POST request-distributing worker `postsizelb1` will be specified here.

Example of mod_jk.conf

```
JkMount /examples/* postsizelb1
```

Reference note

You can also set the upper limit for the POST data size in the `LimitRequestBody` directive of `httpsd.conf` (Cosminexus HTTP Server definition file). For details on the `LimitRequestBody` directive, see the *uCosminexus Application Server HTTP Server User Guide*.

4.6 Communication timeout (Web server integration)

This section describes communication timeout in Web server integration.

When you use the functionality for Web server integration, you can set the communication timeout for receiving requests and sending responses between the client and the Web server, and also between the Web server and the Web container. When the response is awaited due to the network and application failure, you can detect the occurrence of failure from the occurrence of a timeout, if the communication timeout is set.

The following table describes the organization of this section.

Table 4-12: Organization of this section (Communication timeout (Web server integration))

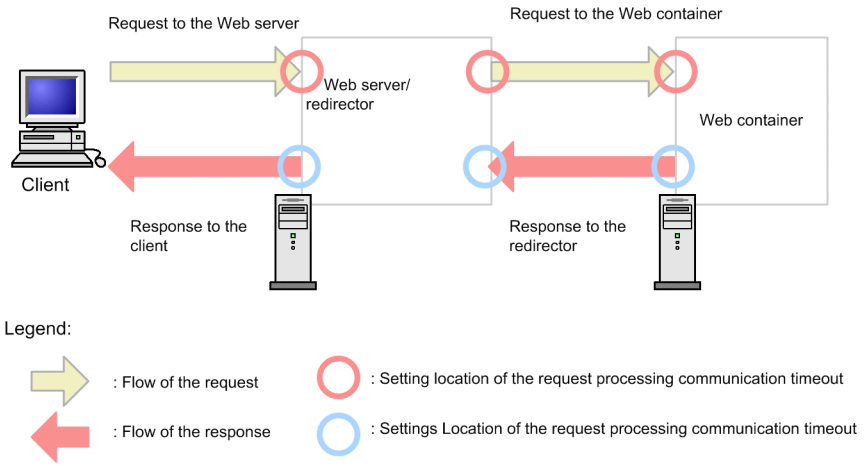
Category	Title	Reference
Description	Communication timeout when sending and receiving a request	*4.6.1*
	Communication timeout when sending and receiving a response	*4.6.2*
Settings	Setting the communication timeout	*4.6.3*
	Setting the communication timeout when sending and receiving a request (When the Smart Composer functionality is used)	*4.6.4*
	Setting the communication timeout when sending and receiving a request (When the Smart Composer functionality is not used)	*4.6.5*
	Setting the communication timeout when sending and receiving a response (When the Smart Composer functionality is used)	*4.6.6*
	Setting the communication timeout when sending and receiving a response (When the Smart Composer functionality is not used)	*4.6.7*

Note:

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

When you use the Web server integration functionality, set the communication timeout for the communication indicated by the four arrows in the following figure. In the case of communication between the redirector and the Web container, you can set the communication timeout in both the redirector and Web container. In the case of sending requests, you can set the communication timeout when the redirector sends the requests and the Web container receives the requests. Similarly, when sending responses, you can set the communication timeout when the Web container sends the responses and the redirector receives the responses. The following figure shows the communication for which timeout can be set.

Figure 4-15: Communication for which timeout can be set

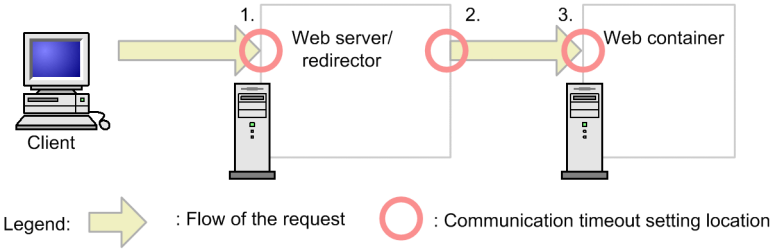


The setting of communication timeout is explained separately for receiving of a request and sending of a response.

4.6.1 Communication timeout when sending and receiving a request

This subsection explains the setting of communication timeout for sending and receiving requests when the Web server integration functionality is used. The following figure shows the locations to set the communication timeout for sending and receiving requests.

Figure 4-16: Locations to set the communication timeout for sending and receiving requests



When you use the Web server integration functionality, set the communication timeout at the locations marked with a O sign in the figure. The locations to set the communication timeout are explained below. The numbers in the figure correspond to the numbers in the following explanation:

1. When a request is received by the Web server (Client - Web server)

   Set the communication timeout when the Web server receives a request from the client. Set the communication timeout in the Web server.

   For details on the failures that can be detected by setting the communication timeout when the request is received by the Web server, see *(1) When a request is received by the Web server* explained below.

2. When a request is sent by the redirector (Redirector - Web container)

   Set the communication timeout when a request is sent from the redirector to the Web container. Set the communication timeout in the redirector.

   For details on the failures that can be detected by setting a communication timeout when the request is sent by the redirector, see *(2) When a request is sent by the redirector* explained below.

3. When a request is received by the Web container (Redirector - Web container)

   Set the communication timeout when the Web container receives a request from the redirector. Set the communication timeout in the Web container.

   For details on the failures that can be detected by setting a communication timeout when the request is received by the Web container, see *(3) When a request is received by the Web container* explained below.

(1) When a request is received by the Web server

You can set the timeout period in the Web server, for receiving a request transferred from the client. You can detect the occurrence of failure in the client by using the communication timeout that was set up. You can detect the following failures:

Detectable failures

- The host on which the client is running is down.
- A network failure occurs between the client and the Web server.
- A failure occurs in client application.

(2) When a request is sent by the redirector

You can set the timeout period in the redirector, for sending a request to the Web container. When the set timeout period is exceeded and a timeout occurs, a message is output to the error log of Cosminexus HTTP Server.

(a) Communication timeout that can be set and detectable failures

You can set the timeout to the following two time-periods when a request is sent by the redirector:

- The time to establish a connection for sending request to the Web container

- The time to send a request to the Web container

You can detect the occurrence of failure in the Web container or on the network using the communication timeout that was set up. You can detect the following failures:

Detectable failures

- The host on which the Web container is running is down.
- A network failure occurs between the redirector and the Web container.

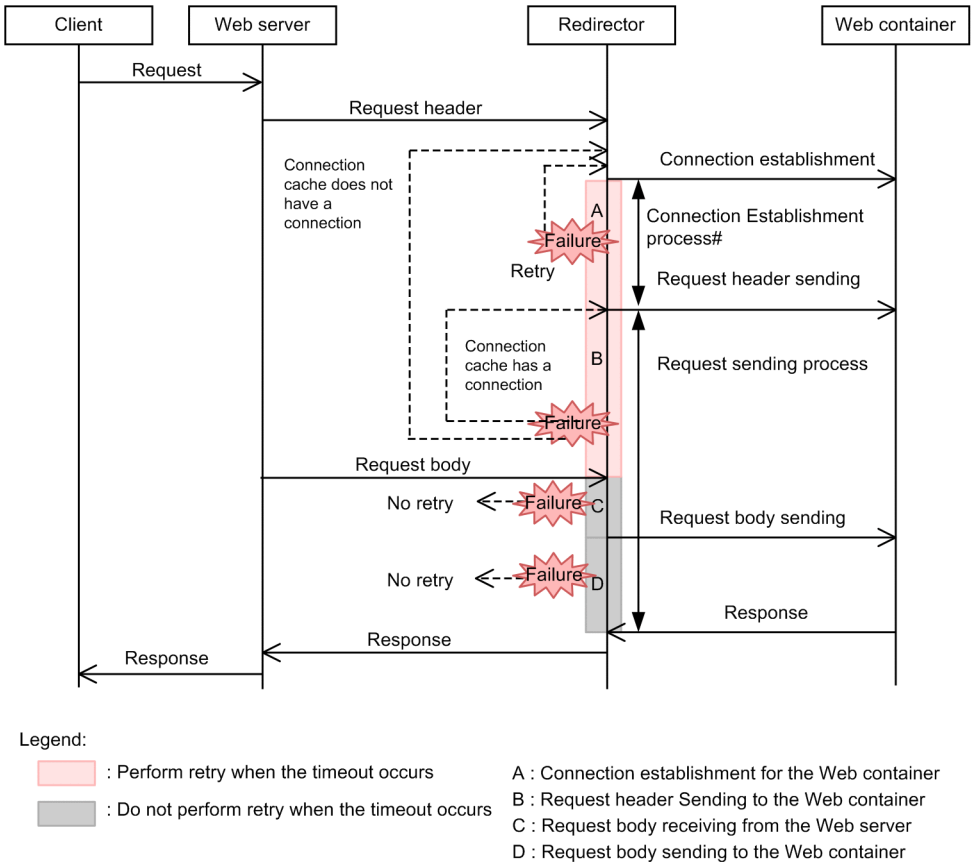
(b) Retrying sending of requests

If a request cannot be sent temporarily from the redirector, you can retry sending the request. A request might not be sent temporarily in the following cases:

- In the case of temporary failure of the network
- When requests are centralized in the Web container during establishment of a connection, and a request for establishing connection overflows temporarily from the listen queue
- When a Web container is yet to start completely

You can retry establishing a connection, and sending request headers. The following figure shows the flow of retry process.

Figure 4-17: Flow of retry process when requests are sent to the Web container



Connection establishment is carried out only when the connection cache does not have a connection.

Retry is executed when a timeout occurs in A or B of the figure. Retry is not executed when processing fails at C or D of the figure, and timeout occurs. The connection is closed and an error is returned to the client. Note that the failure of processing at C or D in the figure indicates failure in receiving request body from the Web server, or failure in sending request body to the Web container.

Tip

In the case of C or D in the figure, the processing of the request may have started already in the Web container. If you execute retry in the case of failure in receiving the request body from the Web server, or in the case of failure in sending the request body to the Web container, the send process may be duplicated, and therefore, retry is not done.

The retry operation in the case of failure in processing at A and B in the figure is explained below:

■ Retry operation in the case of failure in processing at A in the figure

After sending a request from the redirector to the Web container for establishment of a connection, if the power supply to the host on which the Web container is running is disrupted or a network failure occurs, the operation is performed as follows:

1. If the set timeout period elapses, a message indicating the occurrence of timeout during establishment of the connection is output.
2. Retry establishing connection only for specified number of times.

Note that if a connection cannot be established in spite of retrying for the specified number of times, a message indicating failure in sending request is output, and an error (status code 500) is returned to the client.

■ Retry operation in the case of failure in processing at B part in the figure

After a connection is established successfully, or the request header is sent to the Web container, if the power supply to the host on which the Web container is running is disrupted and a network failure occurs, perform the operation as follows:

1. If the set timeout period elapses, a message indicating the occurrence of timeout when sending a request is output.
2. Close the connection that was used for sending the request header.
3. Retry sending the request header only for specified number of times.
   Note that at this point, the operation depends upon whether the connections are available in the connection cache.
   - When connections are available in the connection cache
     Use a connection in the connection cache and retry the process from sending the request header.
   - When there are no connections in the connection cache
     Re-establish a connection, and then retry sending the request header.

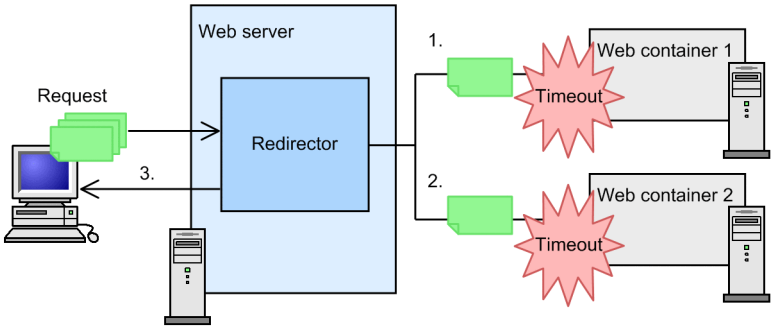
Note that if the request header cannot be sent in spite of retrying for the specified number of times, a message indicating failure in sending request is output, and status code 500 is returned to the client.

■ Retry operation when a request is distributed using a load balancer

The following figure shows the retry operation when you use a load balancer to distribute a request:

Note that in this figure, a request is first distributed to Web container 1 and then to Web container 2, and the retry frequency is set as three times.

Figure 4-18: Retry operation when a request is distributed using a load balancer



The retry operation shown in the figure is as follows:

1. Sending request to the Web container 1

   A request is sent to Web container 1. Retry operation is performed in the case of failure in establishing a connection to Web container 1, or failure in sending the request header. Retry is executed up to three times.

2. Sending request to the Web container 2

   If the retry operation fails three times in Web container 1, the request is transferred to Web container 2. When the request is transferred, retry is counted again from 1, and hence, retry is executed up to three times even in Web container 2.

3. Error notification to the client

   If the process of establishing a connection or sending the request header fails three times even in Web container 2, an error (status code 500) is returned to the client.

Reference note

A request is transferred only to the set number of Web containers.

(3) When a request is received by the Web container

You can set a timeout period in the Web container, for receiving a request transferred from the redirector. You can detect the occurrence of failure in the redirector using the communication timeout that was set up. You can detect the following failures:

Detectable failures

- The host on which the Web server is running is down.
- A network failure occurs between the Web server and the Web container.
- Timeout occurred in the Web container before the processing between the client and the Web server completes.

  This implies that when the data size requested by the Web container was being read between the client and the Web server, the communication timeout set in the Web container occurred before the data is read completely, due to insufficient communication speed between the client and the Web server.

The following table describes the conditions for occurrence of communication timeout and the operations after the timeout occurs.

Table 4-13: Conditions for occurrence of communication timeout and the operations after occurrence

Conditions for occurrence of communication timeout	Operations after occurrence
In the case all the following conditions are satisfied when a request is received: • The request contains a body data. • The body data is not in the chunk format. • After the reading process starts, a failure occurs in the host on which the redirector is running, or in the network between the redirector and the Web container.	• Request is not processed. • Message KDJE39188-E is output to the message log.
In the case all the following conditions are satisfied when using API# in the servlet (JSP): • The request contains a body data. • The body data is not in the chunk format. • After the reading process starts, a failure occurs in the host on which the redirector is running, or in the network between the redirector and the Web container.	• `java.lang.IllegalStateException` occurs. • The connection to the redirector is closed, and thereafter you cannot read or write the data. • Message KDJE39188-E is output to the message log.
In the case all the following conditions are satisfied when POST data is read by using the `java.io.BufferedReader` class acquired by `getReader` method of `javax.servlet.ServletRequest` class or `javax.servlet.ServletInputStream` class in the servlet (JSP):	• `java.net.SocketTimeoutException` occurs. • The connection to the redirector is closed, and thereafter you cannot read or write the data. • Message KDJE39188-E is output to the message log.

Conditions for occurrence of communication timeout	Operations after occurrence
• The request contains a body data. • After the reading process starts, a failure occurs in the host on which the redirector is running, or in the network between the redirector and the Web container.	• `java.net.SocketTimeoutException` occurs. • The connection to the redirector is closed, and thereafter you cannot read or write the data. • Message KDJE39188-E is output to the message log.

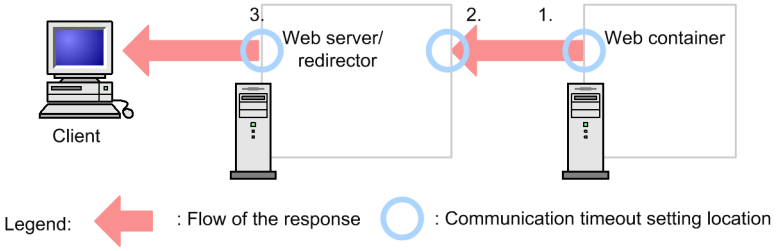
#

Indicates the case of using the `getParameter` method, `getParameterMap` method, `getParameterNames` method, and `getParameterValues` method of `javax.servlet.ServletRequest`.

4.6.2 Setting the communication timeout when sending and receiving a response

This subsection explains the setting of communication timeout for sending and receiving responses when the Web server integration functionality is used. The following figure shows the locations to set the communication timeout for sending and receiving responses:

Figure 4-19: Locations to set the communication timeout for sending and receiving responses (when the Web server integration functionality is used)



When you use the Web server integration functionality, set the communication timeout at the locations marked with a O sign in the figure. The locations to set the communication timeout are explained below. The numbers in the figure correspond to the numbers in the following explanation:

1. When a response is sent by the Web container (Web container - redirector)

   Set the communication timeout when a response is sent from the Web container to the redirector. Set the communication timeout in the Web container.

   For details on the failures that you can detect by setting the communication timeout when a response is sent by the Web container, see *(1) When a response is sent by the Web container* explained below.

2. When a response is received by the redirector (Web container - redirector)

   Set communication timeout when the redirector receives a response from the Web container. Set the communication timeout in the redirector.

   For details on the failures that you can detect by setting communication timeout when response is received by the redirector, see *(2) When a response is received by the redirector* explained below.

3. When a response is sent by the Web server (Web server - Client)

   Set communication timeout when a response is sent from the Web server to the client. Set the communication timeout in the Web server.

   For details on the failures that you can be detect by setting the communication timeout when a response is sent by the Web server, see *(3) When a response is sent by the Web server* explained below.

(1) When a response is sent by the Web container

You can set a timeout period in the Web container, for sending a response to the redirector. You can detect the occurrence of failure in the redirector using the communication timeout that was set up. You can detect the following failures:

Detectable failures

- The host on which the redirector is running is down.
- A network failure occurs between the Web container and the redirector.

When the communication timeout occurs, KDJE39507-E (timeout occurred when sending a response) is output to the message log. The following table describes the conditions for occurrence of communication timeout and the operations after the timeout occurs.

Table 4-14: Conditions for occurrence of communication timeout and the operations after occurrence

Timing of occurrence of communication timeout	Operation of the method after occurrence of communication timeout	Operation of servlets or JSPs after occurrence of communication timeout
When response data is sent to the client by using the method of `javax.servlet.ServletOutputStream` class acquired by `getOutputStream` method of `javax.servlet.ServletResponse` class, in the servlet	Exception `java.net.SocketTimeoutException` occurs.	Since the connection to the redirector is closed, you cannot send or receive the request data and the response data.
When response data is sent to the client by using the method of `java.io.PrintWriter` class acquired by `getWriter` method of `javax.servlet.ServletResponse` class, in the servlet	The send process is interrupted and returned.	• The `checkError` method of `java.io.PrintWriter` class returns true. • Since the connection to the redirector is closed, you cannot send or receive the request data and the response data.
When response data is sent to the client by using the method of `javax.servlet.jsp.JspWriter` class, in JSP	Exception `java.net.SocketTimeoutException` occurs.	Since the connection to the redirector is closed, you cannot send or receive the request data and the response data.
When the response data of static contents is sent to the client	--	--

Legend:
--: Not applicable

(2) When a response is received by the redirector

When a request is sent to the Web container, the redirector awaits for a response from the Web container. You can set the timeout for this response waiting time. You can detect the occurrence of failure in the Web container using the communication timeout that was set up. You can detect the following failures:

Detectable failures

- The host on which the Web container is running is down.
- A network failure occurs between the Web container and the redirector.
- A Web application failure occurs in the Web container.

The following failures occur in the Web application:

Web application failures

- A response is not returned due to an infinite loop in the servlets or JSPs processing.
- The Enterprise Bean and database are invoked as an extension of servlets or JSPs, and response from them is awaited.
- Dead lock occurs in the Web application.
- The Web application does not catch up with the server processing and is running slow during the peak access.

Operation after communication timeout in the redirector

When communication timeout occurs, the redirector disconnects the connection to the Web container, and returns error with status code 500 to the client.

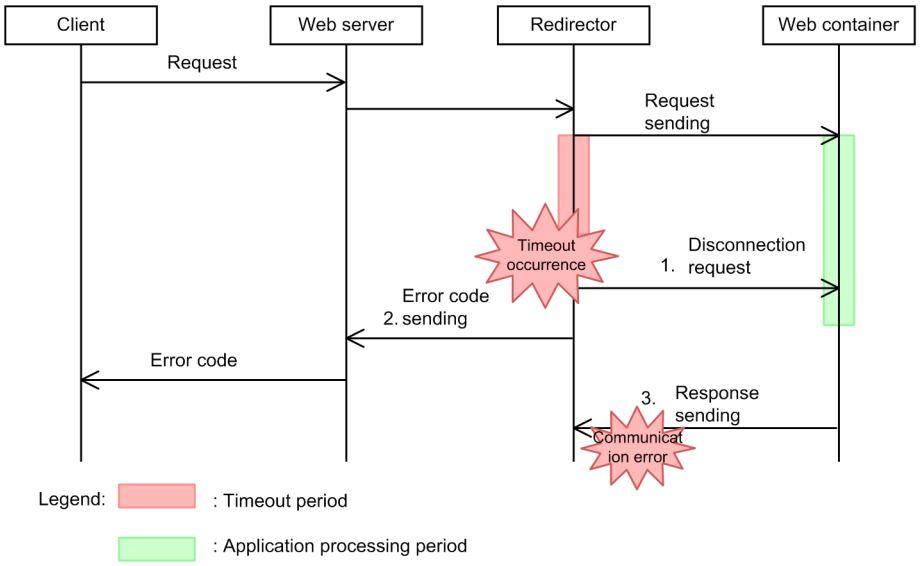
Tip

Operation when timeout occurs during processing of an application

Even if the redirector times out during processing in the Web container, you cannot detect that the redirector has timed out, in the Web container.

You can detect the timeout in the redirector once the processing of the Web container finishes, and a response is transferred to the redirector. In such a case, however, since the redirector has already disconnected the connection to the Web container, an error occurs when sending a response. The following figure illustrates the operation when a timeout occurs during processing of an application.

Figure 4-20: Operation when a timeout occurs during processing of an application



The figure is explained below.

1. When a timeout occurs in the redirector, a request is sent for disconnecting the connection to the Web container.
2. Redirector sends an error code to the Web server.
3. When processing of the application in the Web container finishes, a response is sent to the redirector. However, since the connection between the redirector and the Web container is already disconnected in 1., a communication error occurs.

(3) When a response is sent by the Web server

You can set a timeout period in the Web server, for sending data to the client. You can detect the occurrence of failure in the client by using the communication timeout that was set up. You can detect the following failures:

Detectable failures

- The host on which the client is running is down.
- A network failure occurs between the client and the Web server.
- A failure occurs in client application.

4.6.3 Setting the communication timeout

This subsection describes the communication timeout settings between a client and a Web server and between a Web server (redirector) and a Web container.

You set a communication timeout when sending and receiving requests or when sending and receiving responses. The method of specifying communication timeout differs according to the availability of Smart Composer. The following table lists the communication timeout setting method and the corresponding reference sections:

Table 4-15: Setting method and references of communication timeout (Web server integration)

Set up timing	Usage of Smart Composer	
	Used	Not used
When sending and receiving a request	*4.6.4*	*4.6.5*
When sending and receiving a response	*4.6.6*	*4.6.7*

Note that you can also set communication timeout in the EJB client that invokes EJB. Specify the settings in the EJB client during J2EE application development. For the setting method, see *2.11.5 Timeout of RMI-IIOP communications* in the *uCosminexus Application Server EJB Container Functionality Guide*.

4.6.4 Setting the communication timeout when sending and receiving a request (When the Smart Composer functionality is used)

You set the communication timeout for sending and receiving requests between the client and Web server and the redirector and Web Container.

The following are the methods of setting a communication timeout in each of these cases:

(1) Settings in the Web server for receiving requests

Specify the communication timeout in the Web server, when receiving the requests from the client into the Web server. You can use Cosminexus HTTP Server as the Web server.

(a) How to set

Set the communication timeout for the receiving process of requests forwarded from the client in `httpsd.conf`.

- **Waiting time for the request receiving process**

  In the `Timeout` directive, set the waiting time (seconds) for the process of receiving requests from the client. The default value of the `Timeout` directive is 300 seconds. Note that the value set here is shared with the communication timeout for the process of sending data to the client. For details on the communication timeout for the process of sending data to the client, see *4.6.6(3) Settings in the Web server for sending responses*.

(b) Precautions for setup

To set the timeout for receiving requests in the Web server, take into consideration the network configuration and traffic status between the client and the Web server and specify a time in which the occurrence of failure can be determined.

(2) Settings in the redirector for sending requests

When sending a request from the redirector to the Web container, first establish a connection with the Web container. You can set the communication timeout for sending requests from the redirector to the Web container when the connection is established and when the request is sent. You can also set the retry frequency to be used when an attempt to establish connection and to send the request header fails.

(a) How to set

Specify the communication timeout for establishing the connection and the request sending process and the retry frequency from the redirector in the Easy Setup definition file.

- **Communication timeout in establishing connection**

Set the waiting time (seconds) for the process of establishing a connection with the Web container in the `JkConnectTimeout` parameter in the `<configuration>` tag of the logical Web server (web-server). The default value of the `JkConnectTimeout` parameter is 30 seconds.

- **Timeout for the request sending process**
  Set the waiting time (seconds) for the process of sending a request to the Web container in the `JkSendTimeout` parameter in the `<configuration>` tag of the logical Web server (web-server). The default value of the `JkSendTimeout` parameter is 100 seconds.
- **Retry frequency for establishing a connection and sending a request**
  Set the retry frequency for establishing a connection and sending a request to the Web container in the `JkRequestRetryCount` parameter in the `<configuration>` tag of the logical Web server (web-server). The default value of the `JkRequestRetryCount` parameter is three times.

(b) Precautions for setup

Take the followings into consideration when you specify the communication timeout value for sending requests and the retry frequency set up in the redirector:

- The retry frequency includes the first connection established and the first request sent. Therefore, if the first connection or request sending process fails, the retrying of connection or request sending process is counted as the second time.
- If you specify `0` as the communication timeout for establishing a connection and for the request sending process or if you set a longer time than the re-send timer for establishing a connection and sending data by TCP, the TCP timeout value is applied to communication timeout value.

(3) Settings in the Web Container for receiving requests

Set the communication timeout when the Web container receives a request from the redirector, to the Web container.

(a) How to set

Set the communication timeout for the process of receiving requests forwarded from the redirector in the Easy Setup definition file:

- **Timeout for the request receiving process**
  Set the waiting time (seconds) for the process of receiving requests from the redirector in `webserver.connector.ajp13.receive_timeout parameter` in the `<configuration>` tag of the logical J2EE server (j2ee-server). The default value of the parameter is 100 seconds.

(b) Precautions for setup

Take the followings into the consideration when you specify the timeout value set in the Web container for receiving requests:

- Set a value bigger than the time set in the timeout for receiving Web server requests.
  If a value smaller than the time specified as the timeout for receiving Web server requests is set, when network failure occurs in the client and between the client and Web server, timeout occurs in the Web container before the Web server. In this case, one cannot determine whether the failure has occurred in the Web server or in the client.
- If data must be received from the client, set the time in which data can be received taking into consideration the communication speed with the client.
- When failure occurs in the redirector while the data is being sent to the Web container, the failure is detected by the timeout in the TCP re-send timer.
  Note that in UNIX, the time until timeout depends on the OS.

4.6.5 Setting the communication timeout when sending and receiving a request (When the Smart Composer functionality is not used)

You set the communication timeout for sending and receiving requests between the client and the Web server and between the redirector and the Web container.

The following are the methods for setting the communication timeout in each of the cases:

(1) Settings in the Web server for receiving requests

In the Web server, you specify the communication timeout for receiving the requests from the client in the Web server. You can use either Cosminexus HTTP Server or Microsoft IIS as the Web server.

(a) How to set

Set the communication timeout for the process of receiving requests forwarded from the client in the following files:

- `httpsd.conf` **(In Cosminexus HTTP Server)**

  In the `Timeout` directive, set the waiting time (seconds) for the process of receiving requests from the client. The default value of the `Timeout` directive is 300 seconds. Note that the value set here is shared with the communication timeout for the process of sending data to the client. For details on the communication timeout for the process of sending data to the client, see *4.6.6(3) Settings in the Web server for sending responses*.

- `isapi_redirect.conf` **(In Microsoft IIS)**

  Set the waiting time (seconds) for the process of receiving requests from the client in the `receive_client_timeout` key.

(b) Precautions for setup

To set the timeout for receiving requests in the Web server, take into consideration the network configuration and traffic status between the client and the Web server and specify a time in which the occurrence of failure can be determined.

(2) Settings in the redirector for sending requests

When sending a request from the redirector to the Web container, first establish a connection with the Web container. You can set the communication timeout for sending requests from the redirector to the Web container when the connection is established and when the request is sent. You can also set the retry frequency to be used when an attempt to establish connection and to send the request header fails.

(a) How to set

Specify the communication timeout for establishing the connection and the request sending process and the retry frequency from the redirector in the following files:

- `mod_jk.conf` **(In Cosminexus HTTP Server)**
  - **Communication timeout in establishing connection**

    Set the waiting time (seconds) for the process of establishing a connection with the Web Container in the `JkConnectTimeout` key. The default value of the `JkConnectTimeout` key is 30 seconds.
  - **Timeout for the request sending process**

    Set the waiting time (seconds) for the process of sending a request to the Web container in the `JkSendTimeout` key. The default value of the `JkSendTimeout` key is 100 seconds.
  - **Retry frequency for establishing a connection and sending a request**

    Set the retry frequency for establishing a connection and sending a request to the Web container in the `JkRequestRetryCount` key. The default value of the `JkRequestRetryCount` key is three times.
- `isapi_redirect.conf` **(In Microsoft IIS)**
  - Communication timeout in connection process

    Set the waiting time (seconds) for the process of establishing a connection with the Web container in the `connect_timeout` key. The default value of the `connect_timeout` key is 30 seconds.
  - Timeout for the request sending process

    Set the waiting time (seconds) for the process of sending a request to the Web container in the `send_timeout` key. The default value of the `send_timeout` key is 100 seconds.
  - Retry frequency for establishing a connection and sending a request

Set the retry frequency for establishing a connection and sending a request to the Web container in the `request_retry_count` key. The default value of the `request_retry_count` key is three times.

(b) Precautions for setup

Take the followings into the consideration when you specify the communication timeout value for sending requests and the retry frequency set in the redirector:

- The retry frequency includes the first connection established and the first request sent. Therefore, if the first connection or request sending process fails, the retrying of connection or request sending process is counted as the second time.
- If you specify `0` as the communication timeout for establishing a connection and for the request sending process or if you set a longer time than the re-send timer for establishing a connection and sending data by TCP, the TCP timeout value is applied to communication timeout value.

(3) Settings in the Web container for receiving requests

Set the communication timeout when the Web container receives a request from the redirector, to the Web container.

(a) How to set

Set the communication timeout for the process of receiving requests forwarded from the redirector in the following file:

- `usrconf.properties`

  Set the waiting time (seconds) for the process of receiving requests from the redirector in the `webserver.connector.ajp13.receive_timeout` key.

(b) Precautions for setup

Take the followings into consideration when you specify the timeout value set in the Web container for receiving requests:

- Set a value bigger than the time set in the timeout for receiving Web server requests.

  If a value smaller than the time specified as the timeout for receiving Web server requests is set, and network failure occurs in the client and between the client and Web server, the timeout occurs in the Web container before the Web server. In this case, you cannot determine whether the failure has occurred in the Web server or in the client.
- If data needs to be received from the client, set the time in which data can be received taking into consideration the communication speed with the client.
- When failure occurs in the redirector while the data is being sent to the Web Container, the failure is detected by the timeout in the TCP re-send timer.

  Note that in UNIX, the time until timeout depends on the OS.

4.6.6 Setting the communication timeout when sending and receiving a response (When the Smart Composer functionality is used)

You set the communication timeout for sending and receiving responses between the Web container and the redirector and between the Web server and the client.

The followings are the methods for setting the communication timeout in each of the cases:

(1) Settings in the Web container for sending responses

In the Web container, you specify the communication timeout for the process of sending a response from the Web container to the redirector. Set the waiting time for sending a response from the Web container in the Easy Setup definition file.

- **Timeout for the response sending process**

Set the timeout (seconds) for sending a response from the Web container in the `webserver.connector.ajp13.send_timeout` parameter in the `<configuration>` tag of the logical J2EE server (j2ee-server). The default value of the parameter is 600 seconds.

(2) Settings in the redirector for receiving responses

Set communication timeout when the redirector receives a response from the Web container, to the redirector.

(a) How to set

Set the waiting time for the response from the Web container in the Easy Setup definition file.

- **Timeout for the response receiving process**

  Set the response waiting time (seconds) for each worker in the `worker.`*worker-name*`.receive_timeout` parameter in the `<configuration>` tag of the logical J2EE server (j2ee-server). The default value of the parameter is 3600 seconds. If you want to set communication timeout for J2EE applications, define and map workers for each J2EE application.

(b) Precautions for setup

You can detect Web application failure by the timeout in receiving a response in the redirector. Therefore, depending on the operation state, consider the time required for processing Web applications and set a value from which failure can be detected as the communication timeout value.

Take the followings into the consideration when you set the timeout value for receiving responses in the redirector:

- Set a time longer than the time required for processing the Web applications in communication timeout.

  If the set timeout value is shorter than the Web application processing time, even if the Web application is processing normally, the timeout is determined in the redirector and an error is returned to the client.
- Consider the waiting time based on the controlling of the number of concurrently executing threads at the peak of the Web container.

  Because of the controlling of the number of concurrently executing threads, at the peak of the Web Container, there might be requests awaiting processing. Therefore, if controlling the number of concurrently executing threads is set, you must specify the communication timeout considering the extension of the request processing time. For details on controlling the number of concurrently executing threads, see *2.15 Overview of controlling the number of concurrently executing threads*.
- When failure occurs in the Web container while the data is being sent to the redirector, the failure is detected by the timeout in the TCP re-send timer.

  Note that in UNIX, the time until timeout depends on the OS.

(3) Settings in the Web server for sending responses

In the Web server, you set the communication timeout for sending a response to the client.

(a) How to set

Set the waiting time for the response from the client in `httpsd.conf`.

- **Communication timeout for the data sending process**

  Set the communication timeout in the `Timeout` directive. Note that the communication timeout specified in the `Timeout` directive is specified both as the communication timeout for the request receiving process and the communication timeout for the response sending process. Therefore, you cannot set a different time in the communication timeout for the request receiving process and the communication timeout for the response sending process.

  For details on the communication timeout for the process of receiving requests from clients, see *4.6.4(1) Settings in the Web server for receiving requests*.

(b) Precautions for setup

To set the timeout for sending responses in the Web server, take into the consideration the network configuration and traffic status between the client and the Web server and specify adequate time for sending and receiving data with the client.

Also, for the timeout value set in the Web server for sending responses, specify a value smaller than the timeout value in the Web container for sending responses. If the timeout for sending responses in the Web server is greater than timeout for sending responses in the Web container, and a failure occurs between the client and Web server, the timeout in the Web container for sending a response to the redirector might occur earlier than the timeout in the Web server for sending response to the client. In this case, one cannot determine whether the failure has occurred between the client and the Web server or between the redirector and Web container.

4.6.7 Setting the communication timeout when sending and receiving a response (When the Smart Composer functionality is not used)

You set the communication timeout for sending and receiving responses between the Web container and the redirector and between the Web server and the client.

The following are the methods for setting a communication timeout in each of the cases:

(1) Settings in the Web container for sending responses

In the Web container, you specify the communication timeout for the process of sending a response from the Web container to the redirector. Set the waiting time for sending a response from the Web container in the following file:

- **usrconf.properties**
  Set the timeout (seconds) for sending a response from the Web container in the `webserver.connector.ajp13.send_timeout` key. The default value is 600 seconds.

(2) Settings in the redirector for receiving responses

Set the communication timeout when the redirector receives a response from the Web container, to the redirector.

(a) How to set

Set the waiting time for the response from the Web container in the following file:

- `workers.properties`
  Set the response waiting time (seconds) for each worker in the `worker.`*worker-name*`.receive_timeout` key. If you want to set a communication timeout for J2EE applications, define and map workers for each J2EE application.

(b) Precautions for setup

You can detect Web application failure by a timeout in receiving a response in the redirector. Therefore, depending on the operation state, consider the time required for processing Web applications and set a value from which failure can be detected as the communication timeout value.

Take the following into the consideration when you set the timeout value for receiving responses in the redirector:

- Set a time longer than the time required for processing the Web applications in the communication timeout.
  If the set timeout value is shorter than the Web application processing time, the timeout is determined in the redirector and an error is returned to the client even if the Web application is processing normally.
- Consider the waiting time based on the controlling of the number of concurrently executing threads at the peak of the Web container.
  Because of the controlling of the number of concurrently executing threads, at the peak of the Web container, there might be requests awaiting processing. Therefore, if server management commands are used to specify the control of the number of concurrently executing threads, you must specify the communication timeout considering the extension of the request processing time. For details on controlling the number of concurrently executing threads by using the server management commands, see *2.15 Overview of controlling the number of concurrently executing threads*.

- When failure occurs in the Web container while the data is being sent to the redirector, the failure is detected by the timeout in the TCP re-send timer.

  Note that in UNIX, the time until timeout depends on the OS.

(3) Settings in the Web server for sending responses

In the Web server, you can set the communication timeout for sending a response to the client.

(a) How to set

Set the waiting time for the response from the client in the following files:

- `httpsd.conf` **(In Cosminexus HTTP Server)**

  Set the communication timeout in the `Timeout` directive. Note that the communication timeout specified in the `Timeout` directive is specified both as the communication timeout for the request receiving process and the communication timeout for the response sending process. Therefore, you cannot set the different time for the communication timeout of the request receiving process and for the communication timeout of the response sending process.

  For details on the communication timeout for the process of receiving requests from clients, see *4.6.4(1) Settings in the Web server for receiving requests*.

- `MinFileBytesPerSec` **property (In Microsoft IIS)**

  Set the communication timeout in the `MinFileBytesPerSec` property. In the `MinFileBytesPerSec` property, specify the throughput (byte/second) of the response data sent from the Web server to the client. The communication timeout occurs when the response data throughput falls below the value set in `MinFileBytesPerSec`. Note that, the default value for a communication timeout is 240 byte/second.

  For details on the settings, see the manual *Microsoft IIS*.

(b) Precautions for setup

To set the timeout for sending responses in the Web server, take into consideration the network configuration and traffic status between the client and the Web server and specify adequate time for sending and receiving data with the client.

Also, as the timeout value set in the Web server for sending responses, specify a value smaller than the timeout value in the Web container for sending responses. If the timeout for sending responses in the Web server is greater than timeout for sending responses in the Web container, when failure occurs between the client and the Web server, the timeout in the Web container for sending a response to the redirector might occur earlier than the timeout in the Web server for sending response to the client. In this case, one cannot determine whether the failure has occurred between the client and the Web server or between the redirector and the Web container.

4.7 Specifying the IP address (Web server integration)

This section describes the control of communication with the Web client by specifying the IP address in Web server integration.

The following table describes the organization of this section.

Table 4-16: Organization of this section (Specifying the IP address (Web server integration))

Category	Title	Reference
Description	Bind address specification functionality	*4.7.1*
Settings	Execution environment settings (J2EE server settings)	*4.7.2*
Notes	Precautions for specifying the IP address in Web server integration	*4.7.3*

Note:
There is no description of *Implementation* and *Operations* for this functionality.

4.7.1 Bind address specification functionality

In a Web container, you can explicitly specify the IP address to be used in Web server integration. This functionality is called the *Bind address specification functionality*. By using the bind address specification functionality, you can specify the setting so that only a single specific IP address is used for a host having multiple physical network interfaces or single physical network interface, when executing with a host.

Customize the properties of the J2EE server to set the bind IP address. For details on customizing the J2EE server operation settings, see *4.7.2 Execution environment settings (J2EE server settings)*.

4.7.2 Execution environment settings (J2EE server settings)

To specify the IP address in Web server integration, you must set up the J2EE server.

Implement the J2EE server settings in the Easy Setup definition file. To define the IP address in Web server integration, specify the following parameter in the <configuration> tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file:

`webserver.connector.ajp13.bind_host`
Specifies the IP address or host name used when the Web server integration functionality is used.

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

4.7.3 Precautions for specifying the IP address in Web server integration

The following are the precautions for specifying the IP address in Web server integration:

- When the host name or the IP address is set, only requests for connecting to the specified IP address can be received. Instead of setting the IP address, a connection to any IP address on that host can be received, by specifying the wild card address. By default, the setting is specified to use the wild card address. When using the wild card address, note the following points:
  - If the specified host name cannot be resolved in the hosts file or DNS, start the server by using the wild card address.
  - If the specified host name or IP address is a remote host, start the server by using the wild card address.

4.8 Error page customization with the Web server integration functionality

When the client accesses a non-existent resource, and a servlet in which an exception occurred, the Web container returns an error status code. An error page corresponding to the error status code returned from the Web container is displayed in the client. In an application server, instead of the error pages displayed in the client, pages created by the user can be displayed in the client. This is called *error page customization*.

This section describes the customization of the error page with the Web server functionality when the system is integrated with the Web server.

The following table describes the organization of this section.

Table 4-17: Organization of this section (Error page customization (Web server integration))

Category	Title	Reference
Description	Overview of error page customization	*4.8.1*
	Mechanism of error page customization	*4.8.2*
Settings	Execution environment settings (When the Smart Composer functionality is used)	*4.8.3*
	Execution environment settings (When the Smart Composer functionality is not used)	*4.8.4*
Notes	Precautions related to error page customization	*4.8.5*

Note:
There is no specific description of *Implementation* and *Operations* for this functionality.

! Important note

You can use the error page customization with the Web server functionality, only when you use the Web server integration functionality. You can use the error page customization functionality only in Cosminexus HTTP Server. You cannot use this functionality in Microsoft IIS.

4.8.1 Overview of error page customization

The methods to customize the error pages include: Customization with the `<error-page>` tag of `web.xml` specified in Servlet specifications and customization with the Web server functionality. However, the error page used when the redirector returns an error such as when the communication between the redirector and Web container fails cannot be customized with the method of using the `<error-page>` tag of `web.xml`. Use the Web server functionality to customize the error page when the redirector returns an error. The following table describes the error locations and the corresponding error page customization methods:

Table 4-18: Error locations and the corresponding error page customization methods

Error location	Customization method	
	Method of using the Web server functionality	Method of using the `<error-page>` tag of `web.xml`
Web container	Y	Y
Redirector	Y	--

Legend:
Y: Can be customized
--: Cannot be customized

For details on the conditions for occurrence of an error in the Web container, and the corresponding error status codes, see *Appendix A Error Status Code*.

4.8.2 Mechanism of error page customization

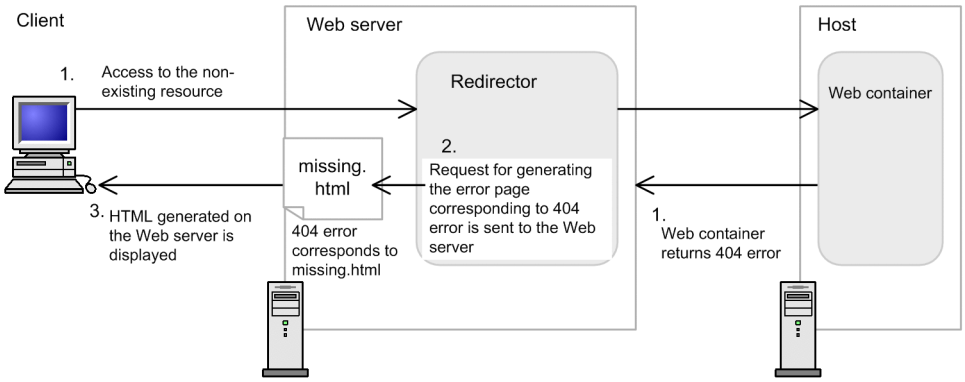
This subsection describes the mechanism of the process of error page customization when an error occurs in the Web container and when an error occurs in the redirector.

When an error occurs in the Web container

The redirector receives the error status codes sent from the Web container. The redirector assigns creation of error pages to the Web server, and the Web server sends the user created pages corresponding to the error status codes to the client. As a result, the pages created by the user are displayed in the client.

The following figure shows the processing flow of error page customization:

Figure 4-21: Processing of displaying error pages created by the user (when the Web server functionality is used)



Stages 1 to 3 of the figure are explained below:

1. If the client accesses a non-existent resource, the Web container sends error 404 to the Web server.
2. When the redirector receives error 404, it requests the Web server to generate an error page corresponding to error 404, based on the setting information#.
3. The Web server returns the error page 'missing.html' corresponding to error 404 to the client according to the setting information#.

When an error occurs in the redirector

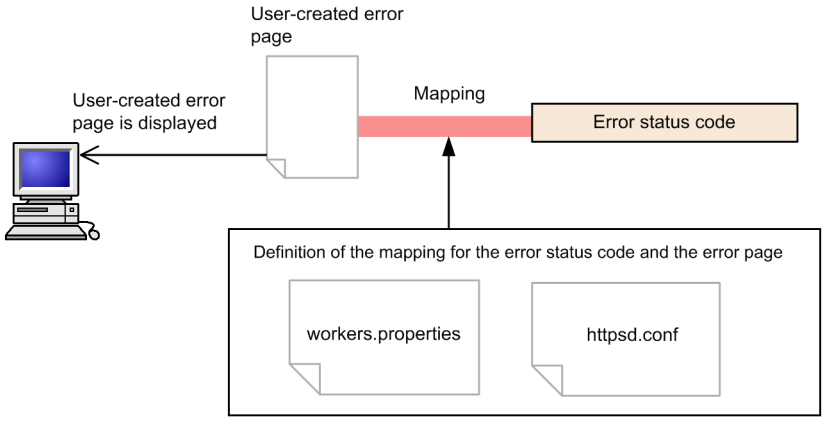
If an error occurs in the redirector, the redirector requests the Web server to generate an error page corresponding to the occurred error, on the basis of the setting information. The Web server sends the user-created page corresponding to the error status code to the client, based on the setting information#. As a result, the pages created by the user are displayed in the client.

#

To customize the error pages, you need to specify the relationship between the error status code and the error page, beforehand.

The following figure shows an overview of the relationship.

Figure 4-22: Specifying the relationship between the error status code and the error page (by using the Web server functionality)



When an error occurs, in order to display the error page created by the user instead of the error page displaying the error status code, you associate the error page created by the user to a specific error status code. When an error with the applicable error status code occurs, the error page corresponding to the error status code is sent to the client, on the basis of the information set in the Web server (Cosminexus HTTP Server).

4.8.3 Execution environment settings (When the Smart Composer functionality is used)

This subsection describes the settings for the error page customization.

(1) How to set

Define the association between the error status code and the error page in the following files:

- Easy Setup definition file

  Specify the error status code that you want to associate with the error page in the `worker.`*worker-name*`.delegate_error_code` parameter in the `<configuration>` tag of the logical Web server (web-server).

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

- `httpsd.conf`

  Associate the error status code and the file name of the corresponding error page in the `ErrorDocument` directive.

  For details on httpsd.conf (HTTP Server definition file), see the *uCosminexus Application Server HTTP Server User Guide*.

(a) Precautions when specifying the error status codes

Take the following precautions when specifying the error status codes in the Easy Setup definition file:

- Specify the error status code for each worker.
- The worker type that can specify the error status code is only `ajp13`. If the worker type is `lb` (settings specified for load balancing based on the round-robin format) and `post_size_lb` (settings specified for distributing requests based on the POST data size), the specified contents are ignored.
- The specifiable error status code is listed in the following table. The error page cannot be associated with error status codes other than the followings:

Table 4-19: Error status codes that can be associated with error pages

Error status codes	Explanation
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Time-out
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
422	Unprocessible Entity
423	Locked
424	Failed Dependency
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Time-out
505	HTTP Version not supported
507	Insufficient Storage
510	Not Extended

(b) Precautions for specifying the ErrorDocument directive

Take the following precautions when you specify the `ErrorDocument` directive in `httpsd.conf`:

- When using the local URL in the `ErrorDocument` directive, specify a URL that the redirector will not forward to the Web container.
- When the URL pattern `/*` is mapped to a worker in the redirector settings such as for using the root context, all the requests are forwarded to the Web container. Therefore, in the ErrorDocument directive, set the resources on the Web container by using the complete URL.

The following is the example settings for displaying `error404.jsp` under the root context on the Web container when the root context is used and the error status code 404 occurs. The `hostA` is the host operating the Web server.

Example:

ErrorDocument 404 http://hostA/error404.jsp

Also, when the Web container is not running, the redirector returns an error with error status code 500. Therefore, for customizing the error page when the Web container is not running, you must specify other Web server resources for the error status code 500 using the complete URL, in the `ErrorDocument` directive.

(2) Example settings

The following example describes the error page customization:

Example of Easy Setup definition file

```
...
<param>
 <param-name>worker.list</param-name>
 <param-value>worker1</param-value>
</param>
<param>
 <param-name>worker.worker1.type</param-name>
 <param-value>ajp13</param-value>
</param>
<param>
 <param-name>worker.worker1.host</param-name>
 <param-value>host1</param-value>
</param>
<param>
 <param-name>worker.worker1.delegate_error_code</param-name>
 <param-value>404</param-value>
</param>
...
```

Define the error status code '404(Not Found)' in the `worker.`*worker-name*`.delegate_error_code` parameter.

Example of `httpsd.conf`

```
# Description of httpsd.conf#
#  ...
ErrorDocument 404 /missing.html
```

The error status code and the file name of the corresponding error page are associated. When an error with error status code '404(Not Found)' occurs, the `missing.html` file is displayed.

For details on the `ErrorDocument` directive, see the *uCosminexus Application Server HTTP Server User Guide*.

4.8.4 Execution environment settings (When the Smart Composer functionality is not used)

This subsection describes the settings for error page customization.

(1) How to set

Define the association between the error status code and error page in the following files:

- `workers.properties`

  Specify the error status code that you want associated with the error page in the `worker.`*worker-name*`.delegate_error_code` key.

  For details on `workers.properties` (worker definition file), see *9.5 workers.properties (worker definition file)* in the *uCosminexus Application Server Definition Reference Guide*.

- `httpsd.conf`

Associate the error status code and the file name of the corresponding error page in the `ErrorDocument` directive.

For details on httpsd.conf (HTTP Server definition file), see the *uCosminexus Application Server HTTP Server User Guide*.

Precautions related to workers.properties settings

- Specify the error status code for each worker.
- The worker type that can specify the error status code is only `ajp13`. If the worker type is `lb` (settings specified for load balancing based on the round-robin format), the specified contents are ignored.
- The specifiable error status code is listed in the following table. The error page cannot be associated with error status codes other than the followings:

Table 4-20: Error status codes that can be associated with error pages

Error status codes	Explanation
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Time-out
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
422	Unprocessible Entity
423	Locked
424	Failed Dependency
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Time-out
505	HTTP Version not supported

Error status codes	Explanation
507	Insufficient Storage
510	Not Extended

Precautions for specifying the ErrorDocument directive

- When using the local URL in the ErrorDocument directive, specify a URL that the redirector will not forward to the Web Container.
- When the URL pattern `/*` is mapped to a worker in the redirector settings such as for using the root context, all the requests are forwarded to the Web container. Therefore, in the `ErrorDocument` directive, set the resources on the Web container by using the complete URL.

  The following example describes the settings for displaying `error404.jsp` under the root context on the Web container when the root context is used and the error status code 404 occurs. The `hostA` is the host operating the Web server.

  ErrorDocument 404 http://hostA/error404.jsp

  Also, when the Web container is not running, the redirector returns an error with error status code 500. Therefore, for customizing the error page when the Web container is not running, you must specify other Web server resources for the error status code 500 using the complete URL, in the `ErrorDocument` directive.

(2) Example settings

The following is an example of error page customization:

Example of `workers.properties`

```
# Description of worker definition file
worker.list=worker1

worker.worker1.type=ajp13
worker.worker1.host=host1
worker.worker1.port=8007
worker.worker1.delegate_error_code=404
```

Define the error status code '404(Not Found)' in the `worker.`*worker-name*`.delegate_error_code` key.

Example of `httpsd.conf`

```
# Description of httpsd.conf#
#  ...
ErrorDocument 404 /missing.html
```

The error status code and the file name of the corresponding error page are associated. When an error with error status code '404(Not Found)' occurs, the `missing.html` file is displayed.

For details on the `ErrorDocument` directive, see the *uCosminexus Application Server HTTP Server User Guide*.

4.8.5 Precautions related to error page customization

Note the following points when customizing the error pages with the Web server functionality, in the case of using the Web server integration functionality:

- You can use the error page customization functionality only in Cosminexus HTTP Server. For this reason, when integrating with the Microsoft IIS, even if error page customization is set in `workers.properties`, it becomes invalid.
- When the Web application supports the Servlet 2.3 specifications and you use the error page customization functionality with the `<error-page>` tag of `web.xml` specified in the Servlet specifications, the Web container returns the result of access to the pages described in the `<error-page>` tag, as the status code. Therefore, if an error does not occur in access to the pages described in the `<error-page>` tag , this functionality does not work.

- If the settings for error page customization are specified only in either the worker definition (the `workers.properties` or Easy Setup definition file) or `httpsd.conf`, then even if the specified error occurs in the Web container, the file set by the user is not displayed.

  If the error status code entrusted with the generation of the error page is only specified in the worker definition (the `ErrorDocument` directive of `httpsd.conf` is not defined), the error page returned when the error with that error status code occurs is the page that is automatically generated by Cosminexus HTTP Server.

4.9 Viewing the top page by specifying the domain name

When accessing a deployed Web application merely by specifying the domain name in the URL, the top pages of Web applications, such as index.html and index.jsp can be displayed. You can use this functionality only when you use the Web server integration functionality. Files such as index.html and index.jsp are called *welcome files*.

This section describes the viewing of the top page by specifying the domain name.

The following table describes the organization of this section.

Table 4-21: Organization of this section (Viewing the top page by specifying the domain name)

Category	Title	Reference
Description	Viewing the top page by specifying the domain name	*4.9.1*
Settings	Execution environment settings (When the Smart Composer functionality is used)	*4.9.2*
	Execution environment settings (When the Smart Composer functionality is not used)	*4.9.3*

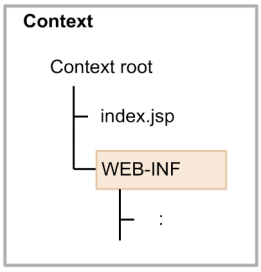
Note:
There is no description of *Implementation*, *Operations*, and *Notes* for this functionality.

4.9.1 Viewing the top page by specifying the domain name

To display the top page only by specifying the domain name, you need to deploy the welcome file in the root context. Root context refers to a context whose context root# is a null character (name is not specified in the context root).

#
The unit of management that compile the Web applications is called a *context*. The root path of this context is called a *context root*. When accessing a Web application, specify the context root on the URL.
The following figure explains the context and the context root:

Figure 4-23: Context and context root



The following settings are required to display the top page only by specifying the domain name:

- **Settings of the redirector**
  The root context is accessed via the Web server. Consequently, you need to specify the settings in the URL mapping definition of the redirector, so that the corresponding URL is redirected. Specify the settings in either `mod_jk.conf` (in Cosminexus HTTP Server) or `uriworkermap.properties` (in Microsoft IIS).
- **Settings of the application**
  Specify a null character in the context root of the imported J2EE application.

■ Notes

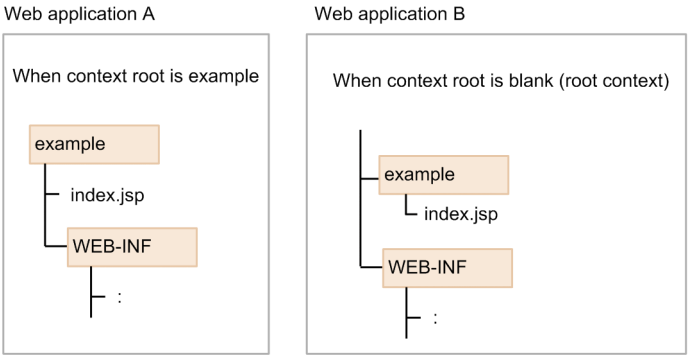
Note the following points when using the 'Viewing the top page by specifying the domain name' functionality:

- **Accessed hierarchy when the context root and the root context have the same hierarchies**
  When the context root and the root context have the same hierarchies, the hierarchy of the context root is accessed. An example is shown below.

Example:

In this example, the context root of Web application A is 'example', while the context root of Web application B is a null character, and both the Web applications have the hierarchy called 'example'.

Figure 4-24: Example of accessed hierarchy when the context root and root context have the same hierarchies



In this case, when 'http://host-name/example' is accessed, example/index.jsp of Web application A that has a context root is executed.

If, however, the directory contains "forward" and "include", and for example "forward" in the directory is accessed, URL is forwarded to the index.jsp of the root context.

- **Configuration in the Web application**

  You cannot use 'ejb' and 'web' at the beginning of the URL.

  Examples of URLs in which you cannot use the 'ejb' and 'web' at the beginning:

  http://host-name:port-number/ejb/

  http://host-name:port-number/web/

  For this reason, do not configure a Web application to be deployed as the root context, so that 'ejb' or 'web' is at the beginning.

- **How to display in a message text**

  In the messages output to the console and log files, the context root is displayed as a null character.

4.9.2 Execution environment settings (When the Smart Composer functionality is used)

This subsection describes the settings for viewing the top page by specifying the domain name.

When accessing a deployed Web application merely by specifying the domain name in the URL, the top pages of Web applications, such as index.html and index.jsp can be displayed.

(1) How to set

To view the top page by specifying the domain name:

1. Specify the root context.

   The root context is a context in which the name is not specified for the context root. The specification of the root context differs according to the operation mode.

   For defining the context root for the J2EE application, see *9.11.1 Defining the context root of J2EE applications* in the *uCosminexus Application Server Application Setup Guide*.

2. Specify distribution of requests to the root context in the redirector.

   Specify the distribution of requests to the root context in the Easy Setup definition file.

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

(2) Example settings

The following is an example of settings for distributing requests to the root context.

To view the top page of a Web application by specifying only the domain name in the URL, specify settings in the URL mapping definition of the redirector in such a way so that the requests are distributed to the root context. For example, to distribute root context to `worker1` and `/examples` to `worker2`, specify as follows:

Example of Easy Setup definition file

```
...
<param>
 <param-name>JkMount</param-name>
 <param-value>/* worker1</param-value>
 <param-value>/examples/* worker2</param-value>
</param>
...
```

4.9.3 Execution environment settings (When the Smart Composer functionality is not used)

This subsection describes the settings for viewing the top page by specifying the domain name.

When accessing a deployed Web application merely by specifying the domain name in the URL, the top pages of Web applications, such as index.html and index.jsp can be displayed.

(1) How to set

To view the top page by specifying the domain name:

1. Specify the root context.

   The root context is a context in which the name is not specified for the context root. The specification of the root context differs according to the operation mode.

   Use the server management commands to specify the root context when you define the J2EE application properties. To set the root context as the context root, specify a null character. For defining the context root for the J2EE application, see *9.11.1 Defining the context root of J2EE applications* in the *uCosminexus Application Server Application Setup Guide*.

2. Specify distribution of requests to the root context in the redirector.

   Specify the distribution of requests to the root context in `mod_jk.conf` when using Cosminexus HTTP Server as the Web server and in `uriworkermap.properties` when using Microsoft IIS as the Web server.

   For details on `mod_jk.conf` (redirector operation definition file for HTTP Server), see *9.3 mod_jk.conf (redirector operation definition file for HTTP Server)* in the *uCosminexus Application Server Definition Reference Guide*.

   For details on `uriworkermap.properties` (mapping definition file for Microsoft IIS), see *9.4 uriworkermap.properties (mapping definition file for Microsoft IIS)* in the *uCosminexus Application Server Definition Reference Guide*.

(2) Example settings

An example of settings for distributing requests to the root context is as follows.

To view the top page of a Web application by specifying only the domain name in the URL, specify settings in the URL mapping definition of the redirector in such a way so that the requests are distributed to the root context. For example, to distribute root context to `worker1` and `/examples` to `worker2`, specify as follows:

Example of mod_jk.conf (in Cosminexus HTTP Server)

```
JkMount /* worker1
JkMount /examples/* worker2
```

Example of uriworkermap.properties (In Microsoft IIS)

```
/*=worker1
/examples/*=worker2
```

4.10 Notification of gateway information to a Web container

This section describes the reporting of the gateway information to a Web container.

This functionality notifies a Web container of gateway information so that the Web container can properly redirect to a welcome file or Form authentication window.

The following table describes the organization of this section.

Table 4-22: Organization of this section (Reporting the gateway information to a Web Container)

Category	Title	Reference
Description	Gateway specification functionality	*4.10.1*
Settings	Execution environment settings (When the Smart Composer functionality is used)	*4.10.2*
	Execution environment settings (When the Smart Composer functionality is not used)	*4.10.3*
Notes	Precautions related to reporting the gateway information to a Web Container	*4.10.4*

Note:
There is no specific description of *Implementation* and *Operations* for this functionality.

4.10.1 Gateway specification functionality

If a gateway such as an SSL accelerator or a load balancer is placed between a client and a Web server, when the Web container automatically redirects to a welcome file or the Form authentication window, the Web container may not properly create a forwarding URL because the container cannot acquire the information about the gateway.

To avoid this problem, you can use the *gateway specification functionality*. This functionality notifies a Web container of gateway information so that the Web container can properly redirect to a welcome file or Form authentication window.

The gateway specification functionality is used in the following case:

- **When an SSL accelerator is placed between a client and Web server:**

  Even if a client accesses an SSL accelerator via HTTPS, the SSL accelerator accesses a Web server via HTTP, which causes the Web container to assume that the access uses HTTP. For this reason, HTTP is used for the URL scheme for the welcome file or Form authentication window that is the redirection destination.

  In this situation, by using the gateway specification function to specify that the scheme be always considered as HTTPS, you can ensure that accesses are properly redirected.

- **When a request without a Host header needs to be redirected away from the Web server that received the request**

  When redirecting a request without a Host header, the host name and the port number of the redirection destination URL will be the host name and the port number of the Web server that receives the request.

  Use the gateway specification functionality when the host name and port number of the URL accessed by the client is different from the Web server that receives the request, such as when a load balancer is deployed before the Web server. As a result, the host name and port number accessed from the client are specified, so the request can be redirected properly.

Note that in the case of Web server integration, gateway specification functionality cannot be used when multiple different routes are used for accessing one Web container (when HTTP requests are forwarded to the Web container from multiple gateways). To use the gateway specification functionality in the case of Web server integration, use a configuration in which there is one access route to the Web Container.

4.10.2 Execution environment settings (When the Smart Composer functionality is used)

This subsection describes the settings to use the gateway specification functionality.

When a gateway such as an SSL accelerator or load balancer is placed between a client and a Web server, you can use the gateway specification functionality to report the gateway information to the Web container and can properly redirect the access to the top page of the Web application or Form authentication window.

(1) How to set

To use the gateway specification functionality:

1. Specify the gateway host name, port number, and URL scheme for redirect destination for each redirector.
2. Restart the Web server.

Specify the gateway host name, port number, and URL scheme for redirect destination in the Easy Setup definition file. Specify the following parameters in the `<configuration>` tag of the logical Web server (web-server):

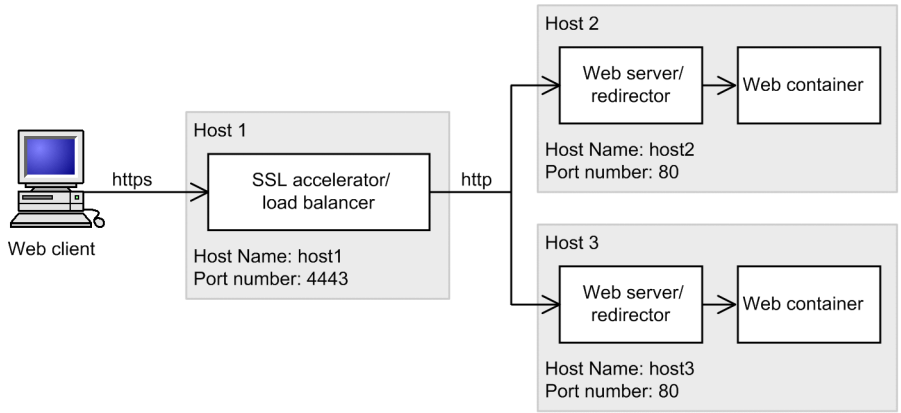
- Host name: `JkGatewayHost`
- Port number: `JkGatewayPort`
- URL scheme for redirect destination: `JkGatewayHttpsScheme`

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

(2) Example settings

The following figure shows the example settings for the gateway specification functionality:

Figure 4-25: Example settings for the gateway specification functionality



In this example, an SSL accelerator is placed between the client and Web server. Even if a client accesses an SSL accelerator via HTTPS, the SSL accelerator accesses a Web server via HTTP, which causes the Web container to assume that the access uses HTTP. For this reason, HTTP is used for the URL scheme for the top page of the Web application or Form authentication window that is the redirection destination. In this situation, by using the gateway specification function to specify that the scheme be always considered as HTTPS, you can ensure that accesses are properly redirected.

An example of the Easy Setup definition file is described below. Specify `On` in the `JkGatewayHttpsScheme` parameter so that the URL scheme for redirect destination is always considered to be HTTPS.

Example of Easy Setup definition file

```
...
<param>
 <param-name>JkGatewayHost</param-name>
 <param-value>host1</param-value>
```

```
</param>
<param>
 <param-name>JkGatewayPort</param-name>
 <param-value>4443</param-value>
</param>
<param>
 <param-name>JkGatewayHttpsScheme</param-name>
 <param-value>On</param-value>
</param>
...
```

4.10.3 Execution environment settings (When the Smart Composer functionality is not used)

This subsection describes the settings to use the gateway specification functionality.

When a gateway such as an SSL accelerator or load balancer is placed between a client and a Web server, you can use the gateway specification functionality to report the gateway information to the Web Container and can properly redirect the access to the top page of the Web application or Form authentication window.

(1) How to set

To use the gateway specification functionality:

1. Specify the gateway host name, port number, and URL scheme for redirect destination for each redirector.
2. Restart the Web server.

Specify the gateway host name, port number, and URL scheme for redirect destination in `mod_jk.conf` when using Cosminexus HTTP Server as the Web server and in `isapi_redirect.conf` when using Microsoft IIS as the Web server. The keys specified are as follows:

For details on `mod_jk.conf` (redirector operation definition file for HTTP Server), see *9.3 mod_jk.conf (redirector operation definition file for HTTP Server)* in the *uCosminexus Application Server Definition Reference Guide*.

For details on `isapi_redirect.conf` (redirector operation definition file for Microsoft IIS), see *9.2 isapi_redirect.conf (redirector operation definition file for Microsoft IIS)* in the *uCosminexus Application Server Definition Reference Guide*.

- In `mod_jk.conf`

  Host name: `JkGatewayHost` key

  Port number: `JkGatewayPort` key

  URL scheme for redirect destination: `JkGatewayHttpsScheme` key
- In `isapi_redirect.conf`

  Host name: `gateway_host` key

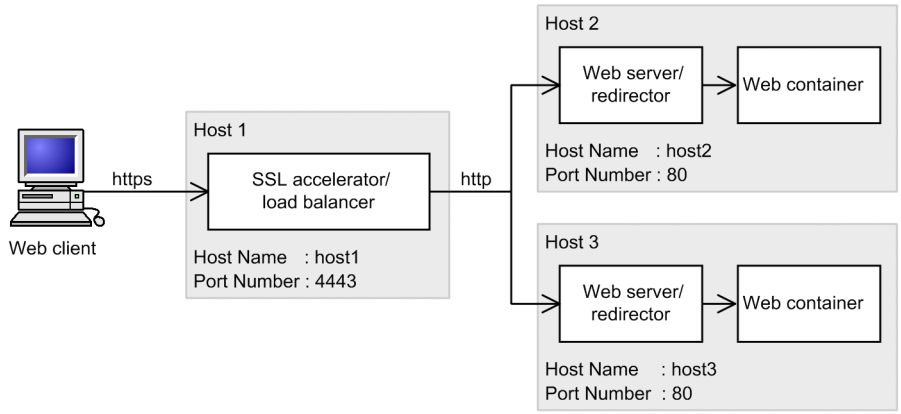
  Port number: `gateway_port` key

  URL scheme for redirect destination: `gateway_https_scheme` key

(2) Example settings

The following figure shows the example settings for the gateway specification functionality:

Figure 4-26: Example settings for the gateway specification functionality



In this example, an SSL accelerator is placed between the client and Web server. Even if a client accesses an SSL accelerator via HTTPS, the SSL accelerator accesses a Web server via HTTP, which causes the Web container to assume that the access uses HTTP. For this reason, HTTP is used for the URL scheme for the top page of the Web application or Form authentication window that is the redirection destination. In this situation, by using the gateway specification function to specify that the scheme be always considered as HTTPS, you can ensure that accesses are properly redirected.

Examples of the `mod_jk.conf` and `isapi_redirect.conf` files are shown below. Specify `On` in the `JkGatewayHttpsScheme` key of `mod_jk.conf` and `true` in the `gateway_https_scheme` key of `isapi_redirect.conf` so that the URL scheme for redirect destination is always considered to be HTTPS.

Example of mod_jk.conf (in Cosminexus HTTP Server)

```
JkGatewayHost host1
JkGatewayPort 4443
JkGatewayHttpsScheme On
```

Example of isapi_redirect.conf (In Microsoft IIS)

```
gateway_host=host1
gateway_port=4443
gateway_https_scheme=true
```

4.10.4 Precautions related to reporting the gateway information to a Web Container

The following are cautionary notes on using the gateway specification functionality:

- Specifying the host name and port number of an URL where an access is redirected:

  A browser usually sends a request with the Host header appended, so it is not necessary to specify the host name or port number for an URL where access is to be redirected.

  Note that you can check whether or not the request has the Host header by calling the `getHeader` method of the `javax.servlet.http.HttpServletRequest` class, with the Host argument specified.

- Servlet API behavior:

  Using the gateway specification functionality causes some servlet API functions to behave differently. Take care when using API functions with a Web application.

  For details on the servlet API functionality where the operations change, see *6.2.2(10) Precautions for using the gateway specification functionality*.

- The `<transport-guarantee>` tag in `web.xml`:

  When you use the gateway specification functionality to specify that a scheme is to be considered as HTTPS, a request to a Web server will be considered to use HTTPS even if the request actually uses HTTP. Note that this prevents an access from being redirected to an URL that uses HTTPS, even if you specify INTEGRAL or CONFIDENTIAL in the `<transport-guarantee>` tag in `web.xml`.

- The Secure attribute for cookies:

  When you use the gateway specification functionality to specify that a scheme is to be considered as HTTPS, when a session ID generated by a Web container is returned to the client by the session cookie, the Secure attribute is appended to the cookie.

- Communicating with the Web server without passing the gateway

  When you enable the gateway specification functionality in the redirector, you cannot perform direct HTTP communication without unless passing through the gateway, such as the SSL accelerator and load balancer, in the Web server.

5 In-Process HTTP Server

This chapter describes the settings for the in-process HTTP server functionality.

5.1 Organization of this chapter

Application Server provides an in-process HTTP server as Web server functionality. The *in-process HTTP server* is Web server functionality provided in the J2EE server processes. Since the J2EE server processing receives the HTTP request directly without passing through the Web server, you can use the Web server functionality with better processing performance than during the Web server integration.

The following table lists the functionality and the reference sections corresponding to the functionality of the in-process HTTP server:

Table 5－1: Functionality and reference sections corresponding to each functionality of in-process HTTP server

Functionality	Reference
Overview of in-process HTTP server	*5.2*
Controlling the number of connections from the Web client	*5.3*
Controlling the number of request processing threads	*5.4*
Controlling the flow of requests by controlling the number of concurrent connections from the Web client	*5.5*
Controlling the flow of requests by controlling the number of concurrently executing threads	*5.6*
Request distribution with the redirector	*5.7*
Controlling the communication with the Web client by Persistent Connection	*5.8*
Communication timeout (In-process HTTP server)	*5.9*
Specifying the IP address (In-process HTTP server)	*5.10*
Controlling access by limiting the hosts that are allowed access	*5.11*
Controlling access by limiting the request data size	*5.12*
Controlling access by limiting the HTTP-enabled methods	*5.13*
Customizing responses to the Web client using HTTP responses	*5.14*
Error page customization (in-process HTTP server)	*5.15*
Notifying the gateway information to the Web container	*5.16*
Output of log and trace	*5.17*

Note that the in-process HTTP server functionality provided in Application Server includes a functionality wherein the functions unique to Application Server are extended beyond the functions defined in J2EE, and also those provided as functions unique to Application Server. For details on whether the functionality is unique to Application Server, see *1.2 Functionality corresponding to the purpose of the system*.

5.2 Overview of in-process HTTP server

This section provides an overview of the in-process HTTP server.

The following table describes the organization of this section.

Table 5-2: Organization of this section (Overview of in-process HTTP server)

Category	Title	Reference
Description	Using the in-process HTTP server	*5.2.1*
	Functionality available in the in-process HTTP server	*5.2.2*
Settings	Execution environment settings (J2EE server settings)	*5.2.3*

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

5.2.1 Using the in-process HTTP server

The *in-process HTTP server* is the Web server functionality provided in the J2EE server processes.

As the J2EE server processing receives the HTTP request directly without passing through the Web server, you can use the Web server functionality with even better processing performance than during the Web server integration. Therefore, for a system that emphasizes performance, Hitachi recommends that you use the in-process HTTP server.

However, there are comparative differences in the functionality provided in Cosminexus HTTP Server and Microsoft IIS, you check the differences in the functionality, and then determine whether to use the in-process HTTP server. Furthermore, you cannot use the in-process HTTP server and the Web server integration functionality simultaneously. When you design the system, you must choose which functionality you will use, in advance. For details on the guidelines for selection, see the *uCosminexus Application Server System Design Guide*.

To use the in-process HTTP server, the prerequisites are as follows:

- The in-process HTTP server must be deployed within the internal network in failover instead of deploying the server in DMZ that is accessible from the external networks to which unauthorized access is assumed. In a system accessed from external networks such as Internet, you must build a system in which a proxy server is deployed on DMZ and forwarded to the in-process HTTP server within the internal network. For details on building a system when using the in-process HTTP server, see the *uCosminexus Application Server System Design Guide*.
- In the in-process HTTP server, only HTTP is supported. HTTPS is not supported. To use HTTPS, the SSL accelerator or reverse proxy of Cosminexus HTTP Server is a prerequisite.

You can use the in-process HTTP server to access only the Web applications deployed on the J2EE server. Note that you cannot deploy static contents alone, but only when you execute request distribution with the redirector and error page customization, you can specify static contents that are not included in the Web application.

Take note of the following when using the in-process HTTP server:

- If you stop the J2EE server by executing the `cjstopsv` command, during the TCP connection of the Web client and the in-process HTTP server, the J2EE server does not stop, until the Web client disconnects the TCP connection of the in-process HTTP server or the timeout specified in the `webserver.connector.inprocess_http.persistent_connection.timeout` key of `usrconf.properties` (user property file for the J2EE server) occurs. If you want to stop the J2EE server regardless of the disconnection of the TCP connection from the Web client or the timeout occurrence, forcibly stop the J2EE server by specifying the `-f` option in the `cjstopsv` command. Customize the J2EE server properties to specify the settings for the in-process HTTP server. For details on customizing the operation settings for the J2EE server, see *5.2.3 Execution environment settings (J2EE server settings)*.

  Tip

  The in-process HTTP server is not the default server.

5.2.2 Functionality available in the in-process HTTP server

The following table lists the functionality available in the in-process HTTP server and the reference section of each functionality:

Table 5-3: Functionality available in the in-process HTTP server and references

Functionality name		Reference
Controlling the number of connections from the Web client		*5.3*
Controlling the number of request processing threads from the Web client		*5.4*
Controlling the flow of requests	Controlling the number of concurrent connections from the Web client	*5.5*
	Controlling the number of concurrently executing threads#	*5.6*
Request distribution with the redirector		*5.7*
Controlling communication with the Web client	Controlling communication by Persistent Connection	*5.8*
	Communication timeout that can be set in the in-process HTTP server	*5.9*
	IP address specification used in the in-process HTTP server#	*5.10*
Controlling access from the Web client	Limiting the hosts that are allowed access	*5.11*
	Limiting the request data size	*5.12*
	Limiting the HTTP-enabled methods	*5.13*
Customizing the responses to the Web client	Customizing the HTTP response header	*5.14*
	Customizing the error page	*5.15*
Notifying the gateway information to the Web container#		*5.16*
Output of log and trace		*5.17*

#
The functionality is not different when the in-process HTTP server is not used (when the Web server integration functionality is used).

5.2.3 Execution environment settings (J2EE server settings)

This subsection describes how to set up the in-process HTTP server.

To receive HTTP requests by using the Web server functionality provided in the J2EE server processes, the system must be built with a configuration using the in-process HTTP server functionality. For details on the system configuration and building with the in-process HTTP server functionality, see the *uCosminexus Application Server System Setup and Operation Guide*.

Procedure:

1. Enabling the in-process HTTP server functionality.
   Define the specifications for the in-process HTTP server by specifying `true` in the `webserver.connector.inprocess_http.enabled` parameter in the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file. By default, `false` is specified. For details on the Easy Setup definition file and the parameters to be specified, see the *uCosminexus Application Server Definition Reference Guide*.
2. Specify settings for controlling the number of connections from the Web client and controlling the number of request processing threads.

By adjusting the number of request processing threads according to the performance of the host that operates the server and the status of access from the client, you can improve the performance of the in-process HTTP server. For details on settings, see *5.3 Controlling the number of connections from the Web client* and *5.4 Controlling the number of request processing threads*.

Note the following when you set up the in-process HTTP server:

- If a large number of requests need to be processed immediately after the server starts, specify a big value as the number of request processing threads to be created when the server starts.
- Note that if you increase the maximum number of spare threads, you can promptly support sudden increase in access, but a lot of resources will be consumed.

3. Specify settings for controlling access from the Web client.

   By enhancing the security for connections and requests sent from the client, you can prevent unauthorized access and attacks on the server from outside. For details on settings, see *5.11 Controlling access by limiting the hosts that are allowed access*, *5.12 Controlling access by limiting the request data size*, and *5.13 Controlling access by limiting the HTTP-enabled methods*.

4. As and when required, specify settings for the functionality that can be used in the in-process HTTP server.

   For details on the functionality available in the in-process HTTP server, see *5.2.2 Functionality available in the in-process HTTP server*.

5.3 Controlling the number of connections from the Web client

By controlling the number of connections and number of request processing threads from the Web client and by optimizing the number of request processing threads, you can constantly control the load on the J2EE server and maintain a stable and high throughput. For details on controlling the number of request processing threads, see *5.4 Controlling the number of request processing threads*.

This section describes the controlling of the number of connections from the Web client.

The following table describes the organization of this section.

Table 5-4: Organization of this section (Controlling the number of connections from the Web client)

Category	Title	Reference
Description	Overview of controlling the number of connections from the Web client	*5.3.1*
Settings	Execution environment settings (J2EE server settings)	*5.3.2*

Note:

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

In the in-process HTTP server, you can control the number of request processing threads created by the in-process HTTP server by setting the number of Web clients that can connect simultaneously. Also, by pooling the constant number of request processing threads that are not running as the spare threads, the processing required for adding and deleting the request processing threads is restrained to the minimum.

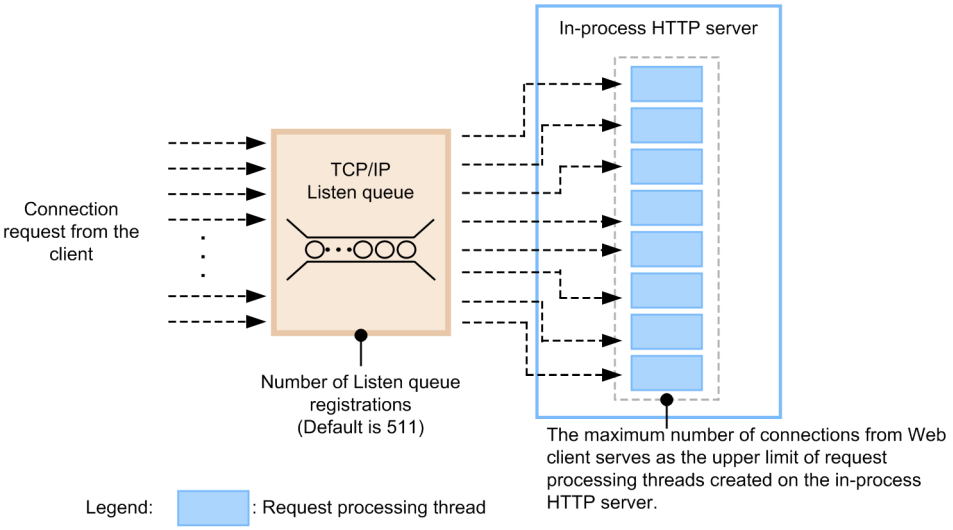
5.3.1 Overview of controlling the number of connections from the Web client

In the in-process HTTP server, you set the maximum number of Web clients and proxy servers connecting simultaneously and control the number of request processing threads. The in-process HTTP server creates the request processing threads for the number of connections from the Web client, and therefore, the maximum number of connections from the Web client becomes the upper limit for the number of request processing threads that are created by the in-process HTTP server.

Note that the connection requests from the client are registered in the TCP/IP Listen queue and are passed to the request processing threads. The connection requests from the client exceeding the upper limit for the number of connections are accumulated in the Listen queue. When the connection requests from the client accumulated in the Listen queue exceed the specified maximum value, the client fails to connect to the server.

The following figure shows an overview of controlling the number of connections from the Web client:

Figure 5-1: Overview of controlling the number of connections from the Web client



5.3.2 Execution environment settings (J2EE server settings)

Specify the definition for controlling the number of connections from the Web client in the <configuration> tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file.

The following table lists the definition in the Easy Setup definition file for controlling the number of connections from the Web client:

Table 5-5: Definition in the Easy Setup definition file for controlling the number of connections from the Web client

Parameter to be specified	Setting contents
webserver.connector.inprocess_http.max_connections	Specifies the maximum number of connections with the Web client and proxy server. The in-process HTTP server creates the request processing threads for the number of connections from the Web client, and therefore, the value specified here becomes the upper limit for the number of request processing threads.
webserver.connector.inprocess_http.backlog	The HTTP requests exceeding the upper limit for the number of connections from the Web client are accumulated in the Listen queue. Specify the maximum number of registrations in the Listen queue in this parameter.

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

5.4 Controlling the number of request processing threads

By controlling the number of connections and number of request processing threads from the Web client and by optimizing the number of request processing threads, you can constantly control the load on the J2EE server and maintain a stable and high throughput. For details on controlling the number of connections from the Web client, see *5.3 Controlling the number of connections from the Web client*.

This section describes the controlling of the number of request processing threads. The following table describes the organization of this section.

Table 5-6: Organization of this section (Controlling the number of request processing threads)

Category	Title	Reference
Description	Overview of controlling the number of request processing threads	*5.4.1*
Settings	Execution environment settings (J2EE server settings)	*5.4.2*

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

5.4.1 Overview of controlling the number of request processing threads

After the request processing threads are created when the in-process HTTP server starts, the status of the request processing threads and the number of threads are monitored periodically. When requests are concentrated in the in-process HTTP server, the request processing threads are added and the adequate number of spare threads is pooled in advance. When there are few requests, the extra pooled spare threads will be deleted.

The controlling of the number of request processing threads is executed as follows:

1. When the J2EE server starts, the specified number of request processing threads is created.
2. While the J2EE server is running, the number of request processing threads is monitored.
3. During monitoring, if the number of spare threads is smaller than the specified minimum value, the request processing threads are added and pooled as the spare threads. Also, if the number of spare threads is greater than the specified maximum value, the extra spare threads are deleted.

   Note that you can also maintain the number of threads created when the J2EE server starts. When maintaining the number of threads created at startup, if the total number of request processing threads and spare threads is less than the number of threads created when the Web server starts, even if the number of spare threads exceeds the maximum value, the spare threads are not deleted. For example, if the number of threads created when the Web server starts is 8 and the maximum number of spare threads is 5, the spare threads are not deleted in the case when the number of request processing threads is 2 and the number of spare threads is 6.

The transition of the number of request processing threads is explained with the following examples:

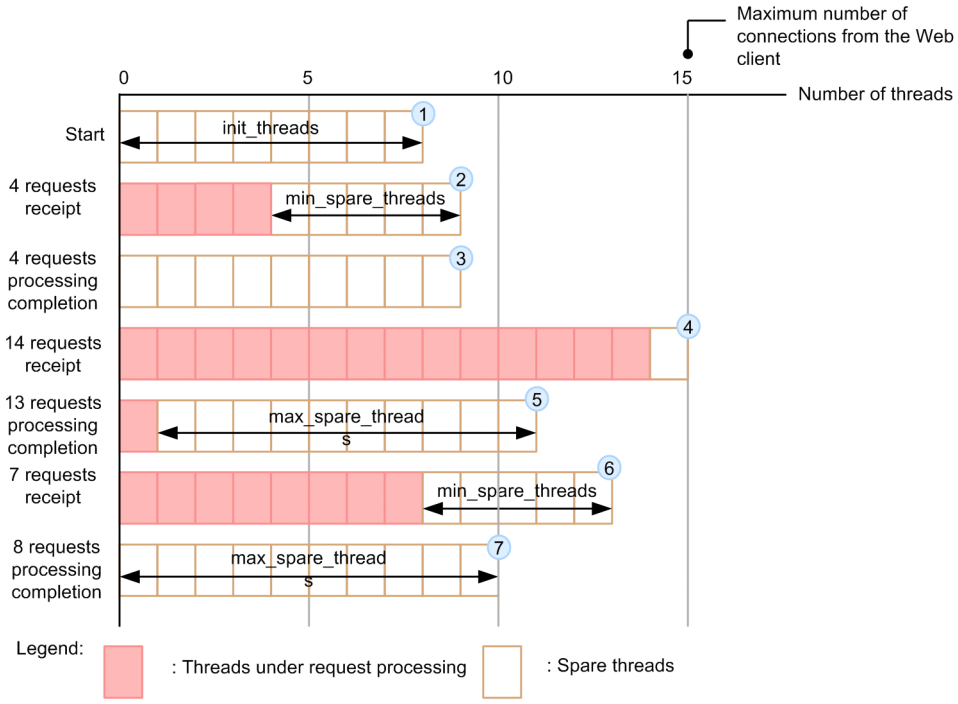
- **Transition example 1**

  Assume the following settings:

  - Maximum number of connections from the Web client: 15
  - Number of registrations in the Listen queue: 100
  - Number of request processing threads created when the J2EE server starts: 8
  - Minimum number of spare threads: 5
  - Maximum number of spare threads: 10
  - Maintenance of the number of threads created when the J2EE server starts: Disabled

  The following figure shows the transition example for the number of request processing threads:

Figure 5-2: Transition example for the number of request processing threads



Stages 1. to 7. of the figure are explained below:

1. When the J2EE server starts, the specified number (8 threads) of request processing threads is created.
2. When 4 requests are received, the number of spare threads becomes 4 and since this number is less than the minimum value, 1 thread is added.
3. When the processing of the 4 requests ends, the number of spare threads becomes 9. Since this number is less than the maximum value and more than the minimum value, the current state is maintained.
4. When 14 requests are received, the number of spare threads is less than the minimum value, but the maximum number of connections from the Web client is reached, therefore, the spare threads add only 1 thread.
5. When the processing of 13 requests ends, the number of spare threads exceeds the maximum value, so 4 threads are deleted.
6. When 7 requests are received, the number of spare threads is less than the minimum value, so 2 threads are added.
7. When the processing of 8 requests ends, the number of spare threads exceeds the maximum value, so 3 threads are deleted.

- **Transition example 2**

  By setting the maximum number of spare threads equal to the maximum number of connections from the Web client, you can continue to use the request processing threads created once without deleting them.

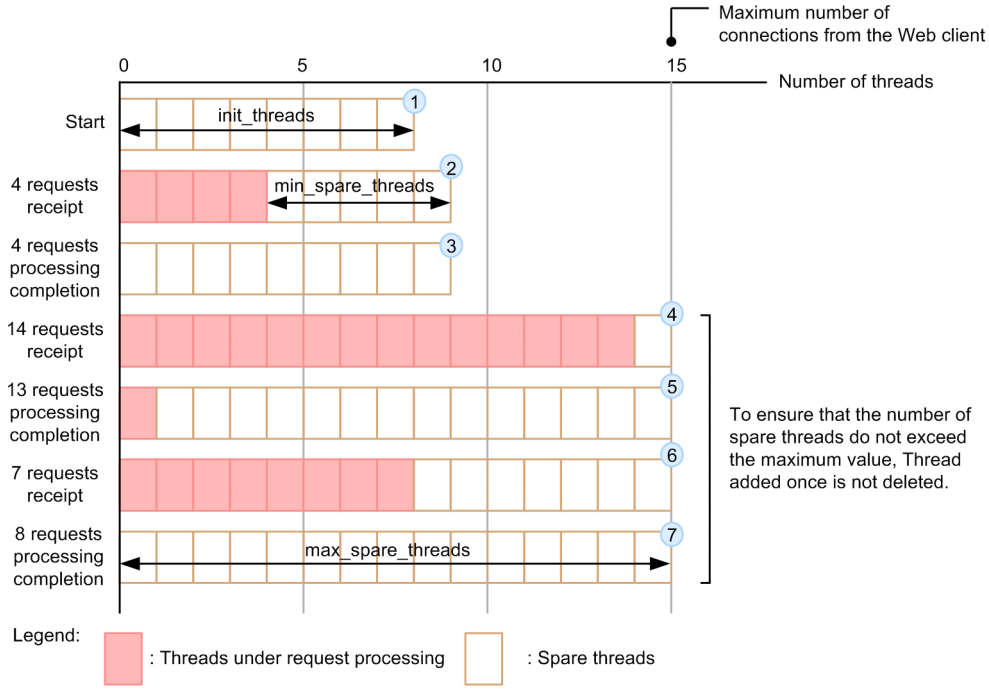
  The transition example for the number of request processing threads when the maximum number of spare threads is equal to the maximum number of connections from the Web client is as follows:

  Assume the following settings:

  - Maximum number of connections from the Web client: 15
  - Maximum number of registrations in the Listen queue: 100
  - Number of request processing threads created when the J2EE server starts: 8
  - Minimum number of spare threads: 5
  - Maximum number of spare threads: 15
  - Maintenance of the number of threads created when the J2EE server starts: Disabled

  The following figure shows the transition example for the number of request processing threads, when the maximum number of spare threads is equal to the maximum number of connections from the Web client:

Figure 5-3: Transition example for the number of request processing threads when the maximum number of spare threads is equal to the maximum number of connections from the Web client



Stages 1. to 7. of the figure are explained below:

1. When the J2EE server starts, the specified number (8 threads) of request processing threads is created.
2. When 4 requests are received, the number of spare threads becomes 4 and since this number is less than the minimum value, 1 thread is added.
3. When the processing of the 4 requests ends, the number of spare threads becomes 9. Since this number is less than the maximum value and more than the minimum value, the current state is maintained.
4. When 14 requests are received, the number of spare threads is less than the minimum value, but so that the total number of request processing threads does not exceed the maximum number of connections from the Web client, spare threads add only 1 thread.
5. When the processing of 13 requests ends, the number of spare threads becomes 14, but this number is less than the maximum value and more than the minimum value, so the current state is maintained.
6. When 7 requests are received, the number of spare threads becomes 7, but this number is less than the maximum value and more than the minimum value, so the current state is maintained.
7. When the processing of 8 requests ends, the number of spare threads becomes 15, but this number is less than the maximum value and more than the minimum value, so the current state is maintained.

- **Setup example-3**

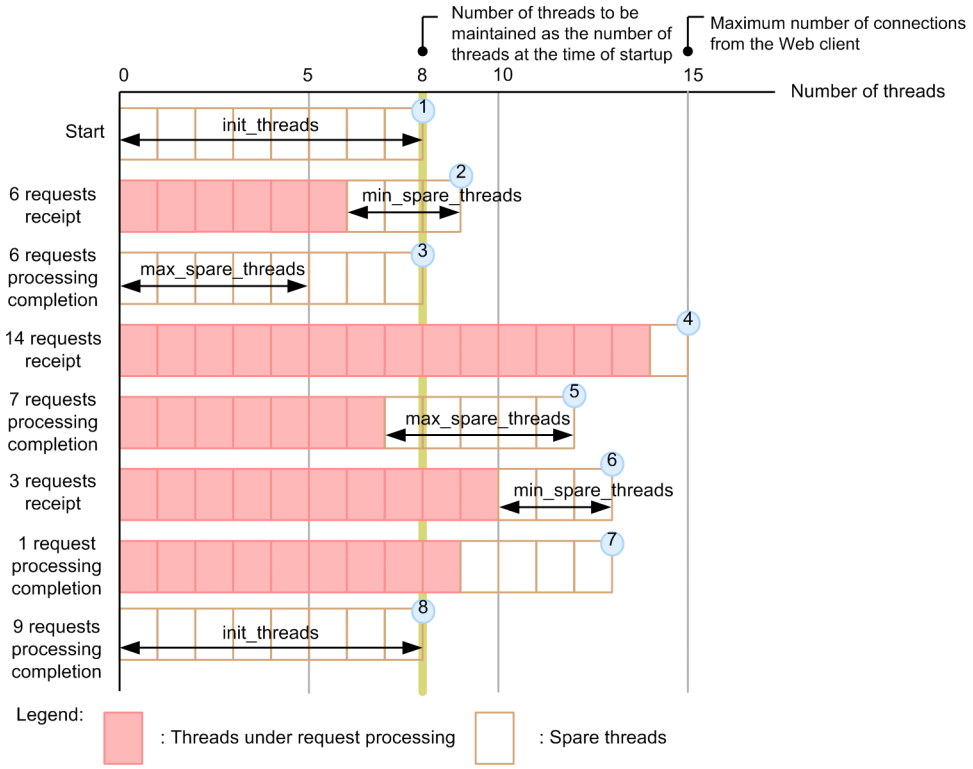
  The following is a transition example for the number of request processing threads when the number of threads created at server startup is maintained:

  Assume the following settings:

  - Maximum number of connections from the Web client: 15
  - Maximum number of registrations in the Listen queue: 100
  - Number of request processing threads created when the J2EE server starts: 8
  - Minimum number of spare threads: 3
  - Maximum number of spare threads: 5
  - Maintenance of the number of threads created when the J2EE server starts: Enabled

  The following figure shows the transition example for the number of request processing threads when the number of threads created at server startup is maintained:

Figure 5-4: Transition example for the number of request processing threads when the number of threads created at J2EE server startup is maintained



Stages 1. to 8. of the figure are explained below:

1. When the J2EE server starts, the specified number (8 threads) of request processing threads is created.
2. When 6 requests are received, the number of spare threads becomes 2. Since this number is less than the minimum value, 1 thread is added.
3. When the processing of the 6 requests ends, the number of spare threads becomes 9 and the maximum value is exceeded, but in order to maintain the number of threads created at server startup, only 1 thread is deleted.
4. When 14 requests are received, the number of spare threads is less than the minimum value, but so that the total number of request processing threads does not exceed the maximum number of connections from the Web client, spare threads add only 1 thread.
5. When the processing of 7 requests ends, the number of spare threads becomes 8 and exceeds the maximum value, so 3 threads are deleted.
6. When 3 requests are received, the number of spare threads becomes 2 and is less than the minimum value, so 1 thread is added.
7. When the processing of 1 request ends, the number of spare threads becomes 4, but this number is less than the maximum value and more than the minimum value, so the current status is maintained.
8. When the processing of 9 requests ends, the number of spare threads becomes 13. This number is more than the maximum value, but in order to maintain the number of threads at J2EE server startup, only 5 threads are deleted.

5.4.2 Execution environment settings (J2EE server settings)

Specify the definition for controlling the number of request processing threads in the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file.

The following table lists the definitions in the Easy Setup definition file for controlling the number of request processing threads:

Table 5-7: Definitions in the Easy Setup definition file for controlling the number of request processing threads

Parameter to be specified	Setting contents
`webserver.connector.inprocess_http.init_threads`	Specify the number of request processing threads created when the J2EE server starts.
`webserver.connector.inprocess_http.min_spare_threads`	Specify the minimum number of spare threads. If the number of spare threads is less than the specified minimum value, the request processing threads are added and are pooled as the spare threads.
`webserver.connector.inprocess_http.max_spare_threads`	Specify the maximum number of spare threads. If the number of spare threads is more than the specified maximum value, the surplus spare threads are deleted. By setting the maximum number of spare threads equal to the maximum number of connections from the Web client, you can continue to use the request processing threads created once without deleting them.
`webserver.connector.inprocess_http.keep_start_threads`	Specify whether or not to maintain the request processing threads created when the J2EE server starts.

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

5.5 Controlling the flow of requests by controlling the number of concurrent connections from the Web client

This section describes the controlling of the flow of requests by controlling the number of concurrent connections from the Web client.

The following table describes the organization of this section.

Table 5-8: Organization of this section (Controlling the flow of requests by controlling the number of concurrent connections from the Web client)

Category	Title	Reference
Description	Controlling the number of concurrent connections from the Web client	*5.5.1*
Settings	Execution environment settings (J2EE server settings)	*5.5.2*

Note:

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

5.5.1 Controlling the number of concurrent connections from the Web client

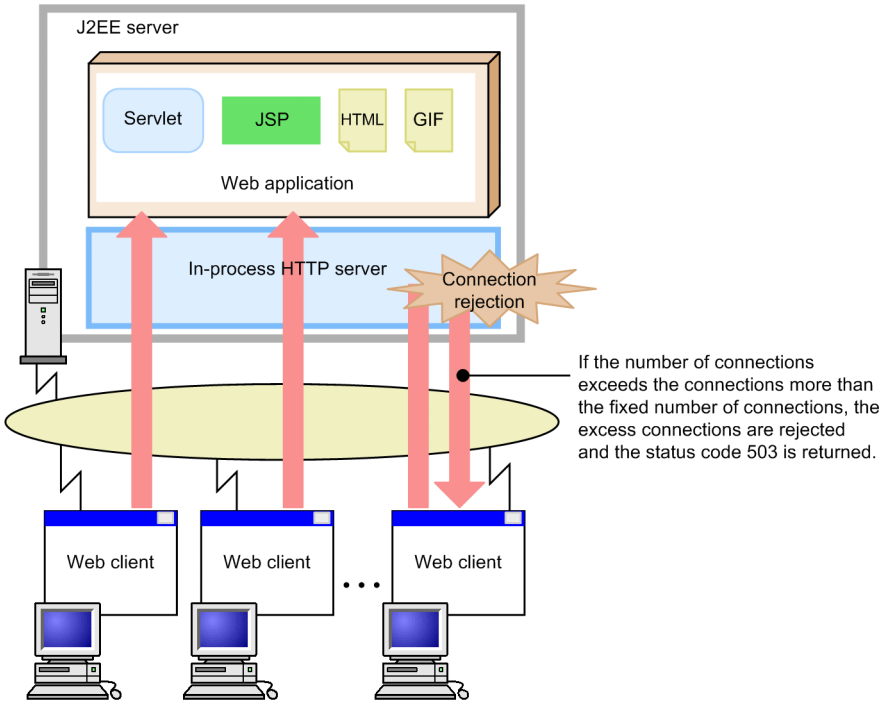
In the in-process HTTP server, you control the number of concurrent connections from the Web client by specifying the maximum number of connections from the Web client along with the number of requests for which connection is rejected.

When the number of connections from the Web client increases or if the load on the J2EE server increases with the effect of the J2EE applications, the Web client can instantly receive a response by rejecting the receipt of requests from the Web client and by returning an error immediately. As a result, the load on the J2EE server is controlled constantly and the response time for the request can be maintained.

The value obtained by subtracting the number of requests for which the connection is rejected from the maximum number of connections by the Web client is the number of requests for which connection is approved by the Web client.

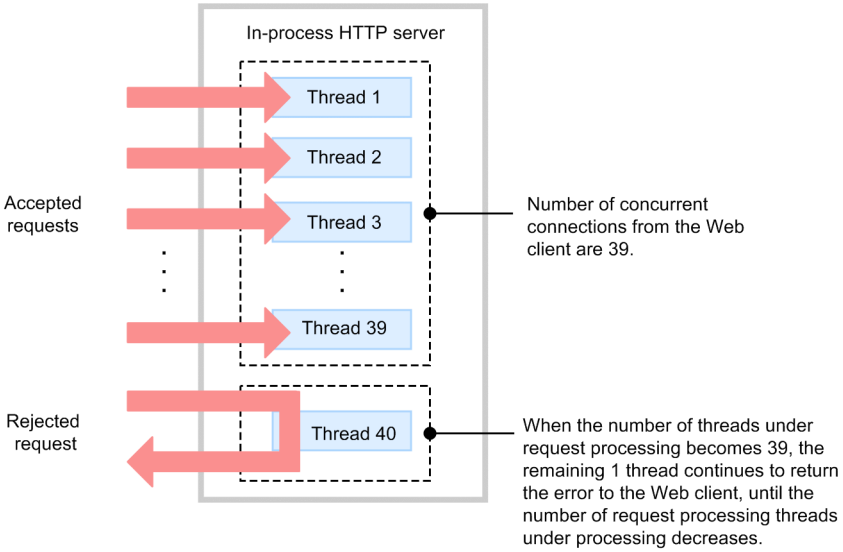
The following figure shows an overview of controlling the number of concurrent connections from the Web client:

Figure 5-5: Overview of controlling the number of concurrent connections from the Web client



For example, assuming the maximum number of connections from the Web client is 40 and the number of requests for which connection is rejected is 1, the number of Web clients that can concurrently process the requests by connecting to the in-process HTTP server is 39. If the number of threads that are processing the requests becomes 39, the remaining 1 thread continues to return an error for the received requests until the number of request processing threads (under processing) will reduce. The following figure shows an example of controlling the number of concurrent connections from the Web client:

Figure 5-6: Example of controlling the number of concurrent connections from the Web client



By controlling the number of concurrent connections from the Web client, an error with the status code 503 is returned to the Web client for the request for which connection is rejected. At this time, if you customize the error page returned to the client, you can customize the response message or redirect to another server.

For details on the settings for customizing the responses to the Web client (in the in-process HTTP server), see *5.14 Customizing responses to the Web client using HTTP responses* and *5.15 Error page customization (In-process HTTP server)*.

Important note

When displaying a page with a frame or a page with an inserted image, the multiple requests might be received from the Web client. In this case, the pages displayed by controlling the number of concurrent connections from the Web client might result in partial errors.

5.5.2 Execution environment settings (J2EE server settings)

To control the number of concurrent connections from the Web client, you must set a J2EE server.

The setting method and example of controlling the number of concurrent connections from the Web client are described here.

(1) How to set

Specify the definition for controlling the number of concurrent connections from the Web client in the following parameter in the <configuration> tag of the logical J2EE server (j2ee-server), in the Easy Setup definition file:

webserver.connector.inprocess_http.rejection_threads

Specifies the number of requests for which connection is rejected.

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

Important note

When displaying a page with a frame or a page with an image inserted, multiple requests might be received from the Web client. In this case, the pages displayed by the controlling of the number of concurrent connections from the Web client might result in partial errors.

(2) Example settings

The example of settings for controlling the number of concurrent connections from the Web client is described here.

The following is an example of settings wherein the upper limit of number of request processing threads is 40 and the number of request processing threads for which the connection is rejected is 1:

```
...
<param>
 <param-name>webserver.connector.inprocess_http.max_connections</param-name>
 <param-value>40</param-value>
</param>
<param>
 <param-name>webserver.connector.inprocess_http.rejection_threads</param-name>
 <param-value>1</param-value>
</param>
...
```

In this example, the number of Web clients that can process requests concurrently after connection is 39. If the number of threads that are processing requests reaches 39, the remaining 1 thread keeps returning error to the Web client.

By controlling the number of concurrent connections from the Web client, an error of the status code 503 (Service Unavailable) is returned to the Web client for the requests for which connection is rejected. At this time if you customize the error page returned to the client, you can customize the response message or redirect to another server. The following are the setting examples for each of these cases. For details on customizing the error page, see *5.15 Error page customization (In-process HTTP server)* .

- **To customize the response message**

  The following is an example of settings for returning a specific file as the response body to the Web client when the connection from the Web client is rejected:

```
...
<param>
 <param-name>webserver.connector.inprocess_http.rejection_threads</param-name>
```

```
 <param-value>3</param-value>
</param>
<param>
 <param-name>webserver.connector.inprocess_http.error_custom.list</param-name>
 <param-value>REJECTION_1</param-value>
</param>
<param>
 <param-name>webserver.connector.inprocess_http.error_custom.REJECTION_1.status</
param-name>
 <param-value>503</param-value>
</param>
<param>
 <param-name>webserver.connector.inprocess_http.error_custom.REJECTION_1.file</
param-name>
 <param-value>C: /data/busy.html</param-value>
</param>
<param>
 <param-
name>webserver.connector.inprocess_http.error_custom.REJECTION_1.file.content_type=text/
html; charset</param-name>
 <param-value>ISO-8859-1</param-value>
</param>
<param>
 <param-
name>webserver.connector.inprocess_http.error_custom.REJECTION_1.request_url</
param-name>
 <param-value>/*</param-value>
</param>
...
```

In this example, '503' is specified in the error status code, `C:/data/busy.html` is specified in the corresponding error page file, 'text/html; charset=ISO-8859-1 (Media-Type is text/html and ISO-8859-1 character set is used)' is specified in the Content-Type header of the response, and `/*` is specified in the URL pattern. Therefore, when error status code '503' occurs, regardless of the request URI, the contents of `C:/data/busy.html` file are returned as a response.

- **To redirect a request to another server**

  The following is an example of settings for redirecting a request for which connection is rejected to another server:

```
...
<param>
 <param-name>webserver.connector.inprocess_http.rejection_threads</param-name>
 <param-value>3</param-value>
</param>
<param>
 <param-name>webserver.connector.inprocess_http.error_custom.list</param-name>
 <param-value>REJECTION_1</param-value>
</param>
<param>
 <param-name>webserver.connector.inprocess_http.error_custom.REJECTION_1.status</
param-name>
 <param-value>503</param-value>
</param>
<param>
 <param-
name>webserver.connector.inprocess_http.error_custom.REJECTION_1.redirect_url</
param-name>
 <param-value>http: //host1/busy.html</param-value>
</param>
<param>
 <param-
name>webserver.connector.inprocess_http.error_custom.REJECTION_1.request_url</
param-name>
 <param-value>/*</param-value>
</param>
<param>
...
```

In this example, '503' is specified in error status code, `http://host1/busy.html` is specified in the corresponding error page file, and `/*` is specified in the URL pattern. Therefore, when error status code '503' occurs, regardless of the request URI, all the requests are redirected to the `http://host1/busy.html` URL.

5.6 Controlling the flow of requests by controlling the number of concurrently executing threads

This section describes the controlling the flow of requests by controlling the number of concurrently executing threads.

The following table describes the organization of this section.

Table 5-9: Organization of this section (Controlling the flow of requests by controlling the number of concurrently executing threads)

Category	Title	Reference
Description	Overview of controlling the flow of requests by controlling the number of concurrently executing threads	*5.6.1*
Settings	Execution environment settings (J2EE server settings)	*5.6.2*

Note:

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

Reference note

There is no difference in the functionality when in-process HTTP server is not used (when the Web server integration functionality is used). For explanation of this functionality, see *2.15 Overview of controlling the number of concurrently executing threads*.

5.6.1 Overview of controlling the flow of requests by controlling the number of concurrently executing threads

In the Web container, the servlet requests are processed in multi-threads. At this time, you can set an upper limit on the number of threads that can be executed concurrently for avoiding the decrease in performance due to slashing. If you set an appropriate number of threads, you can tune the performance as per the access status.

5.6.2 Execution environment settings (J2EE server settings)

To control the number of concurrently executing threads in the Web container, you must set up a J2EE server.

This subsection describes the settings for controlling the number of concurrently executing threads in the Web container.

The methods for controlling the number of concurrently executing threads include the controlling in Web container, in Web application, and in URL group.

(1) Controlling in the Web container

To control the number of concurrently executing threads in the Web container, set the maximum number of threads that can be executed concurrently in the entire Web container. The number of threads set here is shared in all the Web applications deployed on the Web container.

Specify the definition for controlling the number of concurrently executing threads in the Web container in the following parameter within the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file.

webserver.connector.inprocess_http.max_execute_threads

Set the maximum number of threads that you can concurrently execute in the entire Web container.

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

(2) Controlling in the Web application

Note that when you control the number of concurrently executing threads in the Web application, you also need to simultaneously specify the settings to control the number of threads in the Web container.

Specify settings for controlling the number of concurrently executing threads in the Web application using the Easy Setup definition file and the server management commands. The method of setup is the same as is used for the Web server integration functionality. For details on how to set up the number of concurrently executing threads in the Web applications when the Web server integration functionality is used, see *2.17 Controlling the number of concurrently executing threads in the Web application*.

(3) Controlling in the URL group

To control the number of concurrently executing threads in the URL group, you must also specify settings for controlling the number of concurrently executing threads in the Web application and the settings for controlling the number of threads in the Web container simultaneously.

Specify settings for controlling the number of concurrently executing threads in the URL group by using the server management commands. The method of setup is the same as is used for the Web server integration functionality. For details on how to set the number of concurrently executing threads in the URL group when the Web server integration functionality is used, see *2.18 Controlling the number of concurrently executing threads in the URL group*.

5.7 Request distribution with the redirector

This section explains the distribution of requests with the redirector.

In the in-process HTTP server, you can distribute the requests by the URL patterns included in an HTTP request. You can also customize the responses for the distributed requests and return them to the client.

This section describes the distribution of requests by the URL pattern and the process of customizing the responses. The section also provides an overview of the settings for request distribution with the redirector.

The following table describes the organization of this section.

Table 5-10: Organization of this section (Request distribution with the redirector)

Category	Title	Reference
Description	Distributing requests by URL pattern	*5.7.1*
	Response customization	*5.7.2*
Settings	Execution environment settings (J2EE server settings)	*5.7.3*
Notes	Precautions related to request distribution with the redirector	*5.7.4*

Note:
There is no specific description of *Implementation* and *Operations* for this functionality.

5.7.1 Distributing requests by URL pattern

In the in-process HTTP server, of the HTTP requests for the in-process HTTP server, you can distribute and process requests for a specific URL to a specified Web container. As a result, when the Web application is moved to another J2EE server due to the reasons such as changes in the system configuration, the request to the old URL can be forwarded to the new URL.

Also, in the distribution of requests by redirecting to the in-process HTTP server, the requests for specific Web applications and specific servlets and JSPs in the Web application can be temporarily redirected to another Web server. In the in-process HTTP server, the requests are redirected regardless of whether the resources for the requested servlets and JSPs actually exist. The redirection is given a higher priority than the servlets and JSPs. Therefore, when the requests to the servlets and JSPs match with the redirected URL, the servlets and JSPs are not executed.

5.7.2 Response customization

You can also customize a specific file to return as a response for the requests to a specific URL. If the status code of the response for the request to a redirected URL is 300 to 307, the response body is auto-generated and the response is returned to the client. You can also use a specified file as the response body. When you specify the file, specify the Content-Type header of the response as well.

For details on the status code for which the response is auto-generated and for the settings for request distribution with the redirector (in the in-process HTTP server), see *5.7.3 Execution environment settings (J2EE server settings)*.

5.7.3 Execution environment settings (J2EE server settings)

This subsection describes the settings for distributing requests with the redirector.

(1) Overview

In the in-process HTTP server, you can distribute requests by the URL pattern included in the HTTP request. You can also customize the response for the distributed request and return a specific file to the client. When the response status code for the request to a redirected URL is 300, the response body is auto-generated and the response is returned to the client. Also, you can use a specified file as a response body. When you specify the file, also specify the Content-Type header of the response.

The auto-generated response body is as follows:

```
<HTML><HEAD>
<TITLE>Status code and explanation</TITLE>
</HEAD><BODY>
<H1>Status code and explanation</H1>
</BODY></HTML>
```

The status code for which the response body is auto-generated and the description is as follows:

- 300 Multiple Choices
- 301 Moved Permanently
- 302 Found
- 303 See Other
- 305 Use Proxy
- 307 Temporary Redirect

When the reading of the file used as the response body during request processing fails, if 300 is specified as the status code, the response body is auto-generated and returned to the client. If 200 is specified as the status code, status 500 error is returned to the client.

(2) How to set

Specify the definition for distributing requests with the redirector in the <configuration> tag of the logical J2EE server (j2ee-server), in the Easy Setup definition file.

The following table lists the definitions in the Easy Setup definition file for distributing requests with the redirector:

Table 5-11: Definitions in the Easy Setup definition file for distributing requests with the redirector

Parameter to be specified	Setting contents
webserver.connector.inprocess_http.redirect.list	Specifies the redirect definition name.
webserver.connector.inprocess_http.redirect.*redirect-difinition-name*.request_url	Specifies the URL of the redirected request as an absolute path beginning with a slash (/).
webserver.connector.inprocess_http.redirect.*redirect-difinition-name*.redirect_url	Specifies the URL for redirecting the request. Note that when 200 is specified as the status code, the URL cannot be specified.
webserver.connector.inprocess_http.redirect.*redirect-difinition-name*.status	Specifies the response status code used when redirection is executed.
webserver.connector.inprocess_http.redirect.*redirect-difinition-name*.file	Specifies the file to be used as the response body when a specific file is returned to the client as a response. Note that when 200 is specified as the status code, the file to be used must be specified.
webserver.connector.inprocess_http.redirect.*redirect-difinition-name*.file.content_type	Specifies the Content-Type header of the file to be used as the response body when a specific file is returned as a response to the client.

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

! Important note

- When the HTTP response is an HTML file, if another file (such as an image file) is being referenced from that HTML, the file might not be properly displayed in the browser.
- If the value specified in the redirect URL matches with the value specified for the request URL, note that the client keeps redirecting the requests.

- When a session is managed by URL rewriting, the session cannot be inherited even if the request is redirected to the same Web application as the request URL.

(3) Example settings

An example of settings for request distribution with the redirector is as follows:

- **Setup example-1**

  An example of request distribution with the redirector is as follows:

```
...
<param>
 <param-name>webserver.connector.inprocess_http.redirect.list</param-name>
 <param-value>REDIRECT_1, REDIRECT_2</param-value>
</param>
<param>
<param-name>webserver.connector.inprocess_http.redirect.REDIRECT_1.request_url</
param-name>
 <param-value>/index.html</param-value>
</param>
<param>
<param-name>webserver.connector.inprocess_http.redirect.REDIRECT_1.redirect_url</
param-name>
 <param-value>http: //host1/new_dir/index.html</param-value>
</param>
<param>
<param-name>webserver.connector.inprocess_http.redirect.REDIRECT_1.status</param-
name>
 <param-value>302</param-value>
</param>
<param>
 <param-name>webserver.connector.inprocess_http.redirect.REDIRECT_2.request_url</
param-name>
 <param-value>/old_dir/*</param-value>
</param>
<param>
<param-name>webserver.connector.inprocess_http.redirect.REDIRECT_2.redirect_url</
param-name>
 <param-value>http: //host1/new_dir/</param-value>
</param>
<param>
<param-name>webserver.connector.inprocess_http.redirect.REDIRECT_2.status</param-
name>
 <param-value>301</param-value>
</param>
<param>
<param-name>webserver.connector.inprocess_http.redirect.REDIRECT_2.file</param-
name>
 <param-value>C: /data/301.html</param-value>
</param>
<param>
<param-
name>webserver.connector.inprocess_http.redirect.REDIRECT_2.file.content_type</
param-name>
 <param-value>text/html; charset=ISO-8859-1</param-value>
</param>
...
```

  In this example, `REDIRECT_1` and `REDIRECT_2` are used as the redirect definition names. In `REDIRECT_1`, the request to `/index.html` is redirected to `http://host1/new_dir/index.html` with status code '302'. In `REDIRECT_2`, the requests to `/old_dir/` are redirected under `http://host1/new_dir/` with status '301'. Also, `C:/data/301.html` is used as the response body and `text/html; charset=ISO-8859-1` is used as the Content-Type header.

- **Setup example-2**

  When a wild card is used as a request URL and a value ending with slant (/) is specified in the redirect URL, the value set in the Location header of the response becomes 'Value specified in the redirect URL' + 'Actual path from the wild card of the request URL'.

```
...
<param>
 <param-name>webserver.connector.inprocess_http.redirect.list</param-name>
 <param-value>REDIRECT_3</param-value>
</param>
<param>
```

```
<param-name>webserver.connector.inprocess_http.redirect.REDIRECT_3.request_url</
param-name>
 <param-value>/dir1/*</param-value>
</param>
<param>
```

```
<param-name>webserver.connector.inprocess_http.redirect.REDIRECT_3.redirect_url</
param-name>
 <param-value>http: //host/dir2/</param-value>
</param>
...
```

In this example, when the actual request URL is `/dir1/subdir1/index.html`, `http://host/dir2/subdir1/index.html` is set in the Location header. Note that even when a wild card is used in the request URL, if the redirect URL does not end with slant (/), the value of the Location header is the same as the redirect URL.

When redirect is executed, if a query string is added to the actual request URL, the value set in the Location header is a value wherein the query string is added to the value specified for the redirect URL. You can also specify a value with the query string added in the redirect URL. In this case, if a query string is also added to the actual request URL, the value set in the Location header is a value wherein the request query string is added behind the value specified for the redirect URL.

5.7.4 Precautions related to request distribution with the redirector

The precautions related to request distribution with the redirector are as follows:

- When the HTTP response is an HTML file, if another file (such as an image file) is being referenced from that HTML, the file might not be properly displayed in the browser.
- If the value specified in the redirect URL matches with the value specified for the request URL, note that the client keeps redirecting the requests.
- When a session is managed by URL rewriting, the session cannot be inherited even if the request is redirected within the Web application.

5.8 Controlling the communication with the Web client by persistent connection

This section describes the controlling of communication with the Web client by Persistent Connection.

The following table describes the organization of this section.

Table 5-12: Organization of this section (Controlling communication with the Web client by Persistent Connection)

Category	Title	Reference
Description	Controlling communication by Persistent Connection	*5.8.1*
Settings	Execution environment settings (J2EE server settings)	*5.8.2*

Note:

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

5.8.1 Controlling communication by Persistent Connection

The *persistent connection* is functionality used for connecting the TCP connection established between the Web client and the in-process HTTP server and keep using the TCP connection between multiple HTTP requests. By using the persistent connection, the time required for establishing a connection between the Web client and Web server is reduced and an attempt is made to reduce the processing time and lessen the communication traffic.

In the in-process HTTP server, you set up the following items to control communication by the Persistent Connection:

- **Upper limit for the number of persistent connections**

  By setting the upper limit for the number of persistent connections, you control the number of Web clients that can continuously process requests with one TCP connection. If the number of TCP connections exceeds the specified upper limit, the connection disconnects after the processing of the request ends. As a result, you can secure the threads for processing the new requests and prevent the request processing threads from being used exclusively by a specific client.

- **Upper limit for the request processing frequency of persistent connection**

  By setting the maximum value for the request processing frequency of the persistent connection, you can control the processing when there are continuous requests from the same Web client.

  When the request processing frequency of the persistent connection exceeds the specified upper limit, the connection disconnects after the processing of the request ends. As a result, you can prevent the request processing threads from being used exclusively by a specific client.

- **Persistent connection timeout**

  By setting a timeout for the request waiting time of the persistent connection, you control the request waiting time of the persistent connection. If there is no request to process requests until the specified timeout period expires, the TCP connection disconnects. As a result, you can prevent the TCP connection from being continuously occupied in an unused state. Also, even if you specify 0 for the request waiting time of Persistent Connection so that a timeout does not occur, if the number of requests exceeds the upper limit on the number of requests that can be processed, the connection is disconnected.

Note that a disconnected Web client will try to connect and send requests again.

5.8.2 Execution environment settings (J2EE server settings)

To use controlling of communication by the persistent connection, you must set up the J2EE server.

This subsection describes the settings and examples for controlling communication by the persistent connection.

(1) Setting up the J2EE server

Implement the J2EE server settings in the Easy Setup definition file. Specify the definition for controlling communication by the persistent connection in the <configuration> tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file.

The following table lists the definitions in the Easy Setup definition file for controlling communication by the persistent connection.

Table 5-13: Definitions in the Easy Setup definition file for controlling communication by persistent connection

Parameter to be specified	Setting contents
webserver.connector.inprocess_http.persistent_connection.max_connections	Specifies the upper limit for the number of persistent connections to control the number of Web clients that can process requests continuously with one TCP connection.
webserver.connector.inprocess_http.persistent_connection.max_requests	Specifies the upper limit for the request processing frequency of the persistent connection to control the processing when there are continuous requests from the same Web client.
webserver.connector.inprocess_http.persistent_connection.timeout	Specifies the timeout value for persistent connection to control the request waiting time of the persistent connection.

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

(2) Example settings

The following is an example of settings for controlling communication by the persistent connection:

```
...
<param>
<param-name>webserver.connector.inprocess_http.persistent_connection.max_connections</
param-name>
 <param-value>5</param-value>
</param>
<param>
<param-name>webserver.connector.inprocess_http.persistent_connection.max_requests</
param-name>
 <param-value>100</param-value>
</param>
<param>
<param-name>webserver.connector.inprocess_http.persistent_connection.timeout</param-
name>
 <param-value>15</param-value>
</param>
...
```

In this example, when the number of TCP connections exceeds '5' or when the request processing frequency exceeds 100, the TCP connection disconnects after the processing of the request ends. Also, if there is no request to process requests even after the timeout period of 15 seconds passes, the TCP connection disconnects.

5.9 Communication timeout (In-process HTTP server)

This section describes the controlling of communication with the Web client through communication timeout in the in-process HTTP server.

The following table describes the organization of this section.

Table 5-14: Organization of this section (Communication timeout (in-process HTTP server))

Category	Title	Reference
Description	Overview of the Communication Timeout	*5.9.1*
Settings	Execution environment settings (J2EE server settings)	*5.9.2*

Note:

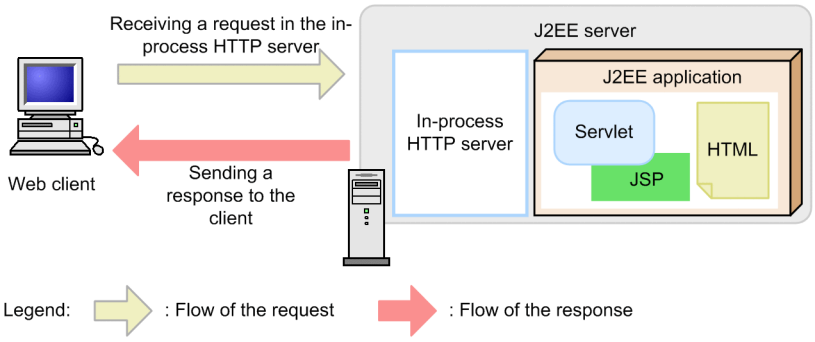
There is no specific description of *Implementation*, *Operations*, and *Notes* for this functionality.

5.9.1 Overview of the communication timeout

When using the in-process HTTP server, you can set communication timeout for receiving a request and sending a response between the Web client and the in-process HTTP server. When the response is awaited due to the network and application failure, you can detect the occurrence of failure from the occurrence of timeout, if the communication timeout is set.

When you use the in-process HTTP server, set the timeout for the communication indicated by the two arrows in the following figure.

Figure 5-7: Communication for which timeout can be set (When using the in-process HTTP server)



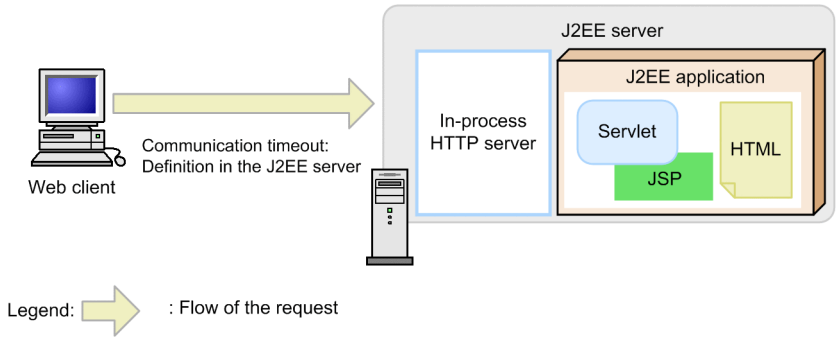
As shown in the figure, communication timeout is set for receiving requests and sending responses. The setting of communication timeout is explained separately for receiving of a request and sending of a response.

Note that when data timeout occurs while receiving a request from the Web client and sending a response to the Web client, a Web client or network failure is assumed and the connection with the Web client is disconnected, so a response is not returned.

(1) Communication timeout when receiving a request

The following figure shows the locations to set the communication timeout for receiving requests.

Figure 5-8: Locations to set the communication timeout for receiving requests (When using the in-process HTTP server)



When using the in-process HTTP server, you set a timeout for the communication between the Web client and the in-process HTTP server.

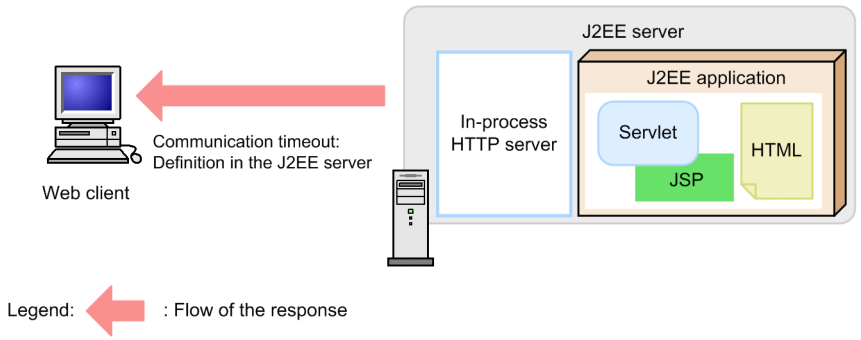
By setting a timeout for the communication between the Web client and the in-process HTTP server, you can detect the following failures in the client:

- The host on which the Web client is running is down.
- A network failure occurred between the Web client and the in-process HTTP server
- A failure occurs in client application.

(2) Communication timeout when sending a response

The following figure shows the locations to set the communication timeout for sending responses.

Figure 5-9: Locations to set the communication timeout for sending responses (When using the in-process HTTP server)



When using the in-process HTTP server, you set a timeout for the communication between the in-process HTTP server and the Web client.

By setting a timeout for the communication between the in-process HTTP server and the Web client, you can detect the following failures:

- The host on which the Web client is running is down.
- A network failure occurred between the Web client and the in-process HTTP server
- A failure occurs in client application.

5.9.2 Execution environment settings (J2EE server settings)

To set up a communication timeout for the in-process HTTP server, you must set up the J2EE server.

This subsection describes the settings and examples of communication timeout in the in-process HTTP server.

You set up communication timeout when receiving requests and when sending responses. The setting of communication timeout is described separately for receiving of a request and sending of a response.

(1) Communication timeout settings for receiving requests

You set the communication timeout for receiving requests between the client and the in-process HTTP server.

Specify the communication timeout for receiving requests in the in-process HTTP server with the following parameters within the <configuration> tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file:

webserver.connector.inprocess_http.receive_timeout

Specifies the waiting time for the process of receiving a request from the client.

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

(2) Communication timeout settings for sending responses

You set the communication timeout for sending responses between the in-process HTTP server and the client.

Specify the communication timeout for sending responses in the in-process HTTP server with the following parameters within the <configuration> tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file:

webserver.connector.inprocess_http.send_timeout

Specifies the waiting time for the process of sending a response to the client.

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

(3) Example settings

An example of settings for communication timeout in the in-process HTTP server is as follows:

```
...
<param>
 <param-name>webserver.connector.inprocess_http.receive_timeout</param-name>
 <param-value>300</param-value>
</param>

<param>
 <param-name>webserver.connector.inprocess_http.send_timeout</param-name>
 <param-value>600</param-value>
</param>
...
```

In this example, 300 seconds is specified as the request receiving timeout and 600 seconds as the response sending timeout.

5.10 Specifying the IP address (In-process HTTP server)

This section describes the controlling of communication with the Web client by specifying the IP address in the in-process HTTP server.

The following table describes the organization of this section.

Table 5-15: Organization of this section (Specifying the IP address (in-process HTTP server))

Category	Title	Reference
Description	Bind address specification functionality	*5.10.1*
Settings	Execution environment settings (J2EE server settings)	*5.10.2*
Notes	Precautions related to IP address specification in the in-process HTTP server	*5.10.3*

Note:
There is no specific description of *Implementation* and *Operations* for this functionality.

Reference note

The functionality is not different when in-process HTTP server is not used (when the Web server integration functionality is used). For details on the functionality, see *4.7 Specifying the IP address (Web server integration)*.

5.10.1 Bind address specification functionality

In the Web container, you can explicitly specify the IP address to be used in the in-process HTTP server. This functionality is called the *Bind address specification functionality*. By using the bind address specification functionality, you can specify the setting so that only a single specific IP address is used for a host having multiple physical network interfaces or single physical network interface, when executing with a host.

5.10.2 Execution environment settings (J2EE server settings)

To specify the IP address of the in-process HTTP server, you must set the J2EE server.

This subsection describes the settings for the IP address of the in-process HTTP server.

Specify the IP address of the in-process HTTP server in the following parameters in the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file:

`webserver.connector.inprocess_http.bind_host`

Specifies the host name or IP address to be used in the in-process HTTP server.

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

5.10.3 Precautions related to IP address specification in the in-process HTTP server

The precautions related to IP address specification in the in-process HTTP server are as follows:

- When the host name or the IP address is set, only requests for connecting to the specified IP address can be received. Instead of setting the IP address, a connection to any IP address on that host can be received, by specifying the wild card address. By default, the setting is specified to use the wild card address. When using the wild card address, note the following points:
  - If the specified host name cannot be resolved in the hosts file or DNS, start the server by using the wild card address.
  - If the specified host name or IP address is a remote host, start the server by using the wild card address.

5.11 Controlling access by limiting the hosts that are allowed access

To prevent unauthorized access of the J2EE server, you can control the hosts that can access the J2EE server. By default, access from all the hosts is allowed. By specifying the host name or the IP address of the host that is allowed access beforehand, you can allow access only from a specific host and prevent unauthorized access.

This section describes the controlling of access by limiting the hosts that are allowed access to the J2EE server.

Table 5-16: Organization of this section (Controlling access by limiting the hosts that are allowed access)

Category	Title	Reference
Description	Limiting the hosts that are allowed access	*5.11.1*
Settings	Execution environment settings (J2EE server settings)	*5.11.2*

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

5.11.1 Limiting the hosts that are allowed access

To limit the hosts, set the host name or IP address of the hosts that are allowed access. In such a case, if you specify an asterisk (*) in place of the host name and the IP address, access from all hosts is allowed. When the hosts that are allowed access are specified by the host name, the name of the host is resolved on starting the J2EE server. Note that even if the local host is not explicitly specified, the access is always allowed. Normally, for the systems that are accessed from external networks, you specify the IP address of the proxy server.

The precautions when the host that is allowed access is specified in the host name are as follows:

Notes

- You need to specify the host name resolvable in the hosts file or DNS. If the host name cannot be resolved, the server is started by the default settings.
- The host name is resolved when the J2EE server is started, and hence, longer time is taken for starting the server. The changed IP address may not be applied after starting the server.

5.11.2 Execution environment settings (J2EE server settings)

To specify settings for limiting the hosts that are given the access, you must set up the J2EE server.

This subsection describes the settings and examples for limiting the hosts that are given the access.

(1) How to set

Specify the settings for limiting the hosts that are given the access in the following parameter within the `<configuration>` tag of the logical J2EE server (j2ee-server), in the Easy Setup definition file.

`webserver.connector.inprocess_http.permitted.hosts`
Specifies the host name or IP address of the host that is given the access.

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

! Important note

The precautions when the host that is given the access is specified in the host name are as follows:

- You need to specify the host name resolvable in the hosts file or DNS. If the host name is not resolvable, the J2EE server is started with the default settings (access is given for all the hosts).

- The host name is resolved when the J2EE server is started, and hence, longer time is taken for starting the J2EE server. The changed IP address may not be applied after starting the server.

(2) Example settings

The following is the setting example for limiting the hosts that are given the access:

```
...
<param>
 <param-name>webserver.connector.inprocess_http.permitted.hosts</param-name>
 <param-value>host1, host2</param-value>
</param>
...
```

In this example, access is given only for 'host1' and 'host2' and access from other hosts is not allowed.

5.12 Controlling access by limiting the request data size

In the in-process HTTP server, by receiving only the request data that is less than a constant size, you can reject the receipt of invalid request data, control the load on the server, and maintain stable operations.

This section describes the controlling of access by limiting the request data size.

The following table describes the organization of this section.

Table 5-17: Organization of this section (Controlling access by limiting the request data size)

Category	Title	Reference
Description	Limiting the request data size	*5.12.1*
Settings	Execution environment settings (J2EE server settings)	*5.12.2*

Note:

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

5.12.1 Limiting the request data size

In the in-process HTTP server, by receiving only the request data that is less than a constant size, you can reject the receipt of invalid request data, control the load on the server, and maintain stable operations.

Set the following items to implement access control by limiting the request data size:

- **Limiting the length of the request line**

  Control the access by setting an upper limit on the length of the request line. The length of a request line includes the HTTP method, URI (including the query string), the HTTP version, and the linefeed (2 bytes) indicating the end of the request line.

  If the length of the received request line exceeds the upper limit, an error of status code 414 is returned to the Web client.

- **Limiting the number of HTTP headers**

  Control the access by setting the upper limit for the number of HTTP headers included in the HTTP request.

  If the number of HTTP headers included in the received HTTP request exceeds the upper limit, an error of status code 400 is returned to Web client.

- **Limiting the request header size**

  Control access by setting the upper limit for the request header size of the HTTP request.

  If the HTTP header size of the received HTTP request exceeds the upper limit, an error of status code 400 is returned to Web client.

- **Limiting the request body size**

  Control access by setting the upper limit for the body size of the HTTP request. In the in-process HTTP server, the body size of the HTTP request is determined by the value of the Content-Length header included in the request header.

  If the body size of the HTTP request exceeds the upper limit, an error of status code 413 is returned to Web client.

  When the request body is sent in a chunk format, the data up to the specified upper limit is read inside the servlet. If the data exceeds the upper limit, an exception (IOException) is thrown in the servlet, but the processing of the servlet continues. Based on the result of data read up to the specified upper limit in the client that sent the request, the response created by the application is returned.

  Tip

  If the gateway device such as SSL accelerator and load balancer exist or if the proxy server is deployed and the gateway equipment and proxy server have the functionality for controlling the request data size, you must set a value less than the value set in the control functionality.

5.12.2 Execution environment settings (J2EE server settings)

To specify the settings for limiting the request data size, you must set up a J2EE server.

This subsection describes the settings and examples for limiting the request data size:

(1) How to set

Implement the J2EE server settings in the Easy Setup definition file. Specify the definition for limiting the request data size in the <configuration> tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file.

The following table lists the definitions in the Easy Setup definition file for limiting the request data size:

Table 5-18: Definitions in the Easy Setup definition file for limiting the request data size

Parameter to be specified	Setting contents
webserver.connector.inprocess_http.limit.max_request_line	Specifies the upper limit of the request line.
webserver.connector.inprocess_http.limit.max_headers	Specifies the upper limit for the number of HTTP headers included in the HTTP request.
webserver.connector.inprocess_http.limit.max_request_header	Specifies the upper limit for the request header size of the HTTP request.
webserver.connector.inprocess_http.limit.max_request_body	Specifies the upper limit for the body size of the HTTP request.

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

Tip

If the gateway device such as SSL accelerator and load balancer exist or if the proxy server is deployed and the gateway device and proxy server have the functionality for controlling the request data size, you must set a value less than the value set in the control functionality.

(2) Example settings

An example of settings for limiting the request data size is as follows:

```
...
<param>
<param-name>webserver.connector.inprocess_http.limit.max_request_line</param-name>
 <param-value>1024</param-value>
</param>
<param>
<param-name>webserver.connector.inprocess_http.limit.max_headers</param-name>
 <param-value>100</param-value>
</param>
<param>
<param-name>webserver.connector.inprocess_http.limit.max_request_header</param-name>
 <param-value>8192</param-value>
</param>
<param>
<param-name>webserver.connector.inprocess_http.limit.max_request_body</param-name>
 <param-value>16384</param-value>
</param>
...
```

5.13 Controlling access by limiting the HTTP-enabled methods

This section describes the controlling of access by limiting the HTTP-enabled methods.

The following table describes the organization of this section:

Table 5-19: Organization of this section (Controlling access by limiting the HTTP-enabled methods)

Category	Title	Reference
Description	Limiting the HTTP-enabled methods	*5.13.1*
Settings	Execution environment settings (J2EE server settings)	*5.13.2*

Note:

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

5.13.1 Limiting the HTTP-enabled methods

In the in-process HTTP server, you reject the receipt of requests containing HTTP-disabled methods by limiting the HTTP-enabled methods for the HTTP request. As a result, you can prevent the unauthorized access of the resources on the server. By default, you can use the `DELETE` method, `HEAD` method, `GET` method, `OPTIONS` method, `POST` method, and `PUT` method.

To limit the HTTP methods, specify the method names of the HTTP-enabled methods. The value defined in RFC2616 must be used for the value set as the HTTP-enabled method. However, an asterisk (*) cannot be used in the method name string. If an asterisk (*) is specified instead of the method name, all the methods can be used.

If a request containing an HTTP-disabled method is received, an error of status code 405 is returned to the Web client.

Note that if a request containing the `OPTIONS` method is sent for the static contents, a method excluding the disabled methods for the in-process HTTP server from the enabled methods (`GET` method, `POST` method, `TRACE` method, and `OPTIONS` method) is returned for the static contents in the `Allow` header included in the response by default. In the case of servlets and JSPs, limiting the HTTP-enabled methods depends on the implementation of the Web application.

5.13.2 Execution environment settings (J2EE server settings)

To specify settings for limiting the HTTP-enabled methods, you must set up the J2EE server.

This subsection describes the settings and examples for limiting the HTTP-enabled methods.

(1) How to set

Specify the settings for limiting the HTTP-enabled methods in the following parameter in the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file.

`webserver.connector.inprocess_http.enabled_methods`

Specifies the method name of an HTTP-enabled method.

For details on the Easy Setup definition file and the parameters to be specified, see 4.*6 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

Tip

If a request containing the `OPTIONS` method is sent for the static contents, a request excluding the disabled methods for the in-process HTTP server from the enabled methods (`GET` method, `POST` method, `TRACE` method, and `OPTIONS` method) is returned for the static contents by default. In the case of servlets and JSPs, limiting the HTTP-enabled methods depends on the implementation of the Web application.

(2) Example settings

The following is the setting example for limiting the HTTP-enabled methods. Note that the following example shows the default settings:

```
...
<param>
 <param-name>webserver.connector.inprocess_http.enabled_methods</param-name>
 <param-value>GET, HEAD, POST, PUT, DELETE, OPTIONS</param-value>
</param>
...
```

In this example, access is allowed for the `GET` method, `HEAD` method, `POST` method, `PUT` method, `DELETE` method, and `OPTIONS` method and access is rejected for the `TRACE` method.

5.14 Customizing responses to the Web client using HTTP responses

This section describes the customizing of responses to the Web client using HTTP responses.

The following table describes the organization of this section:

Table 5-20: Organization of this section (Customizing responses to the Web client using HTTP responses)

Category	Title	Reference
Description	Customizing the HTTP response header	*5.14.1*
Settings	Execution environment settings (J2EE server settings)	*5.14.2*

Note:

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

5.14.1 Customizing the HTTP response header

This subsection describes the customizing of the HTTP response header.

In the in-process HTTP server, you can customize the information that is automatically set up in the Server header of the HTTP response. By default, `CosminexusComponentContainer` is automatically setup.

The value defined in RFC2616 must be used for the value that is automatically set up in the Server header. If the use of Server header is specified in the servlets and JSPs, that setting is given priority.

5.14.2 Execution environment settings (J2EE server settings)

To specify settings for customizing the HTTP response header, you must set up the J2EE server.

This subsection describes the settings and examples for customizing the HTTP response header.

(1) How to set

Specify the settings for customizing the HTTP response header in the following parameter in the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file:

`webserver.connector.inprocess_http.response.header.server`

Specifies the string that is automatically set up in the Server header of the HTTP response.

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

(2) Example settings

The following is the setting example for customizing the HTTP response header:

```
...
<param>
 <param-name>webserver.connector.inprocess_http.response.header.server</param-name>
 <param-value>GyoumuServer/1.0</param-value>
</param>
...
```

In this example, `GyoumuServer/1.0` is specified as the Server header value.

5.15 Error page customization (In-process HTTP server)

If the client access non-existent resources the in-process HTTP server returns the error status code and error page, and the error page generated by the in-process HTTP server is displayed to the client. In the in-process HTTP server, a user-created page can be displayed to the client instead of this error page. This is called *error page customization*.

This section describes the customization of the responses to the Web client with the error page.

The following table describes the organization of this section:

Table 5-21: Organization of this section (Error page customization (in-process HTTP server))

Category	Title	Reference
Description	Error page that can be customized	*5.15.1*
Implementation	Implementation required for customizing the error page	*5.15.2*
Settings	Execution environment settings (J2EE server settings)	*5.15.3*
Notes	Precautions related to error page customization	*5.15.4*

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

5.15.1 Error page that can be customized

When an error such as the access to non-existent resources occurs, a user-created error page can be displayed to the client instead of the error page displaying the error status code.

By using the error page customization with the in-process HTTP server, you can control error page customization corresponding to a specific status code and error page customization corresponding to a request URL in the Web Container at the same time. You can also customize the error page even when you cannot customize the error page with the Web applications in the following cases:

- When the context corresponding to the request does not exist (status code 404)
- When an attempt is made to process the request with a context that is in the process of stopping (status code 503)
- When the in-process HTTP server returns an error status code

For details on the error status codes returned by the in-process HTTP server, see *Appendix A.3 Error status codes returned by the in-process HTTP server*.

In the in-process HTTP server, you can customize the following error pages:

- **Error page customization corresponding to the status codes**

  You can customize the error pages corresponding to the status codes 400 and 500.

  By customizing the error pages corresponding to the status code, you can send the files corresponding to the status code and execute redirection corresponding to the status code.

  - Sending the files corresponding to the status code

    You can return to the client a specific file as a response body for the customized status code. In this case, specify the Content-Type header value of the response.

    Note that if the reading of the file fails during request processing, the default error page is used.

  - Redirection corresponding to the status code

    You can redirect to a specific URL for the customized status code. In the case of redirection, specify 302 as the response status code and the redirect URL in the Location header.

    When sending a file corresponding to the status code, redirection cannot be executed.

- **Error page customization corresponding to the request URLs**

  You can specify a request URL and customize the error page for a specific URL. When the request URL is specified, the customized error page is returned to the client only when an error occurs in the request processing matching with the specified URL.

5.15.2 Implementation required for customizing the error page

To customize the error page with the in-process HTTP server, use the `sendError` method of the `javax.servlet.http.HttpServletResponse` interface, and set the response status code. Note that if you use the `setStatus` method (such as when the `setStatus` method is used in JSP), customization might not be executed by the in-process HTTP server. However, even if you use the `sendError` method, if the Web application fulfills one of the following conditions, the error page is not customized by the in-process HTTP server:

- If an exception is thrown during the execution of the `sendError` method
- When error page customization is specified in the Web application and when an error occurs, the execution of the error page as per the settings terminates normally#

#

Normal termination of error page execution implies the satisfaction of the following conditions:

- An exception that cannot be caught in the error page does not occur.
- The status code ends with a value other than 400 to 599.

5.15.3 Execution environment settings (J2EE server settings)

To customize the error pages, you must set up the J2EE server.

This subsection describes the settings and examples for error page customization.

(1) How to set

Implement the J2EE server settings in the Easy Setup definition file. Specify the definition for error page customization in the <`configuration`> tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file.

The following table lists the definitions in the Easy Setup definition file for the error page customization:

Table 5-22: Definitions in the Easy Setup definition file for error page customization

Parameter to be specified	Setting contents
`webserver.connector.inprocess_http.error_custom.list`	Specifies the definition name for error page customization.
`webserver.connector.inprocess_http.error_custom.`*error-page-customising-definition-name*`.status`	Specifies the error status code that customizes the error page to customize the error page in association with the error status code.
`webserver.connector.inprocess_http.error_custom.`*error-page-customising-definition-name*`.file`	Specifies the file to be returned to the client as a response body to send a file corresponding to the error status code.
`webserver.connector.inprocess_http.error_custom.`*error-page-customising-definition-name*`.file.content_type`	Specifies the value of the Content-Type header when a file specified in the `webserver.connector.inprocess_http.error_custom.`*error-page-customizing-definition-name*`.file` parameter is sent to the client as a response body.
`webserver.connector.inprocess_http.error_custom.`*error-page-customising-definition-name*`.redirect_url`	Specifies the redirection destination URL when redirecting in compliance with the error status code.
`webserver.connector.inprocess_http.error_custom.`*error-page-customising-definition-name*`.request_url`	Specifies the request URL that applies error page customization when customizing the error page in association with the request URL.

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

(2) Example of settings

The following is the setting example of settings for the error page customization:

```
...
<param>

<param-name>webserver.connector.inprocess_http.error_custom.list</param-name>
 <param-value>ERR_CUSTOM_1, ERR_CUSTOM_2</param-value>
</param>
<param>

<param-name>webserver.connector.inprocess_http.error_custom.ERR_CUSTOM_1.status</
param-name>
 <param-value>404</param-value>
</param>
<param>

<param-name>webserver.connector.inprocess_http.error_custom.ERR_CUSTOM_1.file</param-
name>
 <param-value>C: /data/404.html</param-value>
</param>
<param>

<param-
name>webserver.connector.inprocess_http.error_custom.ERR_CUSTOM_1.file.content_type</
param-name>
 <param-value>text/html; charset=ISO-8859-1</param-value>
</param>
<param>

<param-name>webserver.connector.inprocess_http.error_custom.ERR_CUSTOM_2.status</
param-name>
 <param-value>503</param-value>
</param>
<param>

<param-
name>webserver.connector.inprocess_http.error_custom.ERR_CUSTOM_2.redirect_url</param-
name>
 <param-value>http: //host1/503.html</param-value>
</param>
<param>

<param-name>webserver.connector.inprocess_http.error_custom.ERR_CUSTOM_2.request_url</
param-name>
 <param-value>/dir1/*</param-value>
</param>
...
```

In this example, `ERR_CUSTOM_1` and `ERR_CUSTOM_2` are used as the error page customizing definition names. In `ERR_CUSTOM_1`, when the response status code is '404', `C:/data/404.html` is returned to the client. `text/html; charset=ISO-8859-1` is used as the Content-Type header value. In `ERR_CUSTOM_2`, when the request is a URL beginning with `/dir1/` and the response status code is '503', the request is redirected to `http://host1/503.html`.

5.15.4 Precautions related to error page customization

The precautions related to error page customization by the in-process HTTP server are as follows:

- When the HTTP response is an HTML file, if another file (such as an image file) is being referenced from that HTML, the error page might not be properly displayed.
- Note that depending on the browser settings, if the status code indicates an error and if the size of the response body is small, the response body MIGHT be replaced by a browser-specific message. Note that in the default error page displayed when an error page is not specifically customized, the size of the response body is small.
- If the value specified in the redirect URL matches with the value specified for the request URL and if the same error status occurs after redirection, note that the client keeps redirecting the requests.
- When redirect corresponding to the status code is implemented, the query string added to the request URL when an error occurs, is not added to the redirect URL. Also, when a session is managed by URL rewriting, the session cannot be inherited even if the request is redirected to the same Web application as the error page.

5.16 Notification of gateway information to a Web container

This section describes the reporting of the gateway information to the Web container.

The following table describes the organization of this section:

Table 5-23: Organization of this section (Reporting of the gateway information to the Web container)

Category	Title	Reference
Description	Gateway specification functionality	*5.16.1*
Settings	Execution environment settings (J2EE server settings)	*5.16.2*
Notes	Precautions related to reporting the gateway information to the Web Container	*5.16.3*

Note:
There is no specific description of *Implementation* and *Operations* for this functionality.

Reference note

The functionality is not different when in-process HTTP server is not used (when the Web server integration functionality is used). For details on the functionality, see *4.10 Notification of gateway information to a Web container*.

5.16.1 Gateway specification functionality

If a gateway such as an SSL accelerator or a load balancer is placed between a client and an in-process HTTP server, when the Web container automatically redirects to a welcome file or the Form authentication window, the Web container may not properly create a forwarding URL because the container cannot acquire the information about the gateway.

To avoid this problem, you can use the *gateway specification functionality*. This functionality notifies a Web container of gateway information so that the Web container can properly redirect to a welcome file or Form authentication window.

The gateway specification functionality is used in the following case:

- **When an SSL accelerator is placed between a client and in-process HTTP server:**
  Even if a client accesses an SSL accelerator via HTTPS, the SSL accelerator accesses a Web server via HTTP, which causes the Web container to assume that the access uses HTTP. For this reason, HTTP is used for the URL scheme for the welcome file or Form authentication window that is the redirection destination.
  In this situation, by using the gateway specification function to specify that the scheme be always considered as HTTPS, you can ensure that accesses are properly redirected.
- **When a request without a Host header needs to be redirected away from the in-process HTTP server that received the request**
  When redirecting a request without a Host header, the host name and port number of the redirection destination URL becomes the host name and port number of the Web server that receives the request.
  Use the gateway specification functionality when the host name and port number of the URL accessed by the client is different from the Web server or in-process HTTP server that receives the request, such as when a load balancer is deployed before the Web server or in-process HTTP server. As a result, the host name and port number accessed from the client are specified, so the request can be redirected properly.

Note that when using the in-process HTTP server, gateway specification functionality cannot be used if multiple different routes are used for accessing one Web container (when HTTP requests are forwarded to the Web container from multiple gateways). To use the gateway specification functionality in the in-process HTTP server, use a configuration in which there is one access route to the Web container.

5.16.2 Execution environment settings (J2EE server settings)

To specify settings for reporting the gateway information to the Web Container, you must set up a J2EE server.

This subsection describes the settings and examples for reporting the gateway information to the Web container.

(1) How to set

Implement the J2EE server settings in the Easy Setup definition file. Define the settings for reporting the gateway information to the Web container in the <configuration> tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file.

The following table lists the definitions in the Easy Setup definition file for reporting the gateway information to the Web container:

Table 5-24: Definitions in the Easy Setup definition file for reporting the gateway information to the Web container

Parameter to be specified	Setting contents
webserver.connector.inprocess_http.gateway.https_scheme	Specifies the scheme of the redirection destination URL.
webserver.connector.inprocess_http.gateway.host	Specifies the gateway host name.
webserver.connector.inprocess_http.gateway.port	Specifies the gateway port number.

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

(2) Example settings

An example of settings for the gateway specification functionality is as follows:

```
...
<param>
 <param-name>webserver.connector.inprocess_http.gateway.host</param-name>
 <param-value>host1</param-value>
</param>
<param>
 <param-name>webserver.connector.inprocess_http.gateway.port</param-name>
 <param-value>4443</param-value>
</param>
<param>

<param-name>webserver.connector.inprocess_http.gateway.https_scheme</param-name>
 <param-value>true</param-value>
</param>
...
```

In this example, the gateway specification functionality is used to specify settings so that the scheme is always considered to be HTTPS.

5.16.3 Precautions related to reporting the gateway information to the Web container

The following are cautionary notes on using the gateway specification functionality:

- Specifying the host name and port number of an URL where an access is redirected:

  A browser usually sends a request with the Host header appended, so it is not necessary to specify the host name or port number for an URL where access is to be redirected.

  Note that you can check whether or not the request has the Host header by calling the getHeader method of the javax.servlet.http.HttpServletRequest interface, with the Host argument specified.

- Servlet API behavior:

  Using the gateway specification functionality causes some servlet API functions to behave differently. Take care when using API functions with a Web application.

  For details on the servlet API functions where the operations are changed, see *6.2.2(10) Precautions for using the gateway specification functionality*.

- The `<transport-guarantee>` tag in `web.xml`:

  When you use the gateway specification functionality to specify that a scheme is to be considered as HTTPS, a request to a Web server will be considered to use HTTPS even if the request actually uses HTTP. Note that this prevents an access from being redirected to an URL that uses HTTPS, even if you specify INTEGRAL or CONFIDENTIAL in the `<transport-guarantee>` tag in `web.xml`.

- The Secure attribute for cookies:

  When you use the gateway specification functionality to specify that a scheme is to be considered as HTTPS, when a session ID generated by a Web container is returned to the client by the session cookie, the Secure attribute is appended to the cookie.

5.17 Output of log and trace

This section describes the log and trace output by the in-process HTTP server.

The following table describes the organization of this section:

Table 5-25: Organization of this section (Output of log and trace)

Category	Title	Reference
Description	Log and trace output by the in-process HTTP server	*5.17.1*
Settings	Customizing the access log of the in-process HTTP server	*5.17.2*

Note:

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

5.17.1 Log and trace output by the in-process HTTP server

The in-process HTTP server outputs the log and trace described in the following table to support application development, for performance analysis during operations, and for troubleshooting during a failure:

Table 5-26: Log and trace output by the in-process HTTP server

Types of log and trace	Description
Access log	Outputs the result of request and response processing from the Web client. This log is used for analyzing the communication with the Web client. By analyzing the access log, you can analyze the files requested from the Web client and the performance information and session tracking information of the in-process HTTP server.
Performance analysis trace	Outputs the performance analysis information of the sending and receiving of requests and the information for troubleshooting in the case of failure in the in-process HTTP server. The performance analysis trace is converted to a CSV format and can be used to analyze the bottlenecks in the entire system in combination with the performance analysis information output by the functionality of other J2EE servers. For details on the performance analysis trace, see *4.6 Performance analysis trace* in the *uCosminexus Application Server Maintenance and Migration Guide*.
Thread trace	Trace for maintenance.
Communication trace	Trace for maintenance.

5.17.2 Customizing the access log of the in-process HTTP server

The in-process HTTP server outputs the access log, performance analysis trace, thread trace, and communication trace for supporting application development, for performance analysis during operations, and for troubleshooting during failure. You can change the number and size of these files. You can also customize the log output format in the access log.

This subsection describes the customization of the access log output format in the in-process HTTP server. For details on changing the number and size of the access log and trace files in the in-process HTTP server, see *3.3.11 Settings for acquiring logs of the in-process HTTP server* in the *uCosminexus Application Server Maintenance and Migration Guide*.

(1) Customization procedure

To customize the access log output format:

1. Define the format name of the access log.

To create a new format, add the new format name in the `webserver.logger.access_log.format_list` parameter in the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file.

Example settings

```
...
<param>
 <param-name>webserver.logger.access_log.format_list</param-name>
 <param-value>formatA</param-value>
</param>
...
```

For creating a new format, reference the access log format provided by default. For details on the format, see *(2) Access log format*.

2. Define the output format for the access log.

   Define the access log output format in the format specified in *format-name* with the `webserver.logger.access_log.`*format-name* parameter within the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file. Specify the argument for the access log format in the output format definition.

   Example settings

```
...
<param>
 <param-name>webserver.logger.access_log.formatA</param-name>
 <param-value>%h %u %t &amp;quot;%r&amp;quot; %&amp;gt;s HostHeader=&amp;quot;%{host}i&amp;quot;</
param-value>
</param>
...
```

   For details on the specifiable format arguments, see *(3) Arguments of the access log format*.

3. Specify the format that will be used to output the access log.

   Specify the format name that will be used to output the access log with the `webserver.logger.access_log.inprocess_http.usage_format` parameter within the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file. An access log is output in the format specified here:

   Example settings

```
...
<param>
 <param-name>webserver.logger.access_log.inprocess_http.usage_format</param-
name>
 <param-value>formatA</param-value>
</param>
...
```

(2) Access log format

Application Server provides two types of formats namely *common* (default format) and *combined* (extended format) as the access log formats of the in-process HTTP server. When creating a new format, reference these formats.

(a) Output format

The output format of access log is described below. Note that Δ indicates one-byte space. Also, for the convenience of expression, the log is output across multiple lines, but the log is actually output in one line.

The output format of a default format is as follows:

```
Host-name-or-IP-address-of-Web-client Δ Remote-log-name Δ Authentication-user-name
Δ Start-time-of- Web-client-request-processing Δ Request-line
Δ Final-status-code Δ Number-of-bytes-sent-excluding-the-HTTP-header
```

The output format of an extended format is described below:

```
Host-name-or-IP-address-of-Web-client Δ Remote-log-name Δ Authentication-user-name
Δ Start-time-of- Web-client-request-processing Δ Request-line
```

```
△Final-status-code △ Number-of-bytes-sent-excluding-the-HTTP-header
△ "Referer-header-contents" △ "User-Agent-header-contents"
```

The underlined part is the difference between the default format and extended format. In the extended format, `Referer header contents` and `User-Agent header contents` is output in addition to the output contents of the default format.

(b) Example of output

An example of access log output in the default format is as follows:

```
10.20.30.40 - user [20/Dec/2004: 15: 45: 01 +0900] "GET /index.html HTTP/1.1" 200 8358
10.20.30.40 - user [20/Dec/2004: 15: 45: 01 +0900] "GET /left.html HTTP/1.1" 200 2358
10.20.30.40 - user [20/Dec/2004: 15: 45: 01 +0900] "GET /right.html HTTP/1.1" 200 4358
```

An example of access log output in the extended format is as follows:

```
10.20.30.40 - - [18/Jan/2005: 13: 06: 10 +0900] "GET / HTTP/1.0" 200 38 "-" "Mozilla/
4.0 (compatible; MSIE 6.0; Windows NT 5.1) "
10.20.30.40 - - [18/Jan/2005: 13: 06: 25 +0900] "GET /demo/ HTTP/1.0" 500 684 "-"
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1) "
```

(3) Arguments of the access log format

The following table lists the arguments of the access log format that are specified when you define the output format of the format:

Table 5-27: List of arguments of the access log format

Format arguments	Output contents	Example of output
%%	% Sign	%
%a	IP address of the Web client	10.20.30.40
%A	IP address of the J2EE server	10.20.30.100
%b	Number of bytes sent excluding the HTTP header ('-' in the case of 0 bytes)	2048
%B	Number of bytes sent excluding the HTTP header ('0' in the case of 0 bytes)	1024
%h	Host name or IP address of the Web client (IP address when the host name cannot be acquired)	10.20.30.40
%H	Request protocol	HTTP/1.1
%l	Remote log name[#1] (Always '-')	-
%m	Request method	GET
%p	Port number that receives the request from the Web client	80
%q	Query string (Begins with '? '. If the query string does not exist, null character)	?id=100&page=15
%r	Request line	GET /index.html HTTP/1.1
%>s	Final status code (Internally redirected value is not output)	200

Format arguments	Output contents	Example of output
%S[#2]	User's session ID ('-' if the session ID does not exist)	00455AFE4DA4E7B7789F247B8FE5D605
%t	Start time of the Web client request processing (Unit: seconds, output format: dd/MMM/YYYY:HH:mm:ss Z)	[18/Jan/2005: 13: 06: 10 +0900]
%T	Time required for processing the Web client request (Unit: seconds)	2
%d	Start time of the Web client request processing (Unit: milliseconds, output format: dd/MMM/YYYY:HH:mm:ss.nnn Z (nnn indicates milliseconds))	[18/Jan/2005: 13: 06: 10.152 +0900]
%D	Time required for processing the Web client request (Unit: milliseconds)	2000
%u	Basic authentication user name, Form authentication user name ('-' when the authentication user name does not exist)	user
%U	Request file path	/index.html
%v	Local host name of the J2EE server	server
%{foo}I[#3]	Contents of request header `foo` ('-' when foo header does not exist)	In the case of %{Host}i, www.example.com:8888
%{foo}c	Of the Cookie information sent by the Web client, contents of Cookie name `foo` ('-' when the Cookie name does not include `foo`)	In the case of %{JSESSIONID}c, 00455AFE4DA4E7B7789F247B8FE5D605
%{foo}o[#3]	Contents of response header `foo` ('-' when the foo header does not exist)	In the case of %{Server}o, CosminexusComponentContainer

#1

The remote log name is the user name in the Web client that can be acquired with the Identification protocol defined in RFC 1413.

#2

The session ID displayed in `%S` is the value of the Cookie name 'JSESSIONID'. This session ID is different from the global session ID in the memory session failover functionality. When you want to output the global session ID, specify `%{GSESSIONID}c`. If `GIDCookieName` is changed, specify the value of the changed `GIDCookieName`.

#3

The same header name might be sent multiple times in one HTTP request or HTTP response. In this case, the contents of the header read first will be output.

! Important note

If there is an error in the specification of the format argument, the default format is used. The example of the use of the default format is as follows:

- When strings not existing in the list of format arguments (example: `%G`) is specified
- When `0` characters (such as `%{}i`) are specified in the request header contents, response header contents, and Cookie name.

Reference note

If the default format and extended format are coded in the format arguments, the format is as follows:

- Default format
  %h %l %u %t "%r" %>s %b
- Extended format

```
%h %l %u %t "%r" %>s %b "%{Referer}i" "%{User-Agent}i"
```

---

6 Implementation of Servlets and JSPs

This chapter describes the precautions for implementing servlets and JSPs.

6.1 Support range of the functionalities that are added or changed in Servlet specifications and JSP specifications

This section gives an overview of the functionalities that are added or changed in the Servlet and JSP specifications, and also describes the support range of functionalities in Application Server.

For details on operations and precautions when using the functionalities that are added or changed in different versions of Servlet specifications and JSP specifications in Application Server, see the sections starting from *6.2.3*.

The following table gives an overview of the functionalities that are added, or changed in Servlet 3.0 and describes the support range.

Table 6-1: Overview of the Servlet 3.0 functionalities and the support range

No.	Functionality name	Functionality overview	Support
1	`web.xml` (Servlet 3.0) and new annotations	You can use a `web.xml` that is compatible with Servlet 3.0.	Y
		You can define servlet annotations. (`web.xml` can be omitted)	Y
		You can use `web-fragment.xml`.	N #1
2	Dynamic servlet definition	You can define servlet, filters or listeners with an API.	Y
3	File upload	Requests with multipart/form-data as the `content-type` can be processed.	Y
4	Static resource allocation	You can allocate static resources or JSP to `META-INF/resources` of the JAR file.	Y
5	Security enhancements	• You can set security with annotations. • You can use APIs for authentication.	Y
6	Asynchronous servlet	Request processing and response generation can be done in a different thread other than the thread that receives requests.	N #2
7	Other changes in the Servlet specifications	You can add the `HttpOnly` attribute to Cookie.	Y
		You can change the HTTP Cookie name indicating a session ID of the HTTP session.	Y
		You can use the HTTP digest authentication.	N #3
		You can acquire the SSL Session ID as the `ServletRequest` property.	N #4
8	Changes in JSP specifications	You can use a `web.xml` to specify the default content type or the buffer size.	N #5
9	Methods with the `EL` parameter	You can invoke methods having parameters.	Y #6
10	API enhancements	You can use newly added or changed APIs.	L #7

Legend:

Y: Supported

N: Not supported

L: Limited support

#1

Is ignored, if the Web application includes `web-fragment.xml`.

#2

The DD and annotations related to asynchronous servlets are ignored. An exception is thrown if you invoke an API of an asynchronous servlet.

#3

Operation is not guaranteed if you set the digest authentication in the DD.

#4

If you specify `javax.servlet.request.ssl_session_id` as an argument in the `ServletRequest` class and invoke the `getAttribute` method, null is always returned.

#5

A `web.xml` compatible with JSP 2.2 can be read. However, any tag added in JSP 2.2 is ignored.

Even for a Web application of version 3.0, the JSP version that complies with JSP is version 2.1.

#6

You cannot use the Expression Language of JSP 2.2 if the EL is of the JSP 2.1 or 2.0 specifications. If you use an API of JSP 2.2 EL, the operation is not guaranteed because the API is not checked.

#7

You can use APIs of supported functionalities, but not of functionalities that are not supported. For details on APIs of Servlet 3.0, see *6.2.3(9) About APIs*.

6.2 Precautions for implementing servlets and JSPs

This section describes the precautions for implementing servlets and JSPs.

The following table describes the organization of this section.

Table 6-2: Organization of this section (Precautions for implementing servlets and JSPs)

Titles	Reference
Common precautions for implementing servlets and JSPs	*6.2.1*
Precautions for implementing servlets	*6.2.2*
Precautions related to the specifications that are added or changed in the Servlet 3.0 specifications	*6.2.3*
Precautions related to added and changed specifications in the Servlet 2.5 specifications	*6.2.4*
Precautions related to added and changed specifications in the Servlet 2.4 specifications	*6.2.5*
Precautions for implementing JSPs	*6.2.6*
Precautions related to added and changed specifications in the JSP 2.1 specifications	*6.2.7*
Precautions related to added and changed specifications in the JSP 2.0 specifications	*6.2.8*
Precautions for implementing JSPs of the JSP 1.2 specifications	*6.2.9*
Precautions related to the specifications that are added or changed in the EL2.2 specifications	*6.2.10*
Points to remember when upgrading the version of an existing Web application to the Servlet 3.0 specifications	*6.2.11*
Points to remember when upgrading the version of an existing Web application to the Servlet 2.5 specifications	*6.2.12*
Precautions related to Web applications when migrating from a previous version of Application Server to 09-00	*6.2.13*
Points to remember when upgrading the version of an existing Web application to the Servlet 2.4 specifications	*6.2.14*
Using annotations in servlets	*6.2.15*
Precautions related to size limitations for JavaVM methods	*6.2.16*

6.2.1 Common precautions for implementing servlets and JSPs

This subsection describes the common precautions for implementing servlets and JSPs as the application programs running on the Application Server.

(1) J2EE application version requirements for Web application operations

For each Web application version, the following table lists the J2EE version to which the J2EE application conforms and that forms the prerequisite for Web application operations:

Table 6-3: J2EE version to which the J2EE application conforms

Version of J2EE specifications to which the J2EE application conforms	Servlet specifications corresponding to the Web applications				
	3.0	2.5	2.4	2.3	2.2
Java EE 6	Y	Y	Y	Y	Y
Java EE 5	N	Y	Y	Y	Y
J2EE1.4	N	N	Y	Y	Y

Version of J2EE specifications to which the J2EE application conforms	Servlet specifications corresponding to the Web applications				
	3.0	2.5	2.4	2.3	2.2
J2EE1.3	N	N	L	Y	Y
J2EE1.2	N	N	L	L	Y

Legend:

Y: Available

L: Limited (since the version of the J2EE specifications is updated to 1.4 when importing the J2EE application)

N: Not available

(2) Supported range of Web applications

The Web application version is identified by the version information of the Servlet specifications described in `web.xml`. A Web application of a higher version can use the functionality of a lower version. A Web application of a lower version cannot use the functionality of a higher version.

The following table describes the range of functionality available for each Web application version:

Table 6-4: Supported range of Web applications

Version of the Web application	Servlet						JSP				Tag library#1			
	3.0	2.5	2.4	2.3	2.2	2.2	2.1	2.0	1.2	1.1	2.1	2.0	1.2	1.1
3.0	Y	Y	Y	Y	Y	N#2	Y	Y	Y	Y	Y	Y	Y	Y
2.5	N	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y
2.4	N	N	Y	Y	Y	N	N	Y	Y	Y	N	Y	Y	Y
2.3	N	N	N	Y	Y	N	N	N	Y	Y	N	N	Y	Y
2.2	N	N	N	N	Y	N	N	N	N	Y	N	N	N	Y

Legend:

Y: Available

N: Not available

#1

The tag library version indicates the version of the tag library descriptor (TLD file).

#2

A `web.xml` compatible with JSP 2.2 can be read. However, any tag added to JSP 2.2 is ignored.

Note that if the functionality of a later version is used from a Web application of an earlier version, an error might occur. The following table describes the errors in each version:

Table 6-5: Errors that occur if functionalities of Servlet 3.0 are used from a Web application that is compatible with Servlet 2.2, 2.3, 2.4, or 2.5 specifications

Specifications	Functionality used	Operation in the event of an error
Servlet 3.0	Invoking new API	The program does not check whether an API of the Servlet 3.0 specification is being used. To avoid any abnormal operation, do not call the API.
	Using new annotations	For handling errors occurring due to annotations, see *12. Using Annotations* in the *uCosminexus Application Server Common Container Functionality Guide.*
JSP 2.2	New EL API	The program does not check whether an API of EL2.2 specification is being used. To avoid any abnormal operation, do not call the API.

For details on operations and precautions when upgrading a Web application from Servlet 2.2, Servlet 2.3, Servlet 2.4, or Servlet 2.5 specifications to Servlet 3.0 specifications, see *6.2.11 Points to remember when upgrading the version of an existing Web application to the Servlet 3.0 specifications*.

Table 6-6: Errors when Servlet 2.5 functionality is used from a Web application corresponding to Servlet 2.2, 2.3, or 2.4 specifications

Specifications	Functionality used	Error processing
Servlet 2.5	Invoking new API	The check whether the API added in the Servlet 2.5 specifications is used, is not performed. The operations when the API is invoked do not function properly, so do not invoke the API.
	Using new annotations	For the processing when an annotation is used and an error occurs, see *12. Using annotations* in the *uCosminexus Application Server Common Container Functionality Guide*.
JSP 2.1	Attributes of the new directive#1	KDJE39145-E message is output in the servlets log and KDJE39186-E message is output in the message log#2 and a translation error occurs.
	TLD 2.1	If the following TLD file exists when the Web application starts, the KDJE39293-W message is output in the message log and the processing is not performed: • TLD file specified in the `<tablib-location>` element in the `<taglib>` element of `web.xml` • TLD file deployed under the `/META-INF` directory in the Jar file under the `/WEB-INF/lib` directory If a TLD file other than the above exists when the Web application starts, the KDJE39293-W message is output in the message log during JSP compilation and the processing is not performed. If JSP compilation occurs when the application is accessed the first time, the KDJE39145-E message is output in the servlets log and the KDJE39186-E message is output in the message log#2 and a translation error occurs.
	EL addition functionality	• Dedicated processing is not implemented for the `Enum` type added in JSP 2.1. The processing is same as for the general classes. However, if the EL API of the JSP 2.0 specifications that is deprecated in the JSP 2.1 specifications is used, regardless of the Web application version, the API is processed in the EL functionality range of the JSP 2.0 specifications. • EL with `#{}` format is displayed as a string. • `"\#"` is not handled as an escape sequence, and is displayed as a string `"\#"`.

#1

If an undefined directive is specified in the JSP specifications for XXX using the format `<jsp:directive.XXX/>` on the JSP page or if an undefined standard action is specified in the JSP specifications for XXX using the format `<jsp:XXX>`, the definition contents are output as is.

#2

The details of JSP compilation error are output in KDJE39145-E and the reporting of translation error is output in KDJE39186-E.

For details on the operations and precautions when upgrading the Web application version from Servlet 2.2 specifications, Servlet 2.3 specifications, and Servlet 2.4 specifications to Servlet 2.5 specifications, see *6.2.12 Points to remember when upgrading the version of an existing Web application to the Servlet 2.5 specifications*.

Table 6-7: Errors when using the functionality of Servlet 2.4 specifications from a Web application corresponding to Servlet 2.2 or 2.3 specifications

Specifications	Functionality used	Error processing
Servlet 2.4	Invoking new API	The check for whether the API added in the Servlet 2.4 specifications is used, is not performed. The operations when the API is invoked do not function properly, so do not invoke the API.

Specifications	Functionality used	Error processing
Servlet 2.4	Registering new listener	The KDJE39297-W message is output in the message log when the Web application starts and the listener definition is ignored.
JSP 2.0	New directive and new standard action#1	The KDJE39145-E message is output in the servlets log and the KDJE39186-E message is output in the message log#2, and a translation error occurs.
	Tag files	When TLD is not used The `tagdir` attribute that is a new attribute is assumed to be invalid in the `taglib` directive, the KDJE39145-E message is output in the servlets log and the KDJE39186-E message is output in the message log#2, and a translation error occurs. When TLD is used A TLD 2.0 usage error occurs.
	TLD 2.0	The following TLD files are checked when the Web application starts. When applicable, the KDJE39293-W message is output in the message log and the file is ignored: • TLD file specified in `<taglib><tablib-location>` of `web.xml` • TLD file deployed under `/META-INF` in the Jar file under `/WEB-INF/lib` The TLD files other than above-mentioned are checked during the JSP compilation. When the JSP file is compiled, such as during the initial access, the KDJE 39145-E message is output in the servlets log and the KDJE39186-E message is output in the message log#2, and a translation error occurs.
	Simple Tag Handler	The KDJE39145-E message is output in the servlets log and the KDJE39186-E message is output in the message log#2, and a translation error occurs.

#1

If an undefined directive is specified in the JSP specifications for XXX using the format `<jsp:directive.XXX/>` on the JSP page or if an undefined standard action is specified in the JSP specifications for XXX using the format `<jsp:XXX>`, the definition contents are output as it is.

#2

The details of JSP compilation error are output in KDJE39145-E and the reporting of translation error is output in KDJE39186-E.

For details on the operations and precautions when upgrading the Web application version from Servlet 2.2 specifications and Servlet 2.3 specifications to Servlet 2.4 specifications, see *6.2.14 Points to remember when upgrading the version of an existing Web application to the Servlet 2.4 specifications*.

Note that even if you use the Servlet 2.3 functionality from the Web applications corresponding to the Servlet 2.2 specifications, when imported, the application is rewritten by the Web application conforming to the Servlet 2.3 specifications, and therefore, the application is processed normally and error is not reported.

(3) Notes on using the transaction and JDBC connection

To use a transaction in the servlets and JSPs, acquire the JDBC connection with the applicable service method and release the connection before the applicable service method ends. In the servlets and JSPs where the transaction is running, the use of the following JDBC connections is not supported:

- Use of the JDBC connection on the thread generated by the service methods of the servlets and JSPs.
- Use of the JDBC connection in the service methods of the other servlets and JSPs invoked from the service methods of the servlets and JSPs.
- Use of the JDBC connection acquired with the `init` method of the service methods of the servlets and JSPs.
- Use of the JDBC connection stored in the instance variable.#

#

When the servlets and JSPs of `SingleThreadModel` are used, the JDBC connection can be stored in the instance variable.

(4) Notes related to package name specification

If a class with an invalid package name is used in the servlets and JSPs, an error with status code 500 occurs when the class is accessed from the browser. For example, even if a created class file is deployed correctly and accessed from the browser, if the package name declaration is invalid, the applicable class is not found. In this case, an error with status code 500 is returned.

(5) Notes for using cookies

- Do not use cookies containing double-byte codes such as Japanese characters. If such cookies are used, the HTTP session used in the servlets and JSPs might be lost.
- When managing a session with a cookie, the session generated using the servlets or JSPs accessed in the URL by the host name is not inherited in the servlets or JSPs accessed in the URL with the IP address specified instead of the host name (and vice versa).

(6) Notes related to display of input values with special meanings

When the input values of characters with special meanings such as "<" and ">" in forms are displayed as it is, malicious users might use the tags such as `<SCRIPT>`, `<OBJECT>`, `<APPLET>`, `<EMBED>` that can execute scripts and cause important security-related problems. A processing must be added for the application developer to inspect the data entered by a user and the characters with special meanings must be excluded.

(7) Notes related to error page display after response commit

After the response is committed in the servlets or JSPs, even if an error such as an exception occurs, the following error pages are not displayed in the browser:

- Error pages specified in `web.xml`
- Error pages specified in the `errorPage` attribute of the JSP page directive
- Default error pages output by the Web container server

The Web container commits the response automatically when the response buffer becomes full apart from the case in which the user commits the response by invoking the `flushBuffer` method of the `ServletResponse` class explicitly.

To check whether the response is committed in the servlets or JSPs, use the `isCommitted` method of the `ServletResponse` class. Also, you can change the buffer size using the `setBufferSize` method of the `ServletResponse` class in the case of servlets and by specifying the buffer attribute of the page directive for JSPs.

(8) Improving the performance when using the PrintWriter and JSPWriter class

By reducing the frequency of invoking the `print` method and the `println` method of the `PrintWriter` class and the `JSPWriter` class, you can reduce the access frequency and improve the performance. For example, use the `StringBuffer` class and invoke the `println` method finally to reduce the frequency of invoking the `print` and `println` methods.

(9) Notes on referencing the error information by javax.servlet.error.XXXXX

The `javax.servlet.error.XXXXX` attributes defined in the Servlet 2.3 specifications are used for referencing the error information that causes the execution of the error page in the servlets or JSPs specified in the `<error-page>` tag of `web.xml`. Do not reference these attributes from servlets or JSPs other than those specified in the `<error-page>` tag of `web.xml`.

(10) Notes on accessing files

To access a file, make sure you specify the absolute path. If you specify the relative path, the J2EE server tries to search the target path according to the relative path from the execution directory of the Web container server. If you specify the relative path in the `getRealPath` method of the `ServletContext` class, the relative path in the directory that deploys the WAR file is acquired.

Furthermore, when you access a file, make sure that you close the file. If you access a file in the WAR file deployment directory and do not close the file, you cannot perform the normal un-deployment on the J2EE server. If the file not closed even when the path under the WAR file deployment directory is not specified, events such as the files cannot be deleted when the J2EE server is running occur.

(11) Error page settings when an exception occurs

If an exception occurs while accessing the JSPs and servlets, the exception status code is returned to the browser by the default processing in the Web Container. To change this default processing, specify the JSP errorPage or set the error page in `web.xml`.

(12) Notes related to acquisition of the class loader

To use the following methods by acquiring the class loader of the Cosminexus Component Container from the code in the J2EE application, use the `java.net.JarURLConnection` class:

- `getResource(String) .openConnection() .getInputStream()`
- `getResource(String) .openStream()`

In the process in which these methods are invoked, the `openConnection` method of the `java.net.JarURLConnection` class is invoked and the JAR file specified in the applicable URL is opened. This JAR file remains open unless the `close` method is explicitly invoked and cannot be deleted. Do not use these methods in the J2EE application. Also, when the operations for the JAR file are required and when using the `openConnection` method of the `java.net.JarURLConnection` class, make sure that you invoke the `close` method of the `JarFile` instance that the `getJarFile` method of the `java.net.JarURLConnection` returns.

(13) Notes on using the URLConnection class

When the `java.net.URLConnection` class uses the `setUseCaches(boolean)` method to acquire the connection for the specified URL, you can specify whether or not to use the cache information. When the `setUseCaches(false)` method is specified for the `URLConnection` class, the target object is generated for each connection. When the `URLConnection` class is used from the code in the J2EE application, memory might be insufficient for the J2EE server JavaVM.

(14) Notes related to the loading of the native library

Do not use the `System.loadLibrary` method to load the native library from the servlets and JSPs. If the native library is loaded with the servlets and JSPs, the error `java.lang.UnsatisfiedLinkError` might occur depending on the constraints of the JNI specifications. If you must load the native library, create the container extension library that invokes the `System.loadLibrary` method and implement settings so that the container extension library is referenced from the servlets and JSPs. For creating the container extension library, see *14. Container Extension Library* in the *uCosminexus Application Server Common Container Functionality Guide*.

(15) Using user thread

You can generate and use threads from the servlets and JSPs that form the application. The thread that a user explicitly generates in a program is called a user thread.

The user threads are classified as follows depending on the operation method after generating the threads (lifecycle):

- Threads operating within the scope of the `service` method and the `init` method.
- Threads operating in the background of the `service` method and the `init` method.

Usage conditions of the user thread

- The user thread cannot be used in the Enterprise Bean (since the generation of threads from the Enterprise Bean is prohibited in the EJB specifications).
- The server modes of the Application Server that can use the user threads are as follows:

Table 6-8: Preconditions for the user threads (server mode)

Server modes		Availability
J2EE server mode	1.4 mode	Y
	Basic mode	N
Servlet engine mode		N

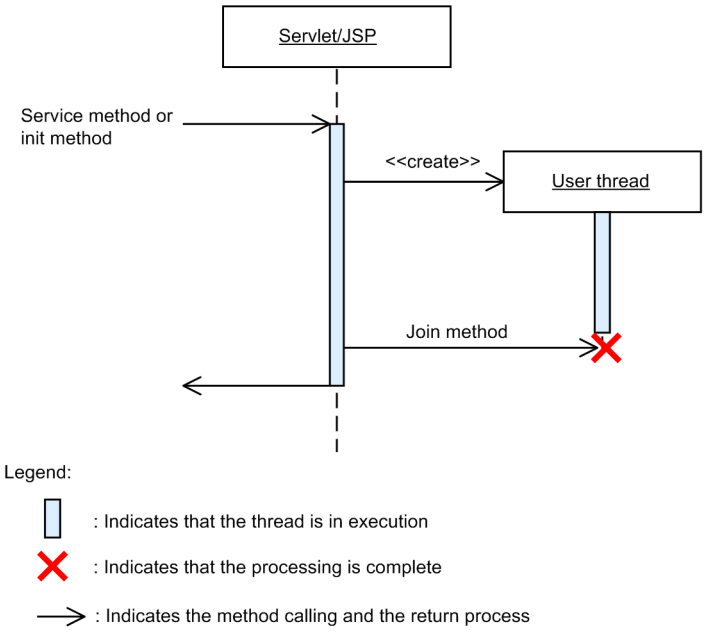
Legend:
Y: Available
N: You can use the functionality in the scope supported in the server mode (basic mode or servlet engine mode). However, you cannot use the connection pool. Note that these functionalities are provided to maintain compatibility.

The following points describe the lifecycle when using the user threads:

(a) When the threads operate within the scope of the service method and init method

In this model, the processing of the user thread is completed with the `service` method and the `init` method. The following figure shows the flow of processing in this model:

Figure 6-1: Processing when the thread operates within the scope of the service method and the init method

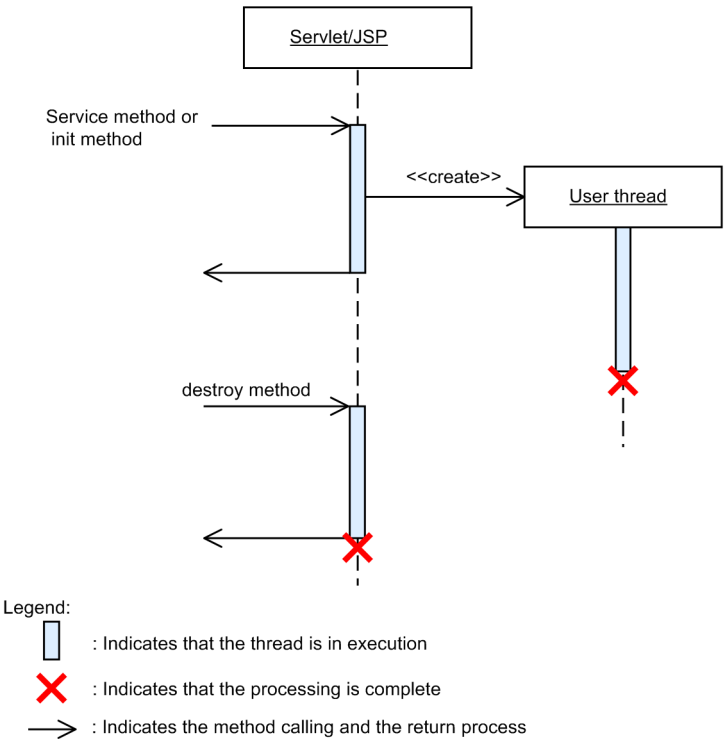


The user thread will be generated within the invocation scope of the `service` method and the `init` method. The `service` method and the `init` method wait for the processing of the user thread by the `join` method to complete, and then the methods are returned.

(b) When the threads operate in the background of the service method and the init method

In this model, the user thread is generated with the `service` method and the `init` method, and then the user thread is operated in the background. The following figure shows the flow of processing in this model:

Figure 6-2: Processing when the threads operate in the background of the service method and the init method



The `service` method and the `init` method that generated the user thread will be returned without waiting for the processing to complete after the user thread is generated. However, after the application is stopped, the J2EE service cannot be used from the user thread. Therefore, no problem occurs if a user thread is stopped by a `contextDestroyed` method of `javax.servlet.ServletContextListener` or a JSP or servlet `destroy` method invoked by the stopping of an application.

(16) Persistence of session information

The Web container does not support the persistence of the session information. Irrespective of whether the session information in the Web container is normal or abnormal, the information is lost once the Web container exits. If you want to maintain the session information, use the session failover functionality.

Furthermore, when the <distributable> tag is specified in `web.xml` or when the not Serializable object is registered as the session information, `IllegalArgumentException` does not occur.

(17) Messages output when the init method and destroy method are not overridden

If you initialize or terminate the servlets that do not override the `init` method and the `destroy` method, the log of the following format is output in the servlets log:

- Message ID: KDJE39037-I
- Message text: path="*aa....aa*" :*bb....bb*: init[#]

aa....aa

Indicates the context path starting with a forward slash (/).

bb....bb

Indicates the servlets name specified in the <servlet-name> tag of `web.xml`. In the case of servlets with default mapping, the name is `org.apache.catalina.INVOKER.`*class-name*.

#

In the case of the `init` method, the name is `init` and in the case of `destroy` method, the name is `destroy`. The output messages are the log output in the `init` method and `destroy` method of the

javax.servlet.GenericServlet class respectively. Therefore, these messages are not output in the servlets that override the init method or the destroy method.

Also, in the case of JSP, if the init method and the destroy method are not overridden in the base class of JSP specified in the extends attribute of the page directive, a similar message is output. In this case, servlets name is com.hitachi.software.web.servlet-name.jsp. If the extends attribute of the page directive is not specified in JSP, only the init method log is output and the destroy method log is not output.

However, for both servlet and JSP, when the init method and the destroy method of the superclass are invoked by overriding the init method and the destroy method, this message is output.

(18) javax.servlet.error.exception attribute of the javax.servlet.ServletRequest object

The javax.servlet.error.exception attribute of the javax.servlet.ServletRequest object is separately described as follows for the case in which exception is thrown in the Servlet and when exception is thrown in the JSP file:

(a) When an exception is thrown in the servlet

When the exception class thrown in the servlet is java.lang.Error or an inherited class

The exception of the javax.servlet.ServletException class is set in the javax.servlet.error.exception attribute of the javax.servlet.ServletRequest object. The exception thrown in the servlet can be acquired using the getRootCause method of the javax.servlet.ServletException class.

When the exception class thrown in the servlet is a class other than java.lang.Error or an inherited class

The exception thrown in the servlet is set in the javax.servlet.error.exception attribute of the javax.servlet.ServletRequest object.

(b) When an exception is thrown in the JSP file

■ When the error page is a JSP file

When the error page is specified in the <error-page> tag of web.xml

The error page specified in the <error-page> tag of web.xml is described separately for JSP 2.0 and later versions and JSP 1.2 version.

JSP 2.0 and later versions

The exception thrown in the JSP file is set in the javax.servlet.error.exception attribute of the javax.servlet.ServletRequest object.

JSP 1.2

If the exception class thrown in the JSP file is one of the following classes, the exception thrown in the JSP file is set in the javax.servlet.error.exception attribute of the javax.servlet.ServletRequest object.

- java.io.IOException or an inherited class
- java.lang.RuntimeException or an inherited class
- javax.servlet.ServletException or an inherited class

If the exception class thrown in the JSP file is other than the above-mentioned classes, the exception of the javax.servlet.ServletException class is set in the javax.servlet.error.exception attribute of the javax.servlet.ServletRequest object. The exception thrown in the JSP file can be acquired using the getRootCause method of the javax.servlet.ServletException class.

When the error page is specified in the errorPage attribute of the page directive

The error page specified in the errorPage attribute of the page directive is described separately for the case in which true is specified and when false is specified in the isErrorPage attribute of the page directive in the error page.

When true is specified in the isErrorPage attribute of the page directive in the error page

The exception thrown in the JSP file is set in the `javax.servlet.error.exception` attribute of the `javax.servlet.ServletRequest` object.

When false is specified in the isErrorPage attribute of the page directive in the error page

A value is not set in the `javax.servlet.error.exception` attribute of the `javax.servlet.ServletRequest` object.

■ When the error page is a Servlet

When the error page is specified in the <error-page> tag of web.xml

The exception thrown in a servlet is the same as in the case in which the error page is a JSP file when the error page is specified in the `<error-page>` tag of `web.xml`.

When the error page is specified in the errorPage attribute of the page directive

A value is not set in the `javax.servlet.error.exception` attribute of the `javax.servlet.ServletRequest` object.

(19) Operating Web applications containing binary data

Note the following items to the Web applications containing binary data:

- When executing the requests to the binary data sent from the client

  Do not acquire `PrintWriter` from the response object in the filter applied in the request to the binary data.
- When the servlets or JSPs that process the requests sent from the client are to be dispatched

  Do not acquire `PrintWriter` from the response object in the following locations:
  - In the filter applied in the request to the binary data
  - In the servlets or JSPs to be dispatched to the binary data

  Reference note

  Binary data means static contents in which the MIME type mapped to the extension does not begin with `"text/"` or static contents for which mapping does not exist.

(20) Notes related to response character encoding

If the character encoding of the response body in the JSPs or servlets is UTF-16 (16 bit UCS conversion format), the character encoding might not be displayed correctly by the browser. In this case, use UTF-16BE (big-endian byte order of the 16 bit UCS conversion format) or UTF-16LE (little-endian byte order of the 16 bit UCS conversion format) for the character encoding of the JSPs or servlets.

(21) Return values of the getServerName method and getServerPort method of the javax.servlet.ServletRequest interface

This point describes the return values of the `getServerName` method and the `getServerPort` method.

In Servlet 2.4 and later specifications, the return values of the `getServerName` method and the `getServerPort` method differ depending on the availability of the Host header. The following table lists the return values of the `getServerName` method and the `getServerPort` method in the Servlet 2.4 and later specifications:

Table 6-9: Return values of the getServerName method and getServerPort method (In Servlet 2.4 or later specifications)

Presence of Host header	Return value of the getServerName method	Return value of the getServerPort method
Yes	Part before the colon (:) of the Host header	Part after the colon (:) of the Host header
No	Resolved server name or IP address	Port number of the server that receives the connection with the client

In Application Server, the return values of the `getServerName` method and the `getServerPort` method are acquired depending on the combination of the HTTP requests and the functionality used. Note that when the Host header is not included in the HTTP 1.1 request, 400 error occurs according to the HTTP 1.1 specifications. Furthermore, the HTTP 1.1 specifications define that if the request URI of the request line is an absolute URI, use the host of the request URI for the host and ignore the contents of the `Host` header. Though not explicitly mentioned in the Servlet specifications, the host name included in the request line URI is given priority based on the HTTP specifications.

The following table lists the return values of the `getServerName` method and the `getServerPort` method obtained depending on the combination of the HTTP requests and the functionality used. For details on the return values of the `getServerName` method and `getServerPort` method when the gateway specification functionality is used, see *Table 6-9*.

Table 6-10: Return values of the getServerName method and getServerPort method (In the application server)

HTTP request		Functionality used	Return value of the getServerName method	Return value of the getServerPort method
Presence of the Host header	URI type of the request line			
Yes	Absolute URI	Web server integration	Host name of request line	Port number of request line
		In-process HTTP server	Host name of request line	Port number of request line
	Relative URI	Web server integration	Host name of Host header	Port number of Host header
		In-process HTTP server	Host name of Host header	Port number of Host header
No	Absolute URI	Web server integration	Host name of request line	Port number of request line
		In-process HTTP server	Host name of request line	Port number of request line
	Relative URI	Web server integration	Host name or IP address of the Web server[#2]	Port number of the Web server
		In-process HTTP server	Host name or IP address of the J2EE server[#1]	Port number of the in-process HTTP server

#1

Return value of the `java.net.InetAddress.getLocalHost` method or the `getHostName` method.

#2

Value specified in the `ServerName` directive when Cosminexus HTTP Server is used. For details on the ServerName directive, see the *uCosminexus Application Server HTTP Server User Guide*.

The following table lists the return values of the `getServerName` method and the `getServerPort` method when you use the gateway specification functionality:

Table 6-11: Return values of the getServerName method and getServerPort method when you use the gateway specification functionality (in the Application Server)

HTTP request		Functionality used	Return value of the getServerName method	Return value of the getServerPort method
Presence of the Host header	URI type of the request line			
Yes	Absolute URI	Web server integration	Host name of request line	Port number of request line
		In-process HTTP server	Host name of request line	Port number of request line
	Relative URI	Web server integration	Host name of Host header	Port number of Host header
		In-process HTTP server	Host name of Host header	Port number of Host header

HTTP request		Functionality used	Return value of the getServerName method	Return value of the getServerPort method
Presence of the Host header	URI type of the request line			
No	Absolute URI	Web server integration	Host name specified in the gateway specification functionality	Port number specified in the gateway specification functionality
		In-process HTTP server	Host name of request line	Port number of request line
	Relative URI	Web server integration	Host name specified in the gateway specification functionality	Port number specified in the gateway specification functionality
		In-process HTTP server	Host name specified in the gateway specification functionality	Port number specified in the gateway specification functionality

(22) Acquiring the root cause exception specified in the constructor of the javax.servlet.ServletException class

With the Application Server, you can acquire the root cause exception specified in the constructor `ServletException (String, Throwable)` or `ServletException (Throwable)` using the `getCause` method. Note that you can also acquire the exception using the `getRootCause` method. However, with versions prior to 07-60 version, the `getCause` method returns null.

This point describes the compatibility parameters and notes related to the acquisition of the root cause exception specified in the constructor of the `javax.servlet.ServletException` class.

- Compatibility parameters

  To perform the same operations as are operated with the versions prior to 07-60 version, specify `true` in the compatibility parameter `webserver.servlet_api.exception.getCause.backcompat` in the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file. For details on the parameters, see the *uCosminexus Application Server Definition Reference Guide*.

  The following table lists the differences in the return values of the `getCause` method and the `getRootCause` method depending on the value specified in the compatibility parameter `webserver.servlet_api.exception.getCause.backcompat`.

  Table 6-12: Differences in the return values of the getCause method and getRootCause method depending on the value specified in webserver.servlet_api.exception.getCause.backcompat

Method	Value specified in `webserver.servlet_api.exception.getCause.backcompat`	
	`true`	`false`
`getCause()`	N	Y
`GetRemoteUser()`	Y	Y

  Legend:

  Y: Returns the root cause exception

  N: Returns null

  Note that the contents specified in the compatibility parameter are also applied to the operations of the `getCause` method and the `getRootCause` method of the `javax.servlet.jsp.JspException` class.

- Notes

  When the root cause exception can be acquired by implementing the `getCause` method, you cannot invoke `initCause(Throwable)` for the `ServletException` object generated by the constructor `ServletException (String, Throwable)` or `ServletException (Throwable)`. If `initCause (Throwable)` is invoked, the `java.lang.IllegalStateException` exception is thrown.

(23) Notes related to the execution of the flush method for the javax.servlet.ServletOutputStream object

With the Application Server, even if the `flush` method is executed after executing the `close` method for the `javax.servlet.ServletOutputStream` object obtained from the `javax.servlet.ServletResponse` object, the `java.io.IOException` exception is not thrown.

(24) Normalizing request URIs

With Application Server, the character strings included in request URIs are used in the following matching processes after normalizing the request URIs:

- Matching context path and context root
- Matching URL pattern of servlets and JSPs
- Matching default mapping
- Matching static contents
- Matching URL pattern of filter
- Matching with the `<error-page>` tag of `web.xml` or with the `errPage` attribute of the `page` directive of JSPs
- Matching URL pattern for restricting access
- Determining URL for login authentication
- Forwarding and including requests
- Matching URL pattern for HTTP response compression filter
- Matching URL pattern to control the number of concurrently executing threads in the URL group
- Error page customization of the in-process HTTP server
- Request distribution by redirecting the in-process HTTP server

(25) Return value of the getRequestURI and getRequestURL methods of the javax.servlet.http.HttpServletRequest interface

The normalized URL is considered as a return value in the `getRequestURI` and `getRequestURL` methods of the `javax.servlet.http.HttpServletRequest` interface.

(26) Specifying a Servlet or JSP mapped to a URL in the welcome file

When a request URL must be forwarded to the `welcome` file without matching it with a Servlet or JSP mapped to the URL, the forwarding-destination `welcome` file is selected as follows in the Web container:

First of all, from the specified `welcome` file name, the static contents and JSP file candidate are selected on priority. If no corresponding item exists, the candidate of the Servlet or JSP for which URL mapping is performed is selected.

The notes on the `welcome` file are as follows:

■ Limitations based on the welcome file forwarding method

The `welcome` file is forwarded based on HTTP redirect (the browser redirects to HTTP status code 302). This forwarding method has some limitations that must be noted when designing the URL.

- When a POST request is received, the information of the request body sent from the browser cannot be inherited in the `welcome` file at the forwarding destination. Only when the posted information is in the form input format (`Content-Type` is `application/x-www-form-urlencoded`), the information can be inherited by assigning to the query string of the forwarding-destination URL of the `welcome` file created by the Web container. However, even in this case, consideration must be given to the fact that when the information of the request body is large in size, the forwarding-destination URL becomes too long, and the information is visible as is in the address bar of the browser in the form of the query string.

- When the doGet method is not implemented in the Servlet of the forwarding-destination welcome file, 400 Bad Request (for other than HTTP/1.1) or 405 Method Not Allowed (for HTTP/1.1) is displayed on the browser.
- When you invoke the include method of the javax.servlet.RequestDispatcher interface from the Web application, then in spite of specifying the directory in which the welcome file exists as the URL to be included, the contents of the welcome file of the forwarding destination are not inserted.

■ Adding the welcome file in an environment for which JSP pre-compilation is complete

When adding the JSP file specified in the welcome file to a Web application for which JSP pre-compilation is complete, you must again perform JSP pre-compilation after adding the JSP file. When you do not perform JSP pre-compilation again, the welcome file is not forwarded properly.

■ Specifying a servlet whose servlet class cannot be referenced in the welcome file

Do not specify a servlet whose servlet class cannot be referenced in the welcome file. When you specify a servlet whose servlet class cannot be referenced, the welcome file is not forwarded properly.

■ Requesting the welcome file to a path in which no directory exists

When a request is sent to the path of a directory that does not exist as a resource within the Web application, the welcome file is not forwarded even when the request URL ends with a forward slash (/).

(27) Starting and stopping order of the servlet, filter and listener

If you start a Web application, perform the initialization process in the order below before starting the receipt of requests, according to the Servlet 2.4 specifications. In *Cosminexus Application Server*, the initialization process is performed in the same order even in Web applications of Servlet2.3 or earlier versions. The order of starting the servlets, filters, and listeners at the time of starting the Web application is as follows:

1. Starting the listener (generating an instance#1, invoking methods of the @PostConstruct annotation and the contextInitialized method of ServletContextListener#2)
2. Starting the filter (generating an instance#1, invoking the methods of *@PostConstruct* annotation and the init method)
3. Starting Servlet/JSP specified in the load-on-startup tag (generating an instance#1, invoking method of the @PostConstruct annotation and the init method)

#1: In Servlet 3.0 or later, you can dynamically add the servlets, filters, and listeners by API calling. However, for the servlets, filters, and listeners for which the definition is added by the API calling that specifies the instances, the instance has already been generated and hence the Web container does not generate an instance.

#2: Even if an exception occurs while invoking the contextInitialized() method of the listener, the system outputs the message of KDJE39103-E and continues the process of starting the Web application.

Note that for servlets for which the execution of the initialization process at the time of Web application startup is not specified with the *load-on-startup* element in web.xml, the system generates the instances of the servlet and invokes the init() method at the time of the initial request execution.

At this time, the instance generation and the init() method invocation of the servlet is performed before the filter.

The order of stopping the servlets, filters, and listeners at the time of stopping the Web application is as follows:

1. Stopping the already started Servlet/JSP (invoking the destroy method or the methods of the @PreDestroy annotation)
2. Stopping the filter (invoking the destroy method or the methods of the @PreDestroy annotation)
3. Stopping the listener (invoking the methods of the @PreDestroy annotation)

(28) Accessing the static contents within a Web application

You can use any one of the methods such as GET, HEAD, POST, TRACE, and OPTIONS, to access the static contents within a Web application.

When you use the `POST` method, same details of the static contents are sent as a response, as when you use the `GET` method.

(29) Precautions related to character encoding

In the same Web application, use the same character encoding in the error page specified in `web.xml` as the character encoding used for the servlets and JSPs that use character encoding in the HTTP response.

(30) Return value when only the part after equal (=) sign is used in the query character string

When only the part after equal (=) sign is specified in the query character string of a request (for example, in the case of), the return value of the Servlet API that obtains the request parameter of `javax.servlet.ServletRequest` differs depending on the type of Web server in use.

The following are the respective return values of Servlet APIs for each type of the Web server:

- When using the Web server linkage functionality of a simple Web server
  - `getParameter` method

    When you specify a blank character (""), the parameter specified after the equal sign ("=") is returned.
  - `getParameterMap` method

    The `java.util.Map` object that includes the parameter for which a blank character ("") acts as a key, is returned.
  - `getParameterNames` method

    The `java.util.Enumeration` object that includes a blank character (""), is returned.
  - `getParameterValues` method

    When you specify a blank character (""), the parameter specified after the equal sign ("=") is returned.
- When using the in-process HTTP server
  - `getParameter` method

    Even if you specify a blank character (""), null is returned.
  - `getParameterMap` method

    A blank `java.util.Map` object is returned.
  - `getParameterNames` method

    A blank `java.util.Enumeration` object is returned.
  - `getParameterValues` method

    Even if you specify a blank character (""), null is returned.

(31) containsHeader method of javax.servlet.http.HttpServletResponse instance

The following response headers are sometimes automatically set to responses, by the Web container. You cannot use the `containsHeader` method of the `javax.servlet.http.HttpServletResponse` interface to check whether such response headers are set for responses.

- For Web server linkage
  - `Content-Length`
  - `Content-Type`
  - `Set-Cookie`
- For the in-process HTTP server
  - `Connection`
  - `Content-Language`
  - `Content-Length`
  - `Content-Type`

- `Date`
- `Server`
- `Set-Cookie`
- `Transfer-Encoding`

(32) Precautions related to the libraries of Application Server

If you include the libraries of Application Server in J2EE applications, it may lead to incorrect operations during the start and execution of the application import, due to reasons like a conflict in the library version. Therefore, do not include the libraries of Application Server in a J2EE application, except for cases where inclusion of the libraries is specified as a method of using the product.

6.2.2 Precautions for implementing servlets

This subsection describes the precautions for implementing servlets.

(1) Notes for using the I/O stream

- Specify a value in the range of `0x00` to `0xFF` in the argument of the `print(char c)` method of the `ServletOutputStream` class. If you specify a value outside the range, `java.io.CharConversionException` is thrown.
- In the `ServletInputStream` class, the `mark` method and `reset` method are not supported. Also, the `markSupported` method always returns `false`.
- If data is read from the `ServletInputStream` class and then forwarded, the data read from the forwarded `ServletInputStream` class is treated as the data that was read before forwarded. Also, if all the data is read from the `ServletInputStream` class before forwarded, the request parameter becomes null at the forwarded destination.

(2) Notes when setting locale

If `Locale.JAPANESE` is specified in the `setLocale` method of the `ServletResponse` class, charset of Content-Type header becomes Shift_JIS.

(3) Notes for acquiring URI

In the `getRequestURI` method of the `HttpServletRequest` class, the optimized URI is returned. For example, `xxx//yyy/zzz` is converted to `xxx/yyy/zzz` and `xxx/yyy/../zzz` is converted to `xxx/zzz`.

(4) Operations when reading of the POST data fails

If a Web server fails to read the POST data, the servlets operating in the Web container generate an `IllegalStateException` exception when invoking the following `ServletRequest` class methods:

- `getParameter` method
- `getParameterNames` method
- `getParameterValues` method
- `getParameterMap` method

Also, when a system receives form data having Content-Type as *multipart/form-data*, the KDJE39336-E message is output and an `IllegalStateException` exception occurs, at the time of invoking the above-mentioned methods or the following methods of the `HttpServletRequest` class. At this time, check whether the size of the received form data is correct and if the size is correct, revise the setup value of `webserver.connector.limit.max_post_form_data`.

- `getPart` method

- `getParts` method

(5) Notes when reporting events for changes in attributes

In the `ServletContextAttributeListener` interface, `HttpSessionAttributeListener` interface, and `ServletRequestAttributeListener` interface, events might be reported even when the attributes used internally by the Web container are added, deleted, and updated. Reference the attribute names of the reported events and ignore them if the attribute name is not used in the Web application.

(6) Notes for using the ServletContext interface

- `java.util.Set` obtained by the `getResourcePaths` method of the `ServletContext` interface is used for the reference. Do not add or delete elements for `java.util.Set`. If the `add`, `addAll`, `clear`, `remove`, and `removeAll` methods are used, the `IllegalStateException` exception might be thrown.
- In the argument of the `getContext` method of the `ServletContext` interface, specify the URL that uses the existing context root name. If a URL that uses a non-existent context root name is used, the operations might not function properly.
- In the `getResource` method and the `getResourceAsStream` method of the `ServletContext` interface, specify the resources included in the relevant Web application. If resources outside the relevant Web application are specified, the operations might not function properly.

(7) Notes for accessing the directory included in the Web application

When accessing the directory included in the Web application, do not specify the query string and POST data since you might not be able to acquire them from the redirect destination resource.

(8) Notes for using the ServletRequest interface

- The `getRemoteHost` method of the `ServletRequest` interface returns the host name of the client that sends the request, but if the settings are such that the Web server cannot resolve or does not resolve the host name, the IP address is returned.

  In the default settings, the Web server settings are not specified, so the IP address is returned. To acquire the host name, you must change the Web server settings. However, if you change the settings, the response might be delayed for resolving the host name. For details on how to change the Web server settings, see the *uCosminexus Application Server HTTP Server User Guide*.

(9) Notes on implementing processing that should not be executed multiple times in the process

If processing that should not be executed multiple times in one process is described in a servlet, specify settings such that the execution of the servlet and the processing is not parallel. Especially, in the initialization process for starting the communication with OTM, resident threads that do not exit even after the instance is deleted might be generated. For example, every time the `ORB.init` method that is the initialization function of Cosminexus TPBroker is invoked, a resident thread for monitoring is generated for the garbage collection and this thread lasts until the process ends. Therefore, if the `ORB.init` method is executed more number of times than necessary in one process, there might be adverse effects such as the unnecessary garbage collection processes increase and the performance of the entire system declines greatly.

To prevent such events, when you describe processing that you want to execute only once in one process in the servlet, you must first determine whether that processing is already executed in the process. Specifically, in an optional class, prepare a static variable as a condition flag that stores the status of whether certain processing is already executed. By executing the processing only if the value of the static variable means 'Not Executed' and changing the value to one that means 'Executed', you can limit the execution frequency of that processing to only once in one process. However, note the following two points:

- When using the static variable in the optional class, do not specify the following settings in `usrconf.properties` or `hitachi_web.properties`:

  - `webserver.context.reloadable=true`

- webserver.jsp.recompilable=true

If the above settings are specified, the instance of that class is automatically destroyed and re-generated, and therefore, the value of the static variable is also initialized along with this setting. For details on the settings of usrconf.properties and hitachi_web.properties, see the *uCosminexus Application Server Definition Reference Guide*.

- Specify the keyword synchronized in the method for executing this processing so that while a certain thread references the value of the static variable that is a condition flag and then changes the value, another thread will not execute similar processing. The following is an coding example in which this method is used such that the initialization function ORB.init method of Cosminexus TPBroker is invoked only once:

```
static org.omg.CORBA.ORB _orb=null;
public static synchronized org.omg.CORBA.ORB getORB(String[] args, Properties
props) {
 if(_orb==null) {
 _orb=org.omg.CORBA.ORB.init(args, props) ;
 }
return _orb;
}
```

(10) Precautions for using the gateway specification functionality

You can use the *gateway specification functionality* that reports the gateway information to the Web container and correctly redirects the information to the welcome file and FORM authentication window. For details on the gateway specification functionality, see *4.10 Notification of gateway information to a Web container*.

If you use the gateway specification functionality, some servlet API functions behave differently. For each used method, the following points describe the precautions for servlet API functions in the gateway specification functionality:

- sendRedirect method of the javax.servlet.http.HttpServletResponse interface

  If relative URL is specified in the argument and the request is without the Host header, the host name and port number of the redirect destination URL is the value specified in the gateway specification functionality. If relative URL is specified in the argument and if the scheme is considered as https in the gateway specification functionality, the scheme of the redirect destination URL is always https.

- getRequestURL method of the javax.servlet.ServletRequest interface

  If you specify the settings for scheme to be considered as https in the gateway specification functionality, the return value is always a URL beginning with https://.

- getServerName method of the javax.servlet.ServletRequest interface

  If the host name of the redirect destination URL is specified in the gateway specification functionality and if the request does not have a Host header, the return value is the specified value.

- getServerPort method of the javax.servlet.ServletRequest interface

  If the port number of the redirect destination URL is specified in the gateway specification functionality and if the request does not have a Host header, the return value is the specified value. If the host name of the redirect destination URL is specified in the gateway specification functionality and if the port number is omitted, the return value is 80 if the request scheme is http and 443 if the request scheme is https.

- getScheme method of the javax.servlet.ServletRequest interface

  If the scheme is considered as https in the gateway specification functionality, the return value is always https.

- isSecure method of the javax.servlet.ServletRequest interface

  If the scheme is considered as https in the gateway specification functionality, the return value is always true.

- getAttribute method of the javax.servlet.ServletRequest interface

  Even if the scheme is considered as https in the gateway specification functionality, the following attributes cannot be acquired:

  - javax.servlet.request.cipher_suite (if Microsoft IIS is used as the Web server, regardless of the use of the gateway specification functionality, this attribute cannot be acquired)
  - javax.servlet.request.key_size
  - javax.servlet.request.X509Certificate

(11) Notes on using the gateway

When gateways such as SSL accelerator and load balancer are used, the return value of the following servlet API functions is the IP address and host name of the gateway instead of the IP address and host name of the client.

- `getRemoteAddr` method of the `javax.servlet.ServletRequest` interface
- `getRemoteHost` method of the `javax.servlet.ServletRequest` interface

(12) Unique Hitachi attributes registered in ServletContext objects

The Web container registers the information required for controlling the Web application in the attributes of the `javax.servlet.ServletContext` object. The attribute names acquired by the `getAttributeNames` method of the `ServletContext` interface in the Web application also include the attribute names registered by the Web container.

When you register the attributes in the `ServletContext` object in the Web application, do not use the key names starting with the following strings:

- `org.apache.catalina`
- `com.hitachi.software.web`
- `jspx`

Also, attributes defined in the Java EE specifications are also added in `ServletContext`, so do not register the attributes with same key names.

(13) Notes on using the proxy acquisition method of the ServletRequest class

The following methods of the `javax.servlet.ServletRequest` interface are the methods for acquiring the information about the client who sent the request or the proxy passed last, but in an environment where the reverse proxy is used, the information to be acquired is the reverse proxy information.

- `getRemoteAddr` method
- `getRemoteHost` method
- `getRemotePort` method

(14) Notes on executing the reset method of the javax.servlet.ServletResponse interface

After executing the `getWriter` method of the `javax.servlet.ServletResponse` interface, if the `reset` method is executed, for the character encoding specified in the Content-Type of the HTTP response, specify the same character encoding once again by using one of the following API functions (all from the `javax.servlet.ServletResponse` interface):

- `setContentType` method
- `setLocale` method
- `setCharacterEncoding` method#

#

A method added in the Servlet 2.4 specifications.

In Servlet 2.4 and later specifications, when you set the character encoding by using these API functions, the API functions must be invoked before executing the `getWriter` method. However, only when the reset method is executed after executing the `getWriter` method, you can set the character encoding using these API functions until the `getWriter` method is invoked once again.

(15) Operations when 0 is specified in the argument of the setMaxInactiveInterval method

If you specify `0` in the argument of the `setMaxInactiveInterval` method of `javax.servlet.http.HttpSession` interface, the session does not timeout.

(16) mark operations of java.io.BufferedReader

When you use the in-process HTTP server functionality, `java.io.BufferedReader` obtained with the `getReader` method of `javax.servlet.ServletRequest` does not support mark operations. The `markSupported` method returns `false`.

(17) Operations when 1 is specified in the argument of the setVersion method

If you specify `1` in the argument of the `setVersion` method of `javax.servlet.http.Cookie` class, the Set-Cookie2 header is added to the response when the Web server integration functionality is used, but the Set-Cookie header is added when the in-process HTTP server functionality is used.

(18) Specifying the path in the getRequestDispatcher method

If a relative path that does not start with a forward slash (/) is specified in the argument of the `getRequestDispatcher` method of `javax.servlet.ServletRequest` interface, the path becomes the relative path from the URL pattern specified in the servlet mapping of this servlet. If the URL pattern ends with a forward slash (/), the path becomes the relative path from the parent directory.

For example, if you execute the `getRequestDispatcher` method by specifying `"hello.html"` from the servlet where the servlet mapping is specified in `"/a/b/"`, `"/a/hello.html"` is obtained.

(19) Notes on using the setBufferSize method to change the buffer size

The servlet buffer used for sending a response is maintained for each request processing thread. If you execute the `setBufferSize` method of `javax.servlet.ServletResponse` interface to change the buffer size, the changed buffer size is applied to all the requests processed by the relevant thread containing the other Web applications on the same J2EE server. When you use the `setBufferSize` method of `javax.servlet.ServletResponse` to change the buffer size, estimate the memory usage after considering that the *memory-of-buffer-size* × *number-of-request-processing-thread* is secured. Note that the buffer that has been acquired once is valid until update is performed by the `setBufferSize` method from the Web application for each processing thread.

(20) Notes for Content-Type header of the HTTP response

If the Content-Type is not explicitly specified in a servlet with the `setContentType` method of the `javax.servlet.ServletResponse` interface, the Content-Type header is not created. Therefore, you cannot check the character encoding of the HTTP response from the `"charset="` field of the Content-Type header.

(21) Precautions related to the getId method of the javax.servlet.http.HttpSession interface

In the Web applications compliant with specifications prior to Servlet 2.4 version, the operations when the `getId` method of the disabled `javax.servlet.http.HttpSession` object is invoked differ in the Servlet specifications and the Application Server. The operations to be performed in these cases are as follows:

Servlet specifications

The `java.lang.IllegalStateException` exception is thrown.

Application server

Null is returned.

(22) Precautions related to the methods of the javax.servlet.ServletRequest interface and javax.servlet.http.HttpServletRequest interface

When the information acquired with the methods listed in the following table is output in the response, the information must be sanitized:

Table 6-13: Methods that require the acquired information to be sanitized

Interface name	Method name
javax.servlet.ServletRequest	getCharacterEncoding()
	getContentType()
	getInputStream()
	getParameter(java.lang.String *name*)
	getParameterMap()
	getParameterNames()
	getParameterValues(java.lang.String *name*)
	getProtocol()
	getReader()
	getServerName()
javax.servlet.http.HttpServletRequest	getCookies()
	getHeader(java.lang.String *name*)
	getHeaderNames()
	getHeaders(java.lang.String *name*)
	getMethod()
	getPathInfo()
	getPathTranslated()
	getQueryString()
	getRequestedSessionId()
	getRequestURI()
	getRequestURL()
	getServletPath()

(23) Precautions related to the getLocale(getLocales) method of the javax.servlet.ServletRequest interface

The `java.util.Locale` object that can be obtained with the `getLocale` method or `getLocales` method of the `javax.servlet.ServletRequest` interface is created from the value of the Accept-Language header of the HTTP request.

The Web Container checks if the locale of the Accept-Language header value (ISO language code, ISO country code, or variants) contains characters other than alphabetic characters. If the locale contains non-alphabetic characters, the locale is determined to be invalid, the KDJE39546-W message is output in the message log for each invalid locale, and the locale is ignored.

If an Accept-Language header containing an invalid locale is received, the `getLocale` method or `getLocales` method returns only the `java.util.Locale` object with the correct locale. If all the locales specified in the Accept-Language header are invalid, it is considered that the Accept-Language header does not exist and the default server locale is returned.

(24) Return value of the getServerName method of javax.servlet.http.HttpServletRequest interface

The return value of the `getServerName` method of the `javax.servlet.http.HttpServletRequest` interface differs from the value of the `Host` header set by the HTTP client, when the `Host` header is rewritten by uCosminexus Application Server HTTP Server User Guide and reverse proxy.

(25) Setting the response header in an include destination servlet of Servlet 2.4 or earlier version

In Servlet 2.4 or earlier versions, the settings of all response headers in the include destination servlet are ignored according to the specifications. However, in Application Server, the settings of the response header in `getSession` are enabled even when Servlet 2.4 is used.

(26) Content format of static contents without MIME mapping

For static contents without MIME mapping, `Content-Type` is not assigned.

(27) Timing of accessing the HTTP session

When a request to which a session ID is added is sent to the Web container, the HTTP session access time is refreshed to the current time. However, this update is not carried out when a session timeout has already occurred or the session has been disabled.

The HTTP session time is used for the following cases:

- The return value of the `getLastAccessedTime()` method of the `javax.servlet.http.HttpSession` interface

  The `getLastAccessedTime()` method of the `javax.servlet.http.HttpSession` interface returns the timing of accessing the HTTP session at the time of previous update.

- HTTP session timeout

  When the difference between the current time and the time of accessing the HTTP session exceeds the timeout value, a timeout occurs for the HTTP session. As the timeout monitoring for the HTTP session is executed at an interval of 30 seconds, a timeout does not always occur at the correct timeout time.

(28) Precautions related to acquiring a value of the Content-Length header

When an HTTP request does not include the Content-Length header, the return value of the `getContentLength()` method of `javax.servlet.ServletRequest` and the return value when "*Content-Length*" is specified in the argument, in the `getIntHeader()` method of `javax.servlet.http.HttpServletRequest` differ in the Servlet specifications and Application Server. The respective return values are as follows:

Servlet specification
: Returns `-1`.

Application Server
: Returns `0`.

6.2.3 Precautions related to the specifications that are added or changed in the Servlet 3.0 specifications

This subsection describes precautions on using specifications that were added or changed in Servlet 3.0 for Application Server. For details on Servlet 3.0 and Servlet 2.5 specifications, see the respective specifications (Servlet 3.0 specifications and Servlet 2.5 specifications).

(1) Specifications where Servlet 3.0 functionalities and Application Server functionalities are combined

This subsection describes specifications where functionalities added in Servlet 3.0 and functionalities of Application Server are combined.

(a) Functionality for setting the Web application version

You cannot set the version to 3.0 in the Web application version setting functionality.

(b) JSP pre-compile under META-INF/resources

Although it is now possible with Servlet 3.0 specifications to include JSP files in the hierarchy of the `META-INF/resources` inside a JAR file, you cannot execute JSP pre-compile for Web applications that have a JSP in the `META-INF/resources` hierarchy. If you execute the JSP pre-compile, the JSP in the `META-INF/resources` hierarchy is not pre-compiled, and an error occurs on executing the request.

(c) Reloading an application that uses Servlet 3.0

Reloading of an application that uses Servlet 3.0 functionalities has the following constraints:

- If an application contains a JAR file that has static contents (also includes the JSP) in `META-INF/resources`, even if you update and reload the JAR file, the pre-update static contents in `META-INF/resources` of the JAR file can only be accessed. If you reload an application that contains a JAR file that has static contents in `META-INF/resources`, the KDJE39556-W message is output.

! Important note

With version 09-00-02 or earlier, if you reload the following applications, the operation is not guaranteed.

- When you reload an application that contains an implementation class of the `ServletContainerInitializer` interface, the KDJE39557-W message is output.
- When you reload an application that contains servlets added by the dynamic servlet definitions, filters, or listeners, the KDJE39558-W message is output.

(d) Using the run-as functionality in servlets defined by using API

You cannot use the run-as functionality in a servlet defined by using an API. Settings are ignored if you:

- use the `@RunAs` annotation in a servlet class.
- call the `setRunAsRole` method of the `ServletRegistration.Dynamic` interface.

(e) Monitoring the execution time of servlets or filters defined by using APIs

You cannot use the execution time monitoring functionality of the J2EE application for the servlets or filters defined by using APIs. Even if you define the *method-observation-timeout* tag in the `cosminexus.xml` file, the tag is ignored.

(f) Continuation of the HTTP session when the Web application is invoked

The operation regarding the continuation of an HTTP session when the Web application is invoked is the same as in the case of Servlet 2.5 specifications. For details, see *6.2.4(6) Continuation of the HTTP session when the Web application is invoked*.

(g) Request redirecting when the cookie name of the session is changed

When the Web container is deployed in a cluster configuration, you can distribute requests by using the Redirector. The Redirector references the session ID added to each request and distributes the requests in such a way that the requests from the same Web client are redirected to the same Web container. However, if you change the session cookie name from the default name `JSESSIONID` to some other name, requests from the same Web client might not be redirected to the same Web container.

(2) Collaborating Servlet 3.0 and CDI

When you want to combine Servlet 3.0 and CDI to use them together, use a web.xml that is compatible with Servlet 3.0. A `web.xml` of any other version cannot be used in combination with CDI.

(3) Combining filters or listeners defined by using an API with @PostConstruct or @PreDestroy annotations

You cannot use the `@PostConstruct` or `@PreDestroy` annotations in filters or listeners defined with APIs. Such annotations are ignored if used.

(4) When there is no content to include

In Servlet 3.0, default servlets (servlets provided by the container to provide static content) are included. If there is no content to be included, the system throws the `FileNotFoundException` exception. If the exception is not handled in the user program, the response is not committed and the status code 500 is returned.

Application Server does not support this specification. The same status code 404 as in versions prior to 08-70 is returned.

(5) Using escape sequence

Do not use an escape sequence (`\b`, `\t`, `\n`, `\f`, `\r`,`\"` ,`\'`,`\\`) for input characters in API or the properties newly added in Servlet 3.0. If you use the escape sequence, new lines might get randomly added in the log thus affecting the layout of the log.

(6) Defining dynamic servlets

The following points describe how the behavior of dynamic servlets differs from the Servlet 3.0 specifications when the dynamic servlets are defined and used in Application Server:

(a) Defining servlets, filters, or listeners

- When you add a filter by using both API and `web.xml`, and set the filter mapping method isMatchAfter defined in the API for the other filter mapping to false, the API definition is called first when a request is processed.
- When you define a listener by using both API and `web.xml`, the listener defined in the `web.xml` is read first when the application starts and the listener is generated. When you want to delete the listener, the definition in the API is read first.
- When you specify a NULL character ("") for the URL pattern and call the `addMapping(String... urlPatterns)` method of the `javax.servlet.ServletRegistration` interface, only the context name is used for accessing the servlet. If the context name is also a NULL character (""), the servlet cannot be accessed.
- Do not call the `addServlet()` method by specifying a servlet name already defined in `web.xml` or the same servlet name as that created by the servlet class with the `@WebServlet` annotation.
- If the `setMultipartConfig()` method is called and the `@MultipartConfig` annotation is specified for the same servlet in a user application, the value specified in the `setMultipartConfig()` method is used.

  Similarly, if the `setServletSecurity()` method is called and the `@ServletSecurity` annotation is also specified for the same servlet in a user application, the value specified in the `setServletSecurity()` method is used.
- You cannot acquire or set a role by using the `getRunAsRole()` or the `setRunAsRole(String roleName)` method. Furthermore, no warning or error message is output.
- If you register a servlet or a filter with an already registered name, the KDJE39552-W message is output when the application starts and the operation continues.
- If a URL pattern contains a linefeed code, the KDJE39555-W message is output when the application starts and the operation continues.

(b) Defining by using the ServletContainerInitializer interface

When you define using the `ServletContainerInitializer` interface in Application Server, the following points differ from the Servlet 3.0 specifications:

- You can deploy a JAR file that contains a class with the `ServletContainerInitializer` interface implemented in the following locations:

  1. The `WEB-INF/lib` directory in the WAR file included in the application
  2. A location other than `WEB-INF/lib` directory

  If you want to deploy a JAR file that contains a class with the `ServletContainerInitializer` interface implemented in a location other than the `WEB-INF/lib` directory, you must specify the absolute path of the JAR file in the following property of `usrconf.properties`.

  `webserver.ServletContainerInitializer_jar.include.path`

  The following example shows how to set the path:

  ```
  webserver.ServletContainerInitializer_jar.include.path=C:/Program Files/
  HITACHI/Cosminexus/foo/lib/bar.jar
  ```

- The search range of the `@HandlesTypes` annotation class is limited to the WAR file that contains the implementation class of the `ServletContainerInitializer` interface. Search can be executed in classes within the `WEB-INF/classes` directory and the JAR files present in the `WEB-INF/lib` directory of WAR.
- If the JAR file that contains a class with the `ServletContainerInitializer` interface implemented cannot be read or processed, the KDJE39548-E message is output when you attempt to start the application and the application fails to start.
- When search is performed for a class that extends or implements a class specified with the `@HandlesTypes` annotation, or for a class to which the annotation of the class specified with the `@HandlesTypes` annotation is attached, if the class fails to load, the KDJE39549-W message is output and the start processing of the application continues.
- If a JAR file that you specified in `webserver.ServletContainerInitializer_jar.include.path` and that contains a class with the `ServletContainerInitializer` interface implemented is not found, or if a JAR file or files cannot be read, the KDJE39553-W message is output when the application starts and the process continues. The KDJE39553-W message is output for each JAR file specified in Web server. `ServletContainerInitializer_jar.include.path`.
- If the JAR file that you specified in `webserver.ServletContainerInitializer_jar.include.path` and that contains a class with the `ServletContainerInitializer` interface implemented is not found in the class path, the KDJE39554-W message is output when the application starts and the process continues. The KDJE39554-W message is output for each JAR file that is specified in `webserver.ServletContainerInitializer_jar.include.path`.

Also, implementation class information of the `ServletContainerInitializer` interface, defined in the `javax.servlet.ServletContainerInitializer` file, is read as described below.

- Only the class coded in the first line is read and the classes coded from second line onwards are ignored.

(7) About file upload

If you use the file upload functionality with Application Server, the following points differ from the Servlet3.0 specifications:

- If the multipart `config` element is set in both web.xml and `@MultipartConfig` annotations, the element defined in the `web.xml` is given priority.
- If you execute the file upload with a request other than the one with the request type multipart/form-data, the `javax.servlet.ServletException` exception is thrown. The following table describes the mapping between the type of requests that execute file upload and the corresponding execution result.

Table 6-14: Type of requests that execute file upload and the corresponding execution result

Request type specified in `web.xml` or `@MultipartConfig` annotation	Execution result
Other than mime-multipart	The `javax.servlet.ServletException` exception is thrown.
mime-multipart other than multipart/form-data	The `javax.servlet.ServletException` exception is thrown.
multipart/form-data	File upload is executed. No exception is thrown.

- When you define `MultipartConfigElement` in the program, it functions in the same way as when you define it in `web.xml`. However, if you specify a NULL character ("") in the location, files are stored in the work directory of the Web application.
- If the size of the uploaded file is greater than the value specified in the *file-size-threshold* tag in the `web.xml` or the `fileSizeThreshold` attribute of the `@MultipartConfig` annotation, temporary files are generated for a part of the objects included in the request. A temporary file is saved with a unique name that begins with `upload` and has a `.tmp` extension.

  Example: `upload__5f8d5c62_1316c5ef08b__8000_00000012.tmp.`

  Temporary files are deleted when the request is completely processed. However, if a temporary file is open, or if the J2EE server is not shut down properly, or if the system shuts down, the file might not get deleted. Delete the file manually from the path specified in the location.

  If a temporary file is generated with the same name as a file created manually by the user, the file created by the user is overwritten by the temporary file.
- You can use the `Part` objects inside a request. The `Part` objects are deleted once the request is completely processed. If you save the `Part` objects in the session for later use, the operation is not guaranteed.

(8) Deploying static resources

This point describes the deployment of static resources and the points that differ from the Servlet 3.0 specifications when the static resources are used with Application Server.

- When a JAR file that contains static resources cannot be read or processed, the KDJE39548-E message is output when you attempt to start the application and the application fails to start.
- If a specified JAR file has an invalid format, the KDJE39550-E message is output when the application starts, and the process continues.
- If an invalid JAR file or a file that cannot be read is included in WEB-INF/lib, the KDJE39551-W message is output when the application starts, the JAR file is ignored, and the process continues.

(9) About API

The following points differ from the Servlet 3.0 specifications if APIs are used with Application Server.

- Among the APIs that are added in Servlet3.0, the APIs that are not listed in the following table can be used with Application Server 09-00. For details on APIs, see the Servlet3.0 specifications. If you use any of the non-supported APIs listed in the following table, Application Server throws the `java.lang.UnsupportedOperationException` exception.

  Table 6-15: Non-supported APIs (Servlet 3.0)

No.	Package	Class	Interface/ Class	Method	Functionality
1	`javax.servlet`	AsyncContext	Interface	All methods	Asynchronous servlet
2		AsyncListener	Interface	All methods	Asynchronous servlet
3		ServletContext	Interface	getJspConfigDescriptor	JSP
4		ServletRequest	Interface	startAsync	Asynchronous servlet

No.	Package	Class	Interface/ Class	Method	Functionality
5	javax.servlet	ServletRequest	Interface	isAsyncStarted	Asynchronous servlet
6				isAsyncSupported	Asynchronous servlet
7				getAsyncContext	Asynchronous servlet
8				getDispatcherType	Asynchronous servlet
9		AsyncEvent	Class	All methods	Asynchronous servlet
10		ServletRequestWrapper	Class	startAsync	Asynchronous servlet
11				isAsyncStarted	Asynchronous servlet
12				isAsyncSupported	Asynchronous servlet
13				getAsyncContext	Asynchronous servlet
14				getDispatcherType	Asynchronous servlet
15		Registration	Interface	setAsyncSupported	Asynchronous servlet
16	javax.servlet.descriptor	JspConfigDescriptor	Interface	All methods	JSP
17		JspPropertyGroupDescriptor	Interface	All methods	JSP
18		TaglibDescriptor	Interface	All methods	JSP

- You cannot use the APIs added in Servlet 3.0 in Web applications that are compatible with Servlet 2.5. Because the program does not check if the correct APIs are being used, the operation is not guaranteed.
- You cannot use SSL in the SessionTrackingMode. You can use only COOKIE and URL with Application Server.

(10) Omitting the web.xml

You can omit `web.xml` in Servlet 3.0 for the following applications:

- Web applications that contain static contents and JSP only (excluding listeners, servlets, and filters)
- Web applications with listeners, servlets, and filters defined with annotations

(11) Changing an HTTP Cookie name that indicates a session ID

When changing the HTTP Cookie name that indicates the session ID, you cannot specify `csfcfc` in the Cookie name.

Also, the following are the conditions for the characters that you can use. If you violate these conditions, KDJE39559-W is output. This may also lead to incorrect operations of the session, therefore use the characters that conform to the conditions.

- Use ASCII characters, excluding the following characters.
  `"#", "(", ")", "<", ">", "@", ",", ";", ":", "\", "/", """, "[", "]", "?", "=", "{", "}"`
- You cannot use the space or control characters.
- You cannot use `$` at the beginning of a character string.

(12) HTTP Cookie names available in Web applications

You cannot use the following names for HTTP Cookies you use in a Web application. Note that the Cookie names are not case sensitive.

- An HTTP Cookie name used as the session ID of an HTTP session. The default name is `JSESSIONID`.
- An HTTP Cookie name that represents a global session. The default name is `GSESSIONID`.

- The same name as an HTTP Cookie name appended with a server ID by using the append server ID functionality of the J2EE server.
- The HTTP Cookie name *csfcf* used internally in the J2EE server.

(13) Class specified in the <servlet-class> element

If you want to omit the *servlet-class* element in the `web.xml`, you must include the subclass of `javax.servlet.http.HttpServlet` in which you have specified the `@WebServlet` annotation in the Web application, and specify the name of the servlet for which you have omitted the *servlet-class* element in the name attribute. If, the corresponding HttpServlet class is not available, the KDJE39339-E message is output and the servlet class fails to load.

(14) Class specified in the *filter-class* element

If you want to omit the *filter-class* element in `web.xml`, you must include the implementation class of `javax.servlet.Filter` in which you have specified the `@WebServlet` annotation in the Web application, and specify the name of the filter for which you have omitted the *filter-class* element in the `filterName` attribute. If the corresponding Filter class is not available, the KDJE39340-E message is output and the application fails to start.

(15) Changing the Path attribute of an HTTP Cookie indicating a session ID

When you change the `Path` attribute of an HTTP Cookie indicating a session ID, and the HTTP Cookie name or the `Path` attribute indicating a session ID duplicates that of another Web application, the HTTP Cookie value indicating the session ID might be overwritten or deleted, and the HTTP session can no longer be inherited.

(16) Characters that can be used in the Path attribute and the Domain attribute of an HTTP Cookie indicating a session ID

Note that the following are the conditions for characters that can be used in the `Path` attribute and `Domain` attribute of an HTTP Cookie indicating a session ID. If the following conditions are not satisfied, the KDJE39559-W message is output. If you use invalid characters, it might result in an abnormal behavior of the session. Therefore, use characters that satisfy the following conditions:

- For the `Path` attribute, you can use only ASCII characters excluding control characters and semicolons (;).
- For the `Domain` attribute, you can use only alphanumeric characters, hyphens (-), and periods (.).

6.2.4 Precautions related to added and changed specifications in the Servlet 2.5 specifications

This subsection describes the precautions for using the specifications that were added and changed in Servlet 2.5 on the Application Server. For details on the Servlet 2.5 specifications and Servlet 2.4 specifications, see the respective specifications (Servlet 2.5 specifications and Servlet 2.4 specifications).

(1) XML schema changes of web.xml

On the Application Server, the XML schema of `web.xml` is changed according to the Servlet 2.5 specifications.

(2) Precautions related to omission of web.xml

The description about the omission of `web.xml` is added in the Servlet 2.5 specifications.

On the Application Server, when the Web application only contains JSP and static contents, you can omit `web.xml`. The Web application version in which `web.xml` is omitted is considered as 2.5. Also, when `web.xml` is omitted, you can omit the WEB-INF directory.

This point describes the precautions when `web.xml` is omitted. For the omission of `web.xml`, see *11.4.5 Operations related to Web applications in which web.xml is omitted* in the *uCosminexus Application Server Common Container Functionality Guide*.

(a) Omitting web.xml when servlets and filters are included in the Web application

The servlets and filters require the mapping to be defined in `web.xml`, so if `web.xml` is omitted, the servlets and filters do not operate.

(b) Omitting web.xml when listener is included in the Web application

The following listeners need to be defined in `web.xml`, so if `web.xml` is omitted, the listeners do not operate:

- `javax.servlet.ServletContextListener`
- `javax.servlet.ServletContextAttributeListener`
- `javax.servlet.ServletRequestListener`
- `javax.servlet.ServletRequestAttributeListener`
- `javax.servlet.http.HttpSessionListener`
- `javax.servlet.http.HttpSessionAttributeListener`

However, the following listeners are not defined in `web.xml`, so even if `web.xml` is omitted, the listeners operate:

- `javax.servlet.http.HttpSessionBindingListener`
- `javax.servlet.http.HttpSessionActivationListener`

Also, if the listener is defined in the tag library, even if `web.xml` is omitted, the listener is executed.

(c) Setting the Web application properties in the execution environment

If `web.xml` is omitted, you cannot set properties with the server management commands and property files.

(d) Settings for mapping

Even if `web.xml` is omitted, the settings for mapping of files with extensions `jsp` and `jspx`, the default welcome file, session timeout, and default `mime-mapping` are enabled. Regardless of the settings for enabling or disabling the default servlet mapping, you cannot use the default servlet mapping.

(e) Using filters

In Web applications where `web.xml` is omitted, you cannot use filters. Therefore, functionality that require the built-in filter, such as the HTTP response compression functionality and the memory session failover functionality cannot be used.

(3) Specifying a null character string in the <load-on-startup> element

In the `web.xml` schema defined in the Servlet 2.5 specifications, the value that can be specified in the `<load-on-startup>` element was changed.

On the Application Server, the value that can be specified in the `<load-on-startup>` element is controlled by the servlets corresponding to the version information of the Servlet specifications described in `web.xml`. The following table lists the values that can be specified in the `<load-on-startup>` element for each servlet version.

Table 6-16: Values that can be specified in the <load-on-startup> element

Servlet version	Specifiable values
Servlet 2.5	Integer or null character string. If a null character string is specified, the servlet is not loaded as in the case when the `<load-on-startup>` element is not specified.
Servlet 2.4	Integer. Specified value same as 07-60.

Note that if you specify a value that cannot be specified, the importing of the J2EE application will fail.

(4) Scope of specifying the HTTP method in the <security-constraint> element

In the `web.xml` schema defined in the Servlet 2.5 specifications, the contents that can be specified in the `<http-method>` element in the `<security-constraint><web-resource-collection>` element were changed. The following table lists the specifiable contents:

Table 6-17: Contents specifiable in the <http-method> element in the <security-constraint><web-resource-collection> element

Servlet version	Specifiable contents
Servlet 2.5	One-byte alphanumeric characters (0-9, A-Z, a-z) and special characters (! # $ % & ' * + - . ^ _ ` | ~) can be specified one or more times.
Servlet 2.4	Either GET, POST, PUT, DELETE, HEAD, OPTIONS, or TRACE.

Specify the HTTP method of the request in the `<http-method>` element in the `<security-constraint><web-resource-collection>` element.

The HTTP method of the request supported in the application server is same as the contents supported in the servlet corresponding to the version information of the Servlet specifications described in `web.xml`. However, the HTTP method of the request supported in the Web application compliant with the Servlet 2.5 specifications differs depending on the Web server used. The HTTP method of the request supported in the Application Server is as follows, for each Web server:

Table 6-18: HTTP method of the request supported in the Application Server

Servlet version	Web server used	Specifiable contents
Servlet 2.5	Cosminexus HTTP Server and Microsoft IIS	Some of the methods that can be used in HTTP1.1#.
	In-process HTTP server	All the methods that can be used in HTTP1.1.
Servlet 2.4	Cosminexus HTTP Server, Microsoft IIS, In-process HTTP server	Either GET, POST, PUT, DELETE, HEAD, OPTIONS, or TRACE.

#

For details on the methods that can be specified, see *Appendix A.2 Error status codes returned by the Redirector*.

If you specify a value that cannot be specified, an attempt to import the J2EE application will fail.

(5) Using annotations

Application Server supports the annotations defined in the Servlet 2.5 specifications. For using annotations, see *12. Using Annotations* in the *uCosminexus Application Server Common Container Functionality Guide*.

(6) Continuation of HTTP session when the Web application is invoked

The Servlet 2.5 specifications contain a description to continue the session in both the following cases when the session is created in the servlets and JSPs called in the cross context:

- Invoking cross context by a direct request for the associated context
- Invoking cross context by dispatch (invocation of the `forward` method and `include` method of the `javax.servlet.RequestDispatcher` interface)

However, in the Application Server, you cannot continue an HTTP session used when the Web application is invoked.

(7) Specifying all the servlets in filter mapping

The Servlet 2.5 specifications contain a description about the specification of one-byte asterisk (`*`) in the `<servlet-name>` element in the `<filter-mapping>` element of `web.xml`.

By specifying an asterisk (*) in the <servlet-name> element in the <filter-mapping> element of web.xml, you can invoke the filter for the requests to all the servlets.

An example of definition in web.xml is as follows:

```
<filter-mapping>
<filter-name>All Filter</filter-name>
<servlet-name>*</servlet-name>
</filter-mapping>
```

In this example, the filter with filter name All Filter is invoked for the requests to all the servlets.

! Important note

In the Web applications compliant with the Servlet 2.4 specifications, the one-byte asterisk (*) specified in the filter mapping did not have a special meaning. However, in the Web applications compliant with the Servlet 2.5 specifications, asterisk (*) implies all the servlets. Note that there is no method to specify only the servlets with servlet name asterisk (*).

In the application server, if one-byte asterisk (*) is specified in the <filter-mapping><servlet-name> element of web.xml, a filter is invoked for all the requests to the Web applications containing the relevant web.xml.

(8) Disabling the invocation of the setCharacterEncoding method of the javax.servlet.ServletRequest interface

The Servlet 2.5 specifications clearly specify that the invocation of the setCharacterEncoding method is disabled after invoking the getReader method of the javax.servlet.ServletRequest interface.

The Application Server follows the Servlet 2.5 specifications regardless of the servlet version. The invocation of the setCharacterEncoding method is disabled after the getReader method is invoked and the following contents are not changed:

- Character encoding used in the BufferedReader object acquired with the getReader method
- Return value of the getCharacterEncoding method

With versions prior to the version 07-60, the return value of the getCharacterEncoding method was changed by invoking the setCharacterEncoding method after the getReader method was invoked, but in 08-00, the return value is not changed regardless of the servlet version.

(9) Changing the URL that forms the return value of the getRequestURL method of the javax.servlet.http.HttpServletRequest interface

The Servlet 2.5 specifications clearly specify that when the getRequestURL method of the javax.servlet.http.HttpServletRequest interface is invoked in the forwarding destination, the path used for acquiring the javax.servlet.RequestDispatcher object is applied as the URL that becomes the return value and not the URL path specified in the client.

On Application Server, the URL that becomes the return value is determined as per the servlet corresponding to the version information of the Servlet specifications described in web.xml.

In the Servlet 2.5 specifications, the path specified in the argument of the getRequestDispatcher method of the javax.servlet.ServletRequest interface invoked for acquiring the javax.servlet.RequestDispatcher object becomes the return value. In the Servlet 2.4 specifications, the path of the request URL becomes the return value.

(10) Operations when the getId method of the javax.servlet.http.HttpSession interface is invoked

In the Servlet 2.5 specifications, the specification is deleted related to the java.lang.IllegalStateException exception that is thrown when the getId method of the disabled javax.servlet.http.HttpSession object is invoked.

On Application Server, the value that is returned when the `getId` method of the disabled `javax.servlet.http.HttpSession` object was invoked, is controlled by the servlet as per the version information of the Servlet specifications coded in `web.xml`.

In the case of the Servlet 2.5 specifications, the return value is session ID and in the case of the Servlet 2.4 specifications, the return value is null same as in the case of version 07-60.

(11) Invocation of cross context between Web applications with different servlet versions

When the cross context is invoked between Web applications with different servlet versions, depending on the servlet version in the invocation source and invocation destination, there are methods in which the operations of the invocation destination are different. The following table describes the operations for each method at the invocation destination of the cross context when the cross context is invoked between the Web applications with different servlet versions:

Table 6-19: Operations at the invocation destination of the cross context between the Web applications with different servlet versions (getRequestURL method of the javax.servlet.http.HttpServletRequest interface)

Servlet version of the Web application		Operations at the invocation destination of the cross context	Reference
Invocation source	Invocation destination		
Servlet 2.5	Version prior to Servlet 2.4	The operations follow the Servlet 2.5 specifications.	*6.2.4 (9)*
Versions prior to Servlet 2.4	Servlet 2.5	The operations follow the Servlet 2.4 specifications.	

Table 6-20: Operations at the invocation destination of the cross context between the Web applications with different servlet versions (getId method of the disabled javax.servlet.http.HttpSession object)

Servlet version of the Web application		Operations at the invocation destination of the cross context	Reference
Invocation source	Invocation destination		
Servlet 2.5	Version prior to Servlet 2.4	The operations follow the Servlet 2.5 specifications and return the session ID.	*6.2.4 (10)*
Versions prior to Servlet 2.4	Servlet 2.5	The operations follow the Servlet 2.4 specifications and return null.	

(12) Setting a response header in the include destination servlet

According to the Servlet 2.5 specifications, the settings of the response header in the include destination servlet are enabled only for `getSession`. However, in Application Server, the settings of the response header in `getSession` are enabled even when Servlet 2.4 is used.

6.2.5 Precautions related to added and changed specifications in the Servlet 2.4 specifications

This subsection describes the precautions for using the specifications that has been added and changed in Servlet 2.4 on the Application Server. For details on the Servlet 2.4 specifications and Servlet 2.3 specifications, see the respective specifications (Servlet 2.4 specifications and Servlet 2.3 specifications).

(1) Using X-Powered-By header

The X-Powered-By header added in the Servlet 2.4 specifications is not added to the response.

(2) Notes on using the forward method of the java.servlet.RequestDispatcher interface

If you execute the `forward` method of the `javax.servlet.RequestDispatcher` class acquired with the `getRequestDispatcher` method of the `javax.servlet.ServletRequest` interface and `javax.servlet.ServletContext` interface, the following key attributes are added to the request object. However, the key attributes are not added in the `forward` method of the `RequestDispatcher` object acquired with the `getNamedDispatcher` method of the `javax.servlet.ServletContext` interface.

- `javax.servlet.forward.request_uri`
- `javax.servlet.forward.context_path`
- `javax.servlet.forward.servlet_path`
- `javax.servlet.forward.path_info`#1
- `javax.servlet.forward.query_string`#2

#1

This attribute is not added when the HTTP request received by the Web container does not contain the additional path information.

#2

This attribute is not added when the request URI of the HTTP request received by the Web container does not contain the query string.

These attributes are added by the Web container. The event of adding an attribute is not reported to `javax.servlet.ServletRequestAttributeListener`. For details on the values of the added attributes, see the Servlet 2.4 specifications.

(3) Deprecated javax.servlet.SingleThreadModel interface

The `javax.servlet.SingleThreadModel` interface is no longer recommended from the Servlet 2.4 specifications.

With the Application Server, you can use the `javax.servlet.SingleThreadModel` interface regardless of the Web application version. However, see the Servlet 2.4 specifications, note the reason for which this interface is deprecated, and then use the interface.

(4) setLocale method of the javax.servlet.ServletResponse interface

The character encoding is set in the Content-Type header of the HTTP response by the `setLocale` method of the `javax.servlet.ServletResponse` interface. In the Servlet 2.4 specifications, the conditions for enabling the set character encoding have been changed.

The conditions enabled in the Application Server is described separately for Servlet 2.4 and later versions and Servlet 2.3.

Servlet 2.4 and later versions

All the following conditions must be fulfilled:

- The character encoding is set before the HTTP response is committed.
- The character encoding is set before the `getWriter` method is invoked.
- The character encoding is set before the `setCharacterEncoding` method is invoked.
- The character encoding is not set by the `setContentType` method.

When the conditions are not fulfilled, the `setLocal` method does not set the character encoding of the response only by setting the locale in the `ServletResponse` class.

Servlet 2.3

The following condition must be fulfilled:

- The character encoding is set before the HTTP response is committed#.

#

- When the character encoding is set before the response is committed, the character encoding is set regardless of whether the setting is specified before or after the `getWriter` method is invoked.
- When the character encoding is set before the response is committed, the character encoding is set regardless of whether the character encoding is set with the `setContentType` method.

(5) javax.servlet.UnavailableException

The `javax.servlet.UnavailableException` exception indicates that usage is permanently unavailable. The specifications of the HTTP response code when the servlets and JSPs that threw the `javax.servlet.UnavailableException` exception are accessed have been post scripted in the Servlet 2.4 specifications.

The HTTP response codes when the servlets and JSPs that threw the exception are accessed in the Application Server are described separately for Servlet 2.4 and later versions and Servlet 2.3.

Servlet 2.4 and later versions

404 error.

Servlet 2.3

503 error.

(6) sessionDestroyed method of the javax.servlet.http.HttpSessionListener interface

The timing for invoking the `sessionDestroyed` method of the `javax.servlet.http.HttpSessionListener` interface has been changed in the Servlet 2.4 specifications.

The timing when this method is invoked on the Application Server is described separately for Servlet 2.4 and later versions and Servlet 2.3.

Servlet 2.4 and later versions

The method is executed before the session is destroyed.

Servlet 2.3

The method is executed after the session is destroyed.

Note that when session timeout is disabled, the listener related to the session is reported in the following order. The order for each Web application version is as follows:

In Servlet 2.4 and later versions

1. `sessionDestroyed()` method of the `javax.servlet.http.HttpSessionListener` interface
2. `valueUnbound()` method of the `javax.servlet.http.HttpSessionBindingListener` interface
3. `attributeRemoved()` method of the `javax.servlet.http.HttpSessionAttributeListener` interface

In Servlet 2.3

1. `valueUnbound()` method of the `javax.servlet.http.HttpSessionBindingListener` interface
2. `attributeRemoved()` method of the `javax.servlet.http.HttpSessionAttributeListener` interface
3. `sessionDestroyed()` method of the `javax.servlet.http.HttpSessionListener` interface

(7) sendRedirect method of the javax.servlet.http.HttpServletResponse interface

The conditions for using the `sendRedirect` method of the `javax.servlet.http.HttpServletResponse` interface have been changed in the Servlet 2.4 specifications.

To execute this method on the Application Server normally, all the following conditions must be fulfilled:

- The method is executed before the response is committed.
- An appropriate URL is specified in the argument.

The controlling of errors if these conditions are not fulfilled is described separately for Servlet 2.4 and later versions and Servlet 2.3.

Servlet 2.4 and later versions

If the conditions are not fulfilled, the `java.lang.IllegalStateException` exception is thrown.

Servlet 2.3

- When the response is already committed
  The `java.lang.IllegalStateException` exception is thrown.
- When an invalid URL is specified in the argument
  The response code is 404.

(8) Status message of HTTP status code 302

In the Servlet 2.4 specifications, `SC_FOUND` is added in the `javax.servlet.http.HttpServletResponse` interface as the constant indicating HTTP status code 302. Furthermore, for downward compatibility, `SC_MOVED_TEMPORARILY` defined in the Servlet 2.3 specifications can be used as it is.

On the Application Server, `SC_FOUND` and `SC_MOVED_TEMPORARILY` can be used regardless of any Web application version.

Note that the status message `Found` is used in the Web container in the following cases:

- When 302 is returned in the Web application
- When the default error page is output (in the HTML text)

(9) HttpSession timeout during the execution of the service method of the servlet

In the Servlet 2.4 specifications, specifications are post scripted for the timeout in the `javax.servlet.http.HttpSession` interface during the execution of the `service` method.

On Application Server, regardless of the Web application version, the `HttpSession` does not timeout when the requests are being processed in the Web application.

Also, by controlling the number of concurrently executing threads in the Web applications or URL groups, the `HttpSession` does not timeout even when the request is pending. However, note that in the case of a pending request due to the controlling of the number of concurrently executing threads in the Web container, the `HttpSession` times out.

(10) Control when exception occurs in the listener

The description about the occurrence of an exception in the listener is added in the Servlet 2.4 specifications.

The control when an exception occurs in the listener if the Application Server is used is described separately for Servlet 2.4 and later versions and Servlet 2.3.

Servlet 2.4 and later versions

Even when there are multiple listeners for processing the relevant events, the listener after the one in which the exception occurred is not executed.

Servlet 2.3

The thrown exception is caught by the Web container. If multiple listeners are registered, the registered listeners are executed serially, as in normal cases, after the exception is caught.

(11) Commonly used external library (Extension)

The description about the handling of the `MANIFEST` file described when using the external library in the Web application has been changed in the Servlet 2.4 specifications.

On the Application Server, the existence of the MANIFEST file and the contents of the MANIFEST file are not checked regardless of the Web application version.

(12) Using the cross context between the Web applications with different servlet versions

The following table lists the operations after the requests using the cross context are forwarded and included between the Web applications with different servlet versions:

Table 6-21: Operations after forwarding and after including

Item No.	Addition functionality of Servlet 2.4 specifications/ Functionality with differences in the Web application version	Operations at the request forward destination/ include destination	
		forward/ include from 2.4 to 2.3#1	forward/ include from 2.3 to 2.4#2
1	Application of filters during forward and during include	When you want to further implement forward or include from the servlets or JSPs after forward/ include is implemented, the Servlet 2.4 specifications are applied and you can use the filter.	When you want to further implement forward or include from the servlets or JSPs after forward/ include is implemented, the Servlet 2.3 specifications are applied and you cannot use the filter.
2	Invocation of javax.servlet.ServletRequestAttributeListener	The Servlet 2.4 specifications are applied and you can use the listener when you add attributes to the request.	The Servlet 2.3 specifications are applied and you cannot use the listener when you add attributes to the request.
3	JSP compilation	JSP compilation is executed as an application corresponding to Servlet 2.3.	JSP compilation is executed as an application corresponding to Servlet 2.4.
4	setLocale method of the javax.servlet.ServletResponse interface	The Servlet 2.4 specifications are applied and when all the following conditions are fulfilled, the settings for character encoding are enabled: • The character encoding is set before the response is committed. • The character encoding is set before the getWriter method is invoked. • The character encoding is set before the setCharacterEncoding method is invoked. • The character encoding is not set with the setContentType method.	The Servlet 2.3 specifications are applied and the settings for character encoding are applied if specified before the response is committed.
5	Dispatch to servlets and JSPs that throw the javax.servlet.UnavailableException indicating that the usage is permanently unavailable	The Servlet 2.3 specifications are applied and a response with status 503 is returned.	The Servlet 2.4 specifications are applied and a response with status 404 is returned.
6	sessionDestroyed method of the javax.servlet.http.HttpSessionListener interface	The Servlet 2.4 specifications are applied and the method is executed before the HTTP session is destroyed.	The Servlet 2.3 specifications are applied and the method is executed after the HTTP session is destroyed.
7	Specification of an invalid URL in the sendRedirect method of the javax.servlet.ht	The Servlet 2.4 specifications are applied and the java.lang.IllegalStateException exception is thrown.	The Servlet 2.3 specifications are applied and status 404 is set in the response.

Item No.	Addition functionality of Servlet 2.4 specifications/ Functionality with differences in the Web application version	Operations at the request forward destination/ include destination	
		forward/ include from 2.4 to 2.3#1	forward/ include from 2.3 to 2.4#2
7	`tp.HttpServletResponse` interface	The Servlet 2.4 specifications are applied and the `java.lang.IllegalStateException` exception is thrown.	The Servlet 2.3 specifications are applied and status 404 is set in the response.
8	Listener definition to be used	In the case of the following listeners, the listener defined in the application at the `forward` or `include` destination will operate: • `javax.servlet.ServletContextAttributeListener` In the case of the following listeners, the listener defined in the application that invokes the `forward` or `include` method will operate: • `javax.servlet.ServletRequestAttributeListener` • `javax.servlet.http.HttpSessionListener` • `javax.servlet.http.HttpSessionAttributeListener`	In the case of the following listeners, the listener defined in the application at the `forward` or `include` destination will operate: • `javax.servlet.ServletContextAttributeListener` In the case of the following listeners, the listener defined in the application that invokes the `forward` or `include` method will operate: • `javax.servlet.http.HttpSessionListener` • `javax.servlet.http.HttpSessionAttributeListener`
9	Operations when an exception occurs in the listener in the application at the forward or include destination when multiple listeners are defined in `web.xml` for processing the relevant events	In the case of the following listeners, the Servlet 2.4 specifications are applied and the listeners after the one in which an exception was thrown are not executed: • `javax.servlet.ServletRequestAttributeListener` • `javax.servlet.http.HttpSessionListener` • `javax.servlet.http.HttpSessionAttributeListener` In the case of the following listeners, the Servlet 2.3 specifications are applied, the thrown exception is caught, and then the process moves to the next registered listener, as in normal operations: • `javax.servlet.ServletContextAttributeListener`	In the case of the following listeners, the Servlet 2.4 specifications are applied and the listeners after the one in which an exception was thrown are not executed: • `javax.servlet.ServletContextAttributeListener` In the case of the following listeners, the Servlet 2.3 specifications are applied, the thrown exception is caught, and then the process moves to the next registered listener, as in normal operations: • `javax.servlet.http.HttpSessionListener` • `javax.servlet.http.HttpSessionAttributeListener`
10	Status code of response that displays the error page specified in `web.xml`	The Servlet 2.4 specifications are applied and the response of the status code used when an error occurs is returned.	The Servlet 2.3 specifications are applied and a response with status 200 is returned.

#1

Indicates the cases when forward or include is implemented from applications corresponding to Servlet 2.4 to applications corresponding to Servlet 2.3.

#2

Indicates the cases when forward or include is implemented from applications corresponding to Servlet 2.3 to applications corresponding to Servlet 2.4.

(13) When calling Session Bean of EJB 3.0 from the Web application of Servlet 2.4

When you call an EJB 3.0 Session Bean from a Servlet 2.4 Web application, do not specify the `ejb-ref` tag or `ejb-local-ref.` tag in `web.xml`. Instead, specify the `@EJB` annotation or `@EJBs` annotation in the servlet class.

The J2EE server executes DI of Enterprise Bean for the servlet.

6.2.6 Precautions for implementing JSPs

This subsection describes the precautions for implementing JSPs.

(1) Notes for using the include directive

- To include a file in the include directive of the JSP file, specify encoding in the JSP file that forms the include source.
- To include a static file such as HTML in JSP, use the include directive instead of the include action. Use the include action to include dynamic pages such as JSPs and servlets.

(2) Notes for using the tag library

- When you create a tag library, make sure that you add the package name with the package statement at the beginning of the class file that describes the tag library. If the package name is not added, that tag library does not function normally.
- Specify either `NESTED`, `AT_BEGIN`, or `AT_END` in the `<scope>` tag element in the `<variable>` tag in the tag library descriptor (TLD file). If other values are specified, the default value `NESTED` is assumed and executed.
- When you implement the tag handler of the tag library, specify settings such that the `doStartTag` method, the `doAfterBody` method, and the `doEndTag` method return the return value defined in the specifications. If a value other than the return value defined in the specifications is returned, the default return value is assumed. The default return value is the return value returned when the methods of the `javax.servlet.jsp.tagext.TagSupport` or `javax.servlet.jsp.tagext.BodyTagSupport` class are not overridden. For example, in the class that implements the `BodyTag` interface (or class that inherits the `BodyTagSupport` class), if the `doStartTag` method returns `EVAL_PAGE` as the return value, it is assumed that the default return value `EVAL_BODY_BUFFERED` is returned.
- When the Web application version is 2.3 and when a custom tag with attributes specified in the tag handler of the tag library is implemented, if you specify the same attribute in the JSP custom tag multiple times, the `setter` method of the relevant tag handler is invoked for the specified number of times. If a `setter` method that overwrites a normally specified value is implemented, the attribute value coded later is enabled.
- Do not specify a null character string as a tag element in the tag library descriptor (TLD) file. Examples of specifying null character strings are cases where no string is coded between the beginning tag and ending tag (for example, `<param-name></param-name>`) or cases where a tag with a null element (for example, `<param-name />`) is coded. If a null character string is specified for a tag element, operation is not guaranteed.
- Use an absolute path to specify the `xsi:schemaLocation` attribute in the Tag Library Descriptor (TLD) file.
- When you use the extension element of tags or a tag library in the TLD file, and specify the `xsi:schemaLocation` attribute with a relative path, operation is not guaranteed.

(3) Notes for coding the <%=%> tag

When using the `<%= %>` tag in JSP, make sure that you do not include a semicolon (;) in the tag. If the semicolon is entered, an error occurs during JSP compilation.

(4) Access based on URL specification and mapping definition

When you access the JSP file path directly by specifying the URL and when you access the path using a URL with the mapping defined, separate instances are generated in each case. Therefore, note that the `jspInit` method is executed in each instance. If the specification in the in the `<load-on-startup>` tag is to load the instance during startup, the instance loaded during startup is same as the one that is accessed using the URL with the mapping defined.

(5) Notes for using the <jsp:plugin> tag

Make sure that you specify the plugin action or the code attribute of the <jsp:plugin> tag in the JSP document on the JSP page. If omitted, a compilation error occurs.

(6) Version attributes in the JSP document

In the JSP document, describe the information about the version used as the attribute of the <jsp:root> tag. However, the scope of functionality available in JSP is in accordance with the version information of web.xml of the Web application containing the relevant JSP.

For example, even if the JSP document describes <jsp:root version="1.2">, you can use JSP EL added in the JSP 2.0 specifications.

(7) Mapping of URI and TLD files specified in the uri attribute of the taglib directive

The URI specified in the uri attribute of the taglib directive is mapped using one of the following methods as per the JSP specifications. You cannot map the same URI to a different TLD file. If the URI is duplicated, the following numbering is considered as the priority order and the mapping with a higher priority order is enabled:

1. Implicit mapping of JSTL and JSF URI (Web application of Servlet 2.5 or later specifications)
2. Mapping of URI specified in the <taglib-uri> element of the <taglib> element in web.xml with the TLD file specified in the <taglib-location> element
3. Mapping of the URI specified in the <uri> element of the TLD file in the Web application with the same TLD file
4. Mapping of the URI specified in the <uri> element of the TLD file stored under the /META-INF directory of the library JAR (in the case of the cjjspc command, the JAR file specified in the -classpath option) with the same TLD file

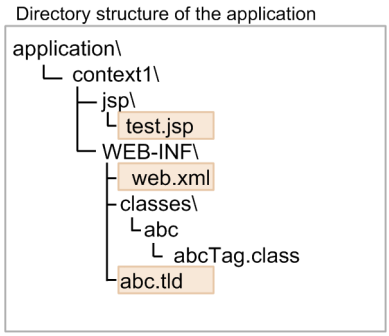
The examples of description of directory structures and files used when the URI and TLD files are mapped as specified from step 2 to 4 are as follows:

(a) Mapping of the URI specified in the <taglib-uri> element of the <taglib> element in web.xml and the TLD file specified in the <taglib-location> element (in 2.)

An example of description of the directory structure and files in the case of point 2 is as follows:

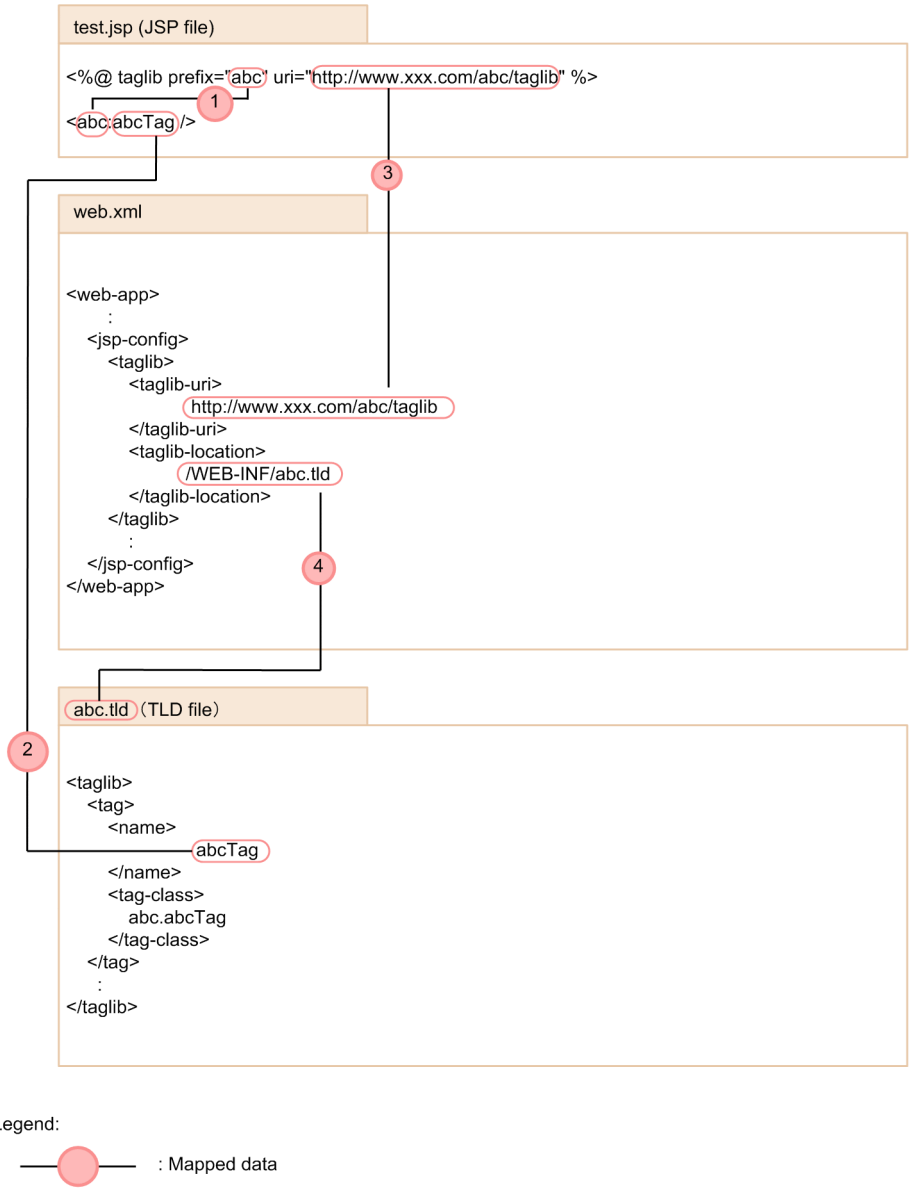
In the case of point 2, the TLD file is stored in the Web application. The example of the directory structure in point 2 is as follows:

Figure 6-3: Directory structure (in the case of point 2.)



The following figure shows the mapping of the URI specified in the uri attribute of the taglib directive and the TLD file for this directory structure.

Figure 6-4: Mapping of the URI and TLD file (in the case of point 2.)



The correspondence of data in the figure is explained below:

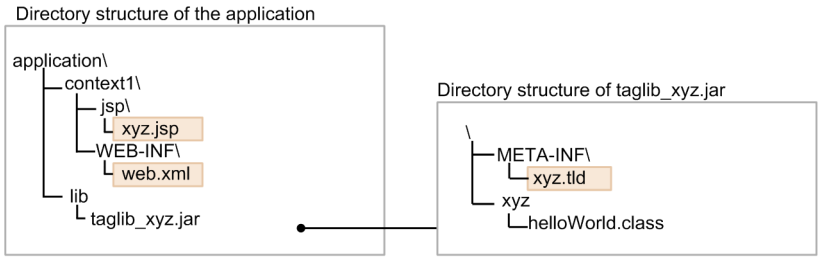
1. The prefix specified in the `taglib` directive is specified in the tag prefix.
2. The value specified in the `<name>` element of the TLD file is associated with the JSP file prefix.
3. The `uri` specified in the `taglib` directive of the JSP file is specified in the `<taglib-uri>` element of `web.xml`.
4. The name of the TLD file to be mapped is specified in the `<taglib-location>` element of `web.xml`.

(b) When mapping the URI specified in the <uri> element of the TLD file and the same TLD file

An example of description of the directory structure and files in the case of points 3 or 4 is described here.

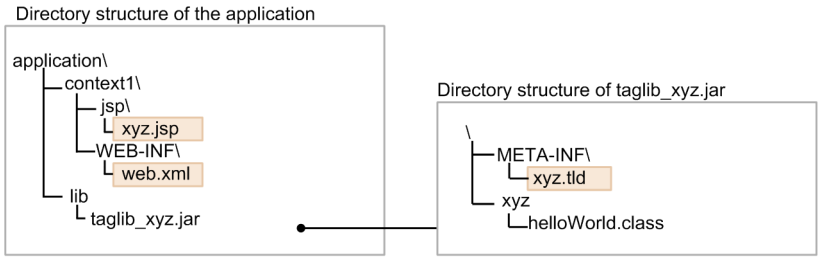
In the case of point 3, the TLD file is stored in the Web application. The example of the directory structure in point 3 is as follows:

Figure 6-5: Directory structure (in the case of point 3)



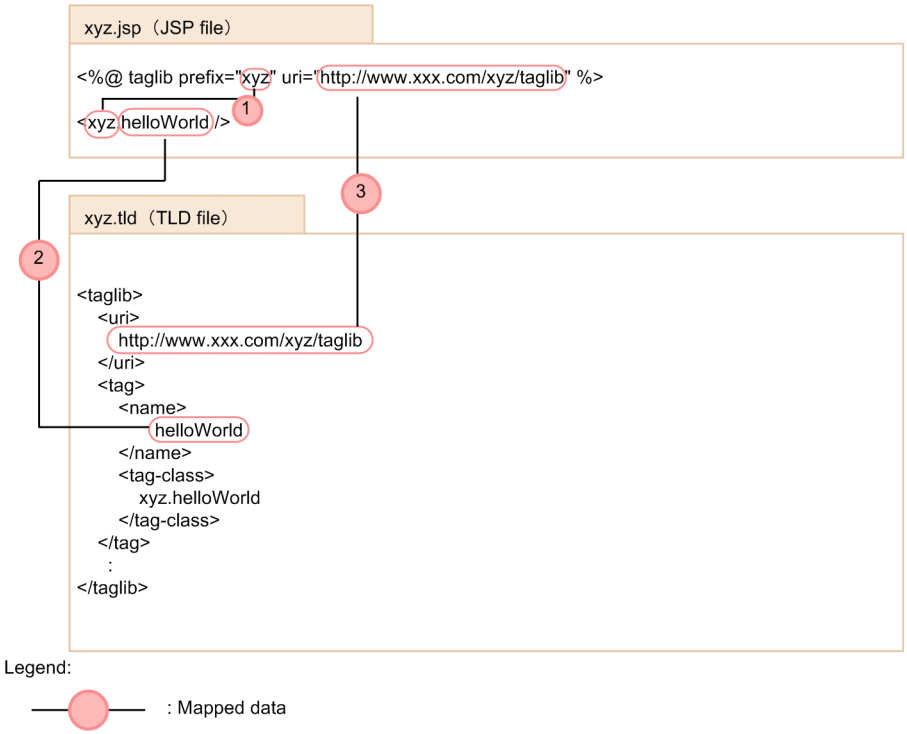
In the case of point 4, the TLD file is stored under the `/META-INF` directory of library JAR. The example of the directory structure in point 4 is as follows:

Figure 6-6: Directory structure (in the case of point 4)



The following figure shows the mapping of the URI specified in the `uri` attribute of the `taglib` directive and the TLD file in the directory structure for the points 3 or 4:

Figure 6-7: Mapping of the URI specified in the <uri> element of the TLD file and the same TLD file (in the case of points 3 or 4)



The correspondence of data in the figure is explained below:

1. The prefix specified in the `taglib` directive is specified in the tag prefix.

2. The value specified in the <name> element of the TLD file is associated with the JSP file prefix.
3. The uri specified in the taglib directive of the JSP file is specified in the <url> element of TLD file.

If a duplicate URI is detected, the following messages are output for the Web applications and the relevant mapping is ignored:

Table 6-22: Messages output and the output conditions in the case of URI duplication

Message ID	Output conditions
KDJE39314-W	In the case of the J2EE server mode, servlet engine mode, or cjjspc command This message is output when the contents of the <uri> element described in the TLD files are duplicated with the contents of the <taglib-uri> element in web.xml or the contents of the <uri> element described in another TLD file. In the case of the cjjsp2java command This message is output when the contents of the <uri> element described in the TLD file are duplicated with the uri contents specified in the command argument -taglib or the contents of the <uri> element described in another TLD file.
KDJE39315-W	This message is output when the contents of the <uri> element described in the TLD file are duplicated with the contents of the <uri> element described in another TLD file.
KDJE39316-W	This message is output when another <taglib> element is specified containing the <taglib-uri> element that duplicates with the contents of <taglib> element specified in web.xml.
KDJE39325-W	This message is output when the contents of the <taglib-uri> element of web.xml or the contents of the <uri> element described in another TLD file is duplicated with the URI of the tag library (JSTL, and JSF) in the Java EE specifications.
KDJE39326-W	This is message is output when the contents of the <uri> element of the TLD file stored in the library JAR (in the case of the cjjspc command, JAR file specified in the -classpath option) are duplicated with the contents of the <taglib-uri> element of web.xml or the contents of the <uri> element described in another TLD file.

(8) Notes for including a dynamic page in JSP

To include dynamic pages such as JSPs and servlets in JSP, use the include action instead of the include method of javax.servlet.RequestDispatcher.

(9) Coding the internal subset to the DOCTYPE declaration in the TLD file

Do not code the internal subset for the DOCTYPE declaration in the TLD file (tag library descriptor).

You can only specify absolute URI as the URI specified in the xsi:schemaLocation attribute with the tag element extension or the element extension of the tag library. Do not reference definitions other than DTD/ XML schema defined in the Java EE specifications with purposes other than tag element extension or the element extension of the tag library.

(10) Specifying the external subset URI in the JSP document and XML tag file

When you specify the DOCTYPE declaration, you can only specify the absolute URI as the external subset URI. Also, when you reference the external entity defined in the XML1.0 specifications, you can only specify the absolute URI as the URI specified in the external entity declaration. You cannot reference the external subset and external entity with the relative URI specified.

(11) Values of the javax.servlet.jsp.jspException attribute of the javax.servlet.ServletRequest object

When an exception is thrown in the JSP file and if the error page is specified in the errorPage attribute of the page directive, the exception is described in the JSP specifications if the error page is set in the

`javax.servlet.jsp.jspException` attribute of the `javax.servlet.ServletRequest` object, but the exception is also set when the error page is specified in the `errorPage` attribute of the page directive.

(12) Precautions related to the TLD file version

In 07-00 or later versions, the version of the TLD file (JSP version to which the TLD file corresponds) is checked during JSP translation. Therefore, if the TLD file version is a JSP of a level higher than the version of JSP and TLD file corresponding to the Web application version, an error occurs in JSP translation. Also, in JSP 1.2 and in JSP 2.0 or later specifications, the schema language must be defined in the TLD file.

The TLD file version is determined from the schema language described in the TLD file. However, if the schema language is not defined, the TLD file version is determined from the Web application version.

The following table lists the versions of TLD file when the schema language is not defined in the TLD file:

Table 6-23: TLD file version when the schema language is not defined

Version of the Web application	TLD file version
2.2	1.1
2.3	1.2
2.4	2.0
2.5	2.1

(13) Character encoding supported in JSP

This point describes the character encoding that can be used in the JSP file and tag file.

(a) In the case of the JSP page

The character encoding that can be used on the JSP page and the specification method of character encoding is described below:

- Character encoding that can be used on the JSP page

  The character encoding supported by JavaVM can be used. For details on the character encoding supported by JavaVM, see the JDK documentation.

  However, the character encoding where the alphanumeric characters, such as UTF-16, are expressed in multiple bytes, must fulfill the following two conditions:

  - The Web application is compliant with Servlet 2.4/ JSP 2.0 or later specifications.
  - BOM is added at the beginning of the JSP file.

- Specification method of character encoding

  You can specify the character encoding you want to use in the JSP page by choosing one or more methods from the methods described below:

  - Add BOM at the beginning of the JSP page (in the case of Web applications compliant with Servlet 2.5/JSP 2.1 or later specifications).
  - Specify the character encoding in the `<page-encoding>` element in the `<jsp-property-group>` element of `web.xml` (in the case of Web applications compliant with Servlet 2.4/JSP 2.0 or later specifications).
  - Specify the character encoding in the pageEncoding attribute of the page directive (in the case of Web applications compliant with Servlet 2.3/JSP 1.2 or later specifications).
  - Specify the character encoding in the contentType attribute of the page directive.

  The strings that can be specified are the character encoding described in the canonical name for java.nio API and canonical name for java.lang API and their alias names.

  Make sure that the character encoding you want to specify matches with the character encoding used on the JSP page.

(b) In the case of the JSP document (in the case of the Web applications compliant with Servlet 2.4 or later specifications)

The character encoding that can be used in the JSP documents in the Web application compliant with the Servlet 2.4 or later specifications and the specification method of the character encoding to be used is described below:

- Character encoding that can be used in the JSP document

  You can use the character encoding# supported by the Cosminexus XML Processor.

  However, for JSP documents without the extension jspx, if the <is-xml> element is not specified in the <jsp-property-group> element of web.xml, ISO-10646-UCS-4 cannot be used in the character encoding of the JSP document.

  However, the character encoding where the alphanumeric characters, such as UTF-16, are expressed in multiple bytes, must fulfill the following two conditions:

  - BOM is added at the beginning of the JSP document
  - If ISO-10646-UCS-2 is used, ISO-10646-UCS-2 of big-endian is used

- Specification method of character encoding

  Specify the character encoding used in the JSP document in the encoding attribute of the XML declaration. The specifiable strings are the character encoding# supported by the Cosminexus XML Processor.

  However, for JSP documents without the extension jspx, if the <is-xml> element is not specified in the <jsp-property-group> element of web.xml, specify the character encoding you want to use by choosing one or more methods from the methods described below:

  - Specify the character encoding in the encoding attribute of the XML declaration.
  - Specify the character encoding in the <page-encoding> element in the <jsp-property-group> element of web.xml.
  - Specify the character encoding in the pageEncoding attribute of the page directive.

  The strings that can be specified are the character encoding described in the canonical name for java.nio API and canonical name for java.lang API and their alias names.

  Make sure that the character encoding you want to specify matches with the character encoding used in the JSP document.

Note

For the character encoding supported by the Cosminexus XML Processor, see *1.3.2 Character codes that can be processed* in the *uCosminexus Application Server XML Processor User Guide*.

(c) In the case of the JSP document (in the case of the Web applications compliant with Servlet 2.3 specifications)

The character encoding that can be used in the JSP documents in the Web application compliant with the Servlet 2.3 specifications and the specification method of the character encoding to be used is as follows:

- Character encoding that can be used in the JSP document

  You can use the character encoding supported by the Cosminexus XML Processor. For on character encoding supported by the Cosminexus XML Processor, see *1.3.2 Character codes that can be processed* in the *uCosminexus Application Server XML Processor User Guide*.

  However, the character encoding where the alphanumeric characters, such as UTF-16, are expressed in multiple bytes cannot be used.

- Specification method of character encoding

  You can specify the character encoding you want to use in the JSP document by choosing both or one of the methods described below:

  - Specify the character encoding in the encoding attribute of the XML declaration.
  - Specify the character encoding in the pageEncoding attribute of the page directive.

  The strings that can be specified are the character encoding described in the canonical name for java.nio API and canonical name for java.lang API and their alias names.

  Make sure that the character encoding you want to specify matches with the character encoding used in the JSP document.

(d) In the case of the tag file with standard syntax

The character encoding that can be used in the tag file with the standard syntax and the specification method of the character encoding to be used is described below:

- Character encoding that can be used in the tag file

  You can use the character encoding supported by JavaVM. For details on the character encoding supported by JavaVM, see the JDK documentation.

  However, for the character encoding where the alphanumeric characters, such as UTF-16, are expressed in multiple bytes, BOM must be added at the beginning of the tag file.

- Specification method of character encoding

  You can specify the character encoding you want to use in the tag file by choosing both or one of the methods described below:

  - Add BOM at the beginning of the tag file (in the case of the Web applications compliant with Servlet 2.5/JSP 2.1 or later specifications).
  - Specify the character encoding in the pageEncoding attribute of the tag directive (in the case of the Web applications compliant with Servlet 2.4/JSP 2.0 or later specifications).

  The strings that can be specified are the character encoding described in the canonical name for java.nio API and canonical name for `java.lang` API and their alias names.

  Make sure that the character encoding you want to specify matches with the character encoding used in the tag file.

(e) In the case of the tag file with XML syntax

The character encoding that can be used in the tag file with the XML syntax and the specification method of the character encoding to be used is described below:

- Character encoding that can be used in the tag file

  You can use the character encoding# supported by the Cosminexus XML Processor.

- Specification method of character encoding

  Specify the character encoding to be used in the tag file according to the XML1.0 specifications. The strings that can be specified are the character encoding# supported by the Cosminexus XML Processor.

  Make sure that the character encoding you want to specify matches with the character encoding used in the tag file.

Note:

For the character encoding supported by the Cosminexus XML Processor, see *1.3.2 Character codes that can be processed* in the *uCosminexus Application Server XML Processor User Guide*.

(f) Default character encoding

If the character encoding is not explicitly specified in the JSP file or tag file, the JSP is processed using the default character encoding.

Note that even in this case, make sure that the default character encoding matches with the character encoding used in the tag file. The following table lists the default character encoding for the Servlet and JSP specifications:

Table 6–24: Default character encoding

Specifications	JSP page	JSP document	Tag file with standard syntax	Tag file with XML syntax
Servlet 2.2/ JSP1.1	ISO-8859-1	ISO-8859-1	--	--
Servlet 2.3/ JSP 1.2	ISO-8859-1	ISO-8859-1	--	--
Servlet 2.4 or later/ JSP 2.0, JSP 2.1	ISO-8859-1	UTF-8	ISO-8859-1	UTF-8

Legend:

--: Not applicable

You can also set the default character encoding by using the setup functionality for default character encoding. For details, see *2.6 Functionality for setting up the default character encoding*.

(14) Notes on using the setProperty action

Do not specify an invalid value in the setProperty action in the JSP page or in the name attribute of the *jsp:setProperty* tag in the JSP document. Operation is not guaranteed if you specify an invalid value in the name attribute of the *jsp:setProperty* tag.

(15) Specifying a null character string in an attribute of a JSP tag

Do not specify a null character string in an attribute of a directive or action tag in a JSP page or in a tag that begins with "jsp:" in a JSP document.

(16) Precautions related to coding the scriptlet before and after the template text of JSP

When you want to code a scriptlet that includes a control character such as an `if` statement before and after the template text of JSP, you must enclose the scriptlet explicitly in "{}" (curly brackets).

As a result of the JSP compilation, a one-lined JSP template text does not necessarily become a one lined statement in the Java file. It is sometimes output as a multi-lined statement. Therefore, if you do not code a scriptlet that includes a control character such as an `if` statement before and after the template text of JSP, by enclosing it explicitly in the "{}" (curly brackets), unintended operations might occur.

(17) Precautions related to using the scriptlet

When you code the Java code in a JSP by using a scriptlet and you want to code the `return` statement or `throw` statement in the scriptlet, code within a block such as an `if` statement.

(18) Translation error when the EL is disabled

If the JSP includes a character string such as `${aaa`, which is incorrect as an EL, an invalid EL translation error occurs even when you set the EL to a disabled status.

The following table lists the EL starting characters for which the translation error occurs for each version of the JSP specifications.

Table 6-25: TableEL starting character for which the translation error occurs

Version of the JSP specifications	EL starting character
JSP2.1	'$', '#'
JSP2.0	'$'
JSP1.2	None

(19) Coding schemaLocation in the TLD file

For `schemaLocation` coded in the TLD file, you must specify either of the values given in the following table, depending on the version of TLD.

Version of TLD	Value to be specified in schemaLocation
`2.0`	`"http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"`
`2.1`	`"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-jsptaglibrary_2_1.xsd"`

6.2.7 Precautions related to added and changed specifications in the JSP 2.1 specifications

This subsection describes the precautions for using the specifications that were added and changed in JSP 2.1 on the application server. For details on the JSP 2.1 specifications and JSP 2.0 specifications, see the respective specifications (JSP 2.1 specifications and JSP 2.0 specifications).

(1) EL2.1

The EL of the JSP 2.1 specifications integrates the EL of the JSP 2.0 specifications and the EL of the JSF1.1 specifications.

The EL syntax and API functions are determined as the EL 2.1 specifications and variables such as the implicit object defined in the JSP 2.1 specifications and JSF1.2 specifications can be used through the API functions of EL.

The following are the addition of the functionality related to EL of JSP 2.1 specifications. The changes in the specifications are described later.

- EL in #{} format
- Adding elements to TLD
- Adding options for downward compatibility

(a) EL in #{} format

In addition to the EL in ${} format that is a JSP 2.0 specification, you can describe EL in #{} format that is an EL of JSF1.1 specification as the functionality of JSP 2.1 specification.

Timing for evaluating EL in #{} format

- EL in #{} format is not evaluated during JSP output.
- `javax.el.ValueExpression` or `javax.el.MethodExpression` that are the EL objects are passed to the tag handler by the Web Container instead of the evaluation result of EL. The methods of the passed object are evaluated at any time depending on the implementation of the tag handler.

(b) Adding elements to TLD

In the JSP 2.1 specifications, the following elements were added in the `<attribute>` element of the TLD file in order to show whether it is a tag attribute that expects the #{} format:

- `<deferred-value>` element
- `<type>` element in the `<deferred-value>` element
- `<deferred-method>` element
- `<method-signature>` element in the `<deferred-method>` element

Also, the attributes corresponding to the TLD elements specified here were added to the attribute directive of the tag file.

(c) Adding options for downward compatibility

In the JSP 2.1 specifications, if EL in #{} format is described in the following locations along with the addition of the #{} format in the EL2.1 specifications, a translation error occurs:

- Template text of the JSP file or tag file
- Attribute value of the custom tag that is not a delay estimation

On the application server, if `true` is specified for the following elements or attribute values, the translation error will not occur even if EL of the `#{}` format exists in the template text or in the attribute value of the custom tag that does not have delayed evaluation. `#{}` is output as it is in a string.

- `<deferred-syntax-allowed-as-literal>` element in the `<web-app><jsp-config><jsp-property-group>` element of `web.xml`

- `deferredSyntaxAllowedAsLiteral` attribute of the page and tag directive

The following table describes whether a translation error occurs for the `#{}` EL when the settings in `web.xml` are combined with the settings in page and tag directive:

Table 6-26: Presence or absence of translation errors when the settings in web.xml are combined with the settings in page and tag directive

<deferred-syntax-allowed-as-literal> element of web.xml	deferredSyntaxAllowedAsLiteral attribute of the page and tag directive		
	true	false	Not specified
true	Y	--	Y
false	Y	--	--
Not specified	Y	--	--

Legend:

Y: Translation error does not occur and #{} EL is output as it is

--: Translation error occurs

Note:

The settings in the page and tag directive are given higher priority than the settings in `web.xml`.

(2) Functionality for deleting unwanted white spaces

In the JSP 2.1 specifications, functionality was added to delete the unwanted white spaces included in the JSP template text. In the application server, this functionality is supported according to the JSP 2.1 specifications.

If you specify true in the following element or attribute values, the template text with only white spaces is deleted when JSP is output.

- `<trim-directive-whitespaces>` element in the `<web-app><jsp-config><jsp-property-group>` element of `web.xml`
- `trimDirectiveWhitespaces` attribute of the page and tag directive

However, the white spaces in continuation with the template text that is not a white space are not deleted.

Table 6-27: Enabling or disabling of the functionality for deleting white spaces when the settings in web.xml are combined with the settings in page and tag directive

<trim-directive-whitespaces> element of web.xml	trimDirectiveWhitespaces attribute of page and tag directive		
	true	false	Not specified
true	Y	--	Y
false	Y	--	--
Not specified	Y	--	--

Legend

Y: Functionality for deleting the white spaces is enabled

--: Functionality for deleting the white spaces is disabled

Note:

The settings in the page and tag directive are given higher priority than the settings in `web.xml`.

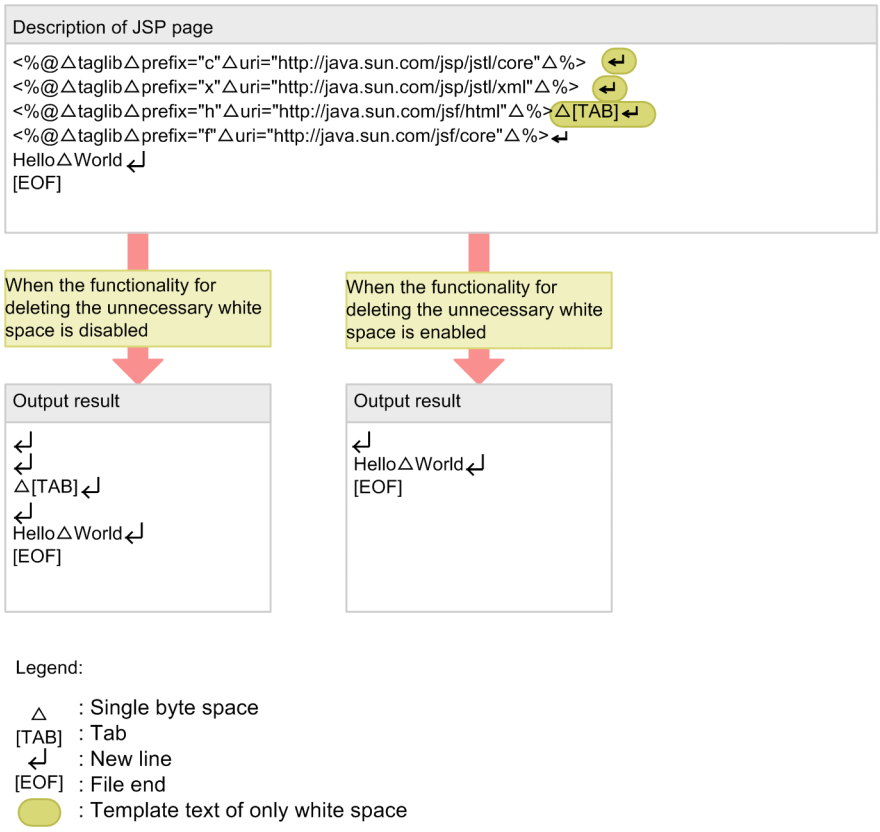
Note that if the `trimDirectiveWhitespaces` attribute is set in the page directive of the JSP document or the tag directive of the XML syntax, the attribute is processed as follows depending on the setup value.

- If the setup value is true or false, the settings of the `trimDirectiveWhitespaces` attribute are ignored and normal processing is performed.
- If an invalid value other than true or false is specified as the setup value, a translation error occurs.

Example of deleting white spaces

The following figure shows an example wherein a template text with only white spaces exists between the JSP elements:

Figure 6-8: Example wherein template text with only white spaces exists



The `taglib` directive from the first to fourth line is not output, and therefore is ignored. The new line from the first new line, second new line, and one-byte space in the third line forms a template text with white spaces only.

The fourth new line in the JSP page is not deleted since this is the white space continuing after the template text that is not a white space. Therefore, if the functionality for deleting white spaces is enabled, the text is output from the fourth new line up to [EOF].

(3) Functionality for setting a unique identifier in the tag handler

In the JSP 2.1 specifications, a functionality is added to set a unique identifier for each translation (file included in the JSP and include directive) in the tag handler. You can use this functionality by implementing the `JspIdConsumer` interface in the tag handler. The package name of the `JspIdConsumer` interface is `javax.servlet.jsp.tagext`.

During the JSP execution, the Web Container uses the `setJspId` method of the interface and sets the identifier for the tag handler that implements this interface. The identifier is determined during the JSP compilation, and therefore, the identifier might be changed in the processing of each request.

Use the string id<N> as the unique identifier. <N> indicates an integer value. <N> is allocated in the range of 0 to 2,147,483,647.

(4) Adding API functions for JSP

In the JSP 2.1 specifications, the following classes and methods are added.

- **Class added in the JSP 2.1 specifications**

Package name	Class name
javax.servlet.jsp	JspApplicationContext

- **Class that adds constructors in the JSP 2.1 specifications**

Package name	Class name
javax.servlet.jsp.tagext	TagAttributeInfo

- **Methods added in the JSP 2.1 specifications**

Package name	Class name	Method name
javax.servlet.jsp	JspFactory	getJspApplicationContext
javax.servlet.jsp.tagext	TagAttributeInfo	getDescription
		isDeferredValue
		isDeferredMethod
		getExpectedTypeName
		getMethodSignature
	TagLibraryInfo	getTagLibraryInfos

(5) Using annotations

The application server supports annotations defined in the JSP 2.1 specifications. For using annotations, see *12. Using Annotations* in the *uCosminexus Application Server Common Container Functionality Guide*.

(6) Method of determining the tag file version

The elements that can be described in the tag file were added in the JSP 2.1 specifications, so clarification of the tag file version is necessary. Therefore, in the application server, the tag file version is determined according to the JSP 2.1 specifications.

However, the tag file version must match in each tag file. The operations do not function properly if TLD files of different versions are used to execute the same tag file.

Note that even if versions that form the version determining elements of TLD file and `implicit.tld` are different, if the versions determined by the actual usage method match in each file, the operations are performed without a problem.

For example, as described in the following table, even if the versions of the TLD file and `implicit.tld` are different, the operations are performed without a problem when the tag file is used by always using TLD or when directory is always specified and used from JSP:

Table 6-28: Example when the versions of the version determining elements are different

Version determining elements	Versions
TLD file that references a tag file	2.1
implicit.tld of the directory that deploys the tag file	2.0

(7) Elements that can be specified in the tag attributes

In the JSP 2.1 specifications, if `rtexprvalue` is false in the tag attribute settings, you cannot specify the expression of scripting elements in the tag attributes even if deferredValue or deferredMethod is true.

In the application server, even if the `rtexprvalue` is false in the tag attribute settings, you can specify the expression of scripting elements in the tag attributes if deferredValue or deferredMethod is true. The following table describes the compliance between the tag attribute settings in the application server and the specifiable elements:

Table 6-29: Compliance between the tag attribute settings and the specifiable elements

Tag attribute settings		Specifiable elements			
rtexprvalue	deferredValue or deferredMethod	Strings	Expression (<%= %>)	EL(${})	EL(#{})
false	false	Y	--	--	--
true	false	Y	Y	Y	--
false	true	Y	Y	--	Y
true	true	Y	Y	Y	Y

Legend:

Y: Can be specified

--: Cannot be specified

(8) Referencing an error page

The operations when the same JSP page is specified in the `errorPage` attribute of the page directive as the transition destination when an error occurs in the JSP page are as follows:

Table 6-30: Operations when the same JSP page is specified in the errorPage attribute of the page directive as the transition destination when an error occurs in the JSP page

Version of the Servlet specifications/ JSP specifications	Contents
Servlet 2.5/JSP 2.1	As in the case of the JSP 2.1 specifications, a translation error occurs#.
Servlet 2.4/JSP 2.0	JSP is accessed. If an exception occurs, action is not taken for the exception and if the exception occurs infinitely, a stack overflow error may occur.

#

A translation error does not occur in cases other than when the transition destination in the case of an error is specified with only the file name and when the relative path from the context root is specified.

(a) Example of specification of transition destination when a translation error occurs

An example of specification of transition destination when an error occurs in the JSP page forming the translation error in the following file configuration is as follows:

```
├─jsp
│  └ example.jsp
└─WEB-INF
   └web.xml
```

If you specify the same JSP page (`example.jsp`) in the transition destination used when an error occurs in the JSP page `example.jsp` with one of the following methods, a translation error occurs.

- JSP page is specified with only the file name
  Example of specification: <%@ page isErrorPage="true" errorPage="example.jsp" %>
- JSP page is specified with the relative path from the context root
  Example of specification: <%@ page isErrorPage="true" errorPage="/jsp/example.jsp" %>

(b) Error that occurs when the transition destination in the case of mutual error is specified between different pages

The following JSP pages are assumed to exist:

- JSP page A: The transition destination used when an error occurs is set in the JSP page B.

- JSP page B: The transition destination used when an error occurs is set in the JSP page A.

If an error occurs in the JSP page A and an error also occurs in the JSP page B, an exception is thrown and the page moves to JSP page A. In the structure described here, if an exception occurs infinitely, the stack overflow error occurs.

(9) Handling of a tag file with the same name in non-extensions

The tag files have two types of extensions - .tag and .tagx. Do not deploy tag files with the same names and only different extensions such as example.tag and example.tagx.

In the JSP2.1 specifications, if tag files with the same names and only different extensions are deployed, the tag library is disabled.

In the application server, even if tag files with the same names and only different extensions are deployed, the tag library is not disabled. However, the tag file uses the files searched from the file system. In the case of tag files with the same names and only different extensions, note that the files searched previously are used and the file search order does not function properly.

(10) Changing the API specifications

In the application server, the API specifications are changed according to the API specification changes in the JSP 2.1 specifications. For details on the changes in the `javax.servlet.jsp.JspException` operations, see *6.2.13 (2) Invocation of initCause(Throwable) for objects generated in javax.servlet.ServletException and javax.servlet.jsp.JspException*.

If non-typesafe methods compliant with versions prior to JSP 2.0 are used for API functions with generic names used in the Web application classes, you can use the classes as they are even in 08-00 and later versions without any changes in the method operations. Note that a warning message (unchecked warning) will occur in the `javac` command during compilation, but the warning message does not affect operations. By applying annotation@SuppressWarnings("unchecked") in the places where the warning message occurs, the warning message will not occur.

(11) Method of setting up the character encoding for the JSP page and tag file

As per the JSP 2.1 specifications, settings are added for the addition of BOM in the setup method of the character encoding for the JSP page and standard syntax tag files in the application server.

The JSP pages are controlled as per the JSP specifications to which the Web application conforms. The tag files are controlled as per the JSP specifications corresponding to the tag file version. The following table describes the method of distinction between the character encoding for the JSP pages and tag files.

Table 6-31: Method of distinction between the character encoding for the JSP pages and tag files

Version of JSP specifications	Contents
JSP 2.1	The character encoding is set as per the JSP 2.1 specifications. However, UTF-32 BOM is not supported.
JSP 2.0	The settings for character encoding cannot be specified in the addition of BOM. If necessary, you can specify the character encoding in the `<page-encoding>` element in the `<jsp-property-group>` element of `web.xml` or in the `pageEncoding` attribute in the page and tag directive.

For details on the character encoding in JSP files and tag files, see *6.2.6 (13) Character encoding supported in JSP*.

In the JSP pages and tag files of the Web applications compliant with the Servlet 2.5 or later specifications, you can use one of the following methods to specify whether or not to set the character encoding in the addition of BOM:

- **Method of specification using the Easy Setup definition file**

  In the value of the parameter `webserver.jsp.jsp_page.bom.enabled` in the `<configuration>` tag of the logical J2EE server (j2ee-server), specify true to enable and false to disable the character encoding in the addition of BOM.
- **Method of specification using the cjjspc command**

In the `cjjspc` command, specify the `-jsppagedisablebom` option that enables the character encoding in the addition of BOM and implement JSP pre-compilation.

In the JSP pages and tag files of the Web applications compliant with the Servlet 2.5 or later specifications, if the character encoding is not set in the addition of BOM, the character encoding for JSP pages is set as per the Servlet 2.4 specifications.

(12) Mapping of the TLD file and URI

The mapping of the TLD file and URI is controlled as per the JSP specifications to which the Web application conforms. The following table lists the method of mapping the TLD file with URI for each version of the JSP specifications.

Table 6-32: Method of mapping the TLD file and URI

Version	Mapping methods
Servlet 2.5 (JSP 2.1)	JSTL and JSF URI are automatically mapped as per the JSP 2.1 specifications. The URI to be mapped are as follows: • http://java.sun.com/jsp/jstl/core • http://java.sun.com/jsp/jstl/xml • http://java.sun.com/jsp/jstl/fmt • http://java.sun.com/jsp/jstl/sql • http://java.sun.com/jsp/jstl/functions • http://java.sun.com/jstl/core • http://java.sun.com/jstl/xml • http://java.sun.com/jstl/fmt • http://java.sun.com/jstl/sql • http://java.sun.com/jstl/core_rt • http://java.sun.com/jstl/xml_rt • http://java.sun.com/jstl/fmt_rt • http://java.sun.com/jstl/sql_rt • http://java.sun.com/jsf/core • http://java.sun.com/jsf/html
Servlet 2.4 (JSP 2.0)	The TLD file and URI are not mapped automatically. If JSTL and JSF are used, deploy the TLD file with the JSTL and JSF specifications as in the case of normal TLD files.

In Web applications compliant with the Servlet 2.5 or later specifications, automatic mapping is processed at top priority, so you cannot overwrite the mapping of the TLD file and URI.

Therefore, in the application server, you can use one of the following methods to specify whether to map the TLD file and URI automatically:

- **Method of specification using the Easy Setup definition file**
  In the value of the parameter `webserver.jsp.tld.mapping.java_ee_tag_library.enabled` in the `<configuration>` tag of the logical J2EE server (j2ee-server), specify true to enable and false to disable automatic mapping.
- **Method of specification using the cjjspc command**
  In the `cjjspc` command, specify the `-nojavaeetaglib` option that disables automatic mapping and implement JSP pre-compilation.

If you want to use multiple versions of tag libraries for each Web application, disable automatic mapping using these methods. If you disable automatic mapping, you can mix libraries of multiple versions by deploying the libraries in respective Web applications.

(13) EL escape sequence

In the JSP 2.1 specifications, "#{" is added to the string indicating the start of EL.

The JSP pages and JSP documents are controlled as per the JSP specifications to which the Web applications conform. The tag files are controlled as per the JSP specifications corresponding to the tag file version. Also, the control differs depending on whether the EL settings are enabled.

The following table describes the controlling of the escape sequence that expresses "#" as a string for each version of the servlet and JSP specifications.

Table 6-33: Controlling of the escape sequence expressing # as a string

Versions of Servlet specifications/ JSP specifications	Specifications			
	When EL settings are enabled		When EL settings are disabled	
	\$ output	\# output	\$ output	\# output
Servlet 2.5/JSP 2.1	$	#	$	#
Servlet 2.4/JSP 2.0	$	\#	$	\#

In the Web applications compliant with Servlet 2.5 specifications, "\#" is output as # by the escape sequence regardless of the conditions as in the case of "\$". Therefore, if you want to output "\#", you must describe "\\#".

For details on the escape sequence specifications of "\$"in the JSP 2.0 specifications, see *6.2.8 (15) Escape sequence of EL (Expression Language)*.

(14) Changing the processing for Enum type in EL

The processing corresponding to the Enum type object defined in the Java SE 5 specifications was added from the EL of JSP 2.1 specifications.

The JSP pages and JSP documents are controlled as per the JSP specifications to which the Web applications conform. The tag files are controlled as per the JSP specifications corresponding to the tag file version.

The following table describes the processing for the Enum type in EL for each version of the servlet and JSP specifications in the application server.

Table 6-34: Processing for the Enum type in EL

Versions of Servlet specifications/ JSP specifications	Contents
Servlet 2.5/JSP 2.1	Processed as per the JSP 2.1 specifications.
Servlet 2.4/JSP 2.0	In spite of Enum type, the processing is performed like other objects as per the JSP 2.0 specifications.

However, if you use API functions of EL defined in the JSP 2.0 specifications, but deprecated in the JSP 2.1 specifications, the API functions are processed in the range of EL functionality in JSP 2.0 specifications, regardless of the Web application version.

(15) Changing the processing of <, >, lt, and gt operators in EL

The JSP pages and JSP documents are controlled as per the JSP specifications to which the Web applications conform. The tag files are controlled as per the JSP specifications corresponding to the tag file version.

The following table describes the processing of <, >, lt, and gt operators in EL in the application server for each version of the servlet and JSP specifications.

Table 6-35: Processing of <, >, lt, and gt operators in EL

Versions of Servlet specifications/ JSP specifications	Contents
Servlet 2.5/JSP 2.1	The <, >, lt, and gt operators are processed as per the JSP 2.1 specifications.
Servlet 2.4/JSP 2.0	The <, >, lt, and gt operators are processed as per the JSP 2.0 specifications.

However, if you use API functions of EL defined in the JSP 2.0 specifications, but deprecated in the JSP 2.1 specifications, the API functions are processed in the range of EL functionality in JSP 2.0 specifications, regardless of the Web application version.

(16) Changing the API functions of EL

This point describes the API functions of EL for JSP specifications in the application server.

Corresponds to the API functions added in the JSP 2.1 specifications. When you use the EL functionality added in the JSP 2.1 specifications, use the API function of the javax.el package.

When you use the API functions of EL defined in the JSP 2.0 specifications, EL is evaluated as per the EL specifications defined in the JSP 2.0 specifications.

6.2.8 Precautions related to added and changed specifications in the JSP 2.0 specifications

This subsection describes the precautions for using the specifications that were added and changed in JSP 2.0 on the application server. For details on the JSP 2.0 specifications and JSP 1.2 specifications, see the respective specifications (JSP 2.0 specifications and JSP 1.2 specifications).

(1) Default extension of JSP documents

In the JSP 2.0 specifications, the standard extension of JSP documents is jspx. In the Web Container used on the application server, a file with extension jspx is handled as a JSP document even if URL mapping is not defined in `web.xml` using default mapping.

(2) Output destination of the Java source files and class files of the tag file

Like a JSP file, Java source files and class files are generated by JSP compilation for a tag file. The Java source files and class files are output in the directory to output the JSP compilation result.

You can change the directory to output the JSP compilation result. Note that you must change the output destination directory when the path of the generated Java source files and class files exceeds the upper limit for the OS.

For details on the directory to output the JSP compilation result, see *2.5.5(2) Output destination of JSP compilation results* when the JSP pre-compilation functionality is used and see *2.5.6(3) Output destination of JSP compilation results* when the JSP pre-compilation functionality is not used.

(3) Multiple assignment of evaluation API functions for JSP EL expression

In the JSP 2.0 specifications, the following API functions are provided as the API functions for performing the syntax analysis and evaluating the EL expression:

- `evaluate` method of the `javax.servlet.jsp.el.ExpressionEvaluator` class
- `evaluate` method of the `javax.servlet.jsp.el.Expression` class

In the JSP 2.0 specifications, you cannot specify multiple EL expressions from these API functions, but in the application server, you can specify multiple EL expressions.

(4) JSP files and tag files coded in the XML syntax

- **Character encoding**

  If the character encoding of the JSP document is specified in Web application version 2.4, specify the character encoding in the XML declaration.

  In the case of the JSP 1.2 specifications, the character encoding of the file was specified in the pageEncoding attribute of the page directive or the charset value of the contentType attribute, but from the JSP 2.0 specifications, changes were made such that the character encoding is specified in the XML declaration.

- **Prefix of the standard action**

  The JSP standard action is identified by http://java.sun.com/JSP/Page of the XML name space. Consequently, standard action must be specified in the prefix of the XML name space. An example of description wherein the prefix is assumed as jsp is described below.

```
<?xml version="1.0" ?>
<jsp:root
 xmlns:jsp=http://java.sun.com/JSP/Page
 version="2.0">
 <jsp:directive.page import="java.util.* "/>
 <jsp:useBean id="name" class="test.Bean"/>

</jsp:root>
```

- **Handling the <jsp:root> element**

  From the JSP 2.0 specifications, even if the <jsp:root> element is not specified in the root element, the <jsp:root> element is handled as a JSP file or tag file of the XML syntax.

  In the JSP 1.2 specifications, <jsp:root> had to be specified in the root element as a condition for the JSP document, but from JSP 2.0 specifications, changes were made so that even if <jsp:root> is not specified, when the value of <jsp-config><jsp-property-group><is-xml> defined in web.xml is true or when the extension is jspx and tagx, <jsp:root> is handled as JSP with XML syntax format.

(5) Deprecated isThreadSafe attribute of the page directive

The isThreadSafe attribute of the page directive is deprecated in the JSP 2.0 specifications since the javax.servlet.SingleThreadModel interface is deprecated.

In the application server, you can use the isThreadSafe attribute of the page directive regardless of the Web application version. However, in the Servlet 2.4 specifications, note the cause due to which the javax.servlet.SingleThreadModel interface is deprecated, and then use.

(6) Default ContentType value of HTTP response in JSP documents

In JSP 2.0 specifications, a postscript has been added that the when the JSP document is used, the value of the default ContentType is text/xml.

In the application server, text/xml is operated as the default value in JSP 2.0 and later versions and text/html is operated as the default value in JSP 1.2.

(7) Deploying the tag library descriptor (TLD file)

In the JSP 2.0 specifications, the provisions regarding the deployment location of the tag library descriptor is added.

In the application server, the KDJE39289-W message may be output when the Web application starts and during JSP compilation depending on the directory deployed. However, the Web application is executed without an error.

The message output conditions are described below:

Deployed directory

- Not under /WEB-INF directory
- Under /WEB-INF/classes directory
- Under /WEB-INF/lib directory

The time when the message is output

- When the applicable tag library descriptor is specified in the `<taglib><taglib-location>` tag of `web.xml` and the Web application is started
- When compiling JSP that directly specifies and uses the applicable tag library descriptor in the tag library declaration

(8) XML view information that can be acquired with the getInputStream method of the javax.servlet.jsp.tagext.PageData class

The specifications of the XML view information that can be acquired with the `getInputStream` method of the `javax.servlet.jsp.tagext.PageData` object were changed in the JSP 2.0 specifications. The `getInputStream` method is specified and used in the third argument of the validate method of the `javax.servlet.jsp.tagext.TagLibraryValidator` class.

The changes in the application server are described as follows for the JSP 2.0 and later versions and JSP 1.2:

(a) jsp:id attribute

JSP 2.0 and later versions

The jsp:id attribute is added.

JSP 1.2

The jsp:id attribute is not added.

(b) Character encoding of XML view

JSP 2.0 and later versions

The character encoding of XML view is always considered as UTF-8, the character code is considered as UTF-8, and the XML declaration is output.

JSP 1.2

The character encoding of XML view is always considered as UTF-8, the character code is considered as UTF-8, and the XML declaration is not output.

(c) pageEncoding attribute of the page directive

JSP 2.0 and later versions

The value of the pageEncoding attribute is set in UTF-8. If the pageEncoding attribute does not exist, the pageEncoding attribute is added.

JSP 1.2

The value of the pageEncoding attribute is not changed.

(d) contentType attribute of the page directive

JSP 2.0 and later versions

Set the value set in the `setContentType` method of the `ServletResponse` class in the value of the contentType attribute. If the contentType attribute does not exist, the contentType attribute is added.

JSP 1.2

The value of the contentType attribute is not changed.

(9) Default character coding of files included in the include directive

In the JSP 2.0 specifications, a postscript has been added that the pageEncoding attribute of the page directive is only applied to the file that describes the pageEncoding attribute.

In the application server, regardless of the Web application version, if the character code is not specified in the include destination file while including the file in the include directive, the character code at the include source is applied to the include destination file.

(10) Conflicting character codes in the JSP documents

The specifications when the character code specified for the XML declaration in the JSP document differs from the character code specified in the pageEncoding attribute of the page directive in the JSP document, have been added in the JSP 2.0 specifications. This description is not present in the JSP 1.2 specifications.

The control when the character codes in the application server are different will be separately described for JSP 2.0 and later versions and JSP 1.2.

JSP 2.0 and later versions

A translation error occurs.

JSP 1.2

The pageEncoding attribute of the page directive is used.

(11) Default value of HTTP response character code in the JSP documents

The default character code of HTTP response used when the contentType attribute of the page directive does not exist in the JSP document and when the CHARSET value does not exist in the attribute has been added in the JSP 2.0 specifications.

The default value in the application server will be separately described for JSP 2.0 and later versions and JSP 1.2.

JSP 2.0 and later versions

UTF-8 is used.

JSP 1.2

ISO-8859-1 is used.

(12) Specification of multiple pageEncoding attributes of the page directive

The specifications for multiple pageEncoding attributes of the page directive are changed in the JSP 2.0 specifications.

In JSP 2.0 specifications, multiple pageEncoding attributes can be specified for each translation (files included in the JSP and include directive). A specification was also added that if multiple pageEncoding attributes are specified in the same JSP file, a compilation error occurs.

In the application server, regardless of the Web application version, you can specify multiple pageEncoding attributes in each translation. In this case, the value specified in each file is applied to the applicable file. Also, the specification of multiple pageEncoding attributes in the same JSP file differs in the JSP 2.0 and later versions and in JSP 1.2. The specifications in the application server will be described separately for JSP 2.0 and later versions and for JSP 1.2.

JSP 2.0 and later versions

The pageEncoding attribute can be specified only once in one file. If multiple pageEncoding attributes are specified, a compilation error occurs.

JSP 1.2

Multiple pageEncoding attributes can be specified in one file. The value described first is applied.

(13) When uri that is not registered in the taglib map is described in the tag library declaration of the JSP document

Specifications were added in the JSP 2.0 specifications for the operations when the tag library is declared in the JSP document using the name space and the specified uri is not found in the taglib map (mapping of the uri and tag library descriptor).

The operations in the application server will be described separately for JSP 2.0 and later versions and JSP 1.2.

JSP 2.0 and later versions

If the specified uri is not registered in the taglib map, the action defined in the uri name space is handled without being analyzed (output as text).

JSP 1.2

- When the uri is an absolute URI

  A translation error occurs.

- When the uri is not an absolute URI

  The TLD file (tag library descriptor) is searched and used as the path in the Web application. If the TLD file does not exist, a translation error occurs.

(14) Character code in JSP documents

The method for determining the file character code in the JSP document was changed in the JSP 2.0 specifications.

The method for determining the character code in the application server will be described separately for JSP 2.0 and later versions and JSP 1.2.

JSP 2.0 and later versions

The XML declaration is followed according to the XML 1.0 specifications. If the XML declaration is not present, the default value is UTF-8.

JSP 1.2

The pageEncoding attribute of the page directive is followed. If the pageEncoding attribute does not exist, the character code specified in the contentType attribute charset= is followed. If both do not exist, the default value is ISO-8859-1.

(15) Escape sequence of EL (Expression Language)

The JSP specifications and the specifications for the Web Container used in the application server are described below for the JSP 2.0 specification of escape sequence that expresses "$" included in "${", indicating the start of EL, as a string.

In the Web Container used on the application server, "\$" is output as "$" using the escape sequence. If you want to output "\$", code as "\\$".

The operations when "\$" is coded will be described separately for JSP 2.0 and JSP 1.2.

JSP 2.0

In the JSP 2.0 specifications, if the EL settings are disabled, "$" need not be considered as the starting character of EL and "\" is not handled as a control code. When operating in JSP 2.0, the output result of "\$" differs depending on whether the EL settings are enabled. The following table describes the output results of "\$" when the operations are performed in JSP 2.0.

Table 6-36: Output results of "\$" when the operations are performed in JSP 2.0

Enabling/ Disabling of EL settings	Specifications	Output results
Enabled	JSP 2.0 specifications	"$"
	Web Container used in the application server	"$"
Disabled	JSP 2.0 specifications	"\$"
	Web Container used in the application server	"$"

To disable the EL settings, use one of the following methods:

- Specify true in the isELIgnored attribute of the page directive.
- Specify true in the isELIgnored attribute of the tag directive.
- Specify true in the `<el-ignored>` tag of `web.xml`.

JSP 1.2

In the JSP 1.2 specifications, "$" is not a reserved word. "\" is not handled as a control code, therefore, "\$" is output as "\$".

In the Web Container used in the application server, "\" is handled as a control code even when operations are performed in JSP1.2, so "\$" is output as "$". However, when "\$" is used in the attribute value of the JSP document format, "\$" is output as "\$".

The following table describes the output result of "\$" when operations are performed in JSP 1.2.

Table 6-37: Output result of "\$" when operations are performed in JSP 1.2

Specifications	Output results
JSP 1.2 specifications	"\$"
Web Container used in the application server	"$"

(16) Type of EL evaluation results

The JSP specifications and application server specifications for the type of EL evaluation results specified in the custom tag attribute are described below.

JSP 2.0 specifications

The EL evaluation result is converted to the expected type of custom tag attribute.

Application server

The EL evaluation result is converted to the type of the `setter` method argument corresponding to the custom tag attribute. The type element defined in the TLD attribute is not used for type conversion.

If the description location of EL is the tag file, the EL evaluation result is converted to the type specified in the type attribute of the attribute directive.

The examples when the types of EL evaluation results differ in the JSP specifications and the application server are described below.

Example:

- Custom tag attribute name: `attr`
- Signature of the custom tag setter: `void setAttr(java.lang.String` *hoge*`)`
- Value of type element of the attr attribute in TLD: `java.lang.Integer`

In this example, the type of EL evaluation result is as follows:

JSP 2.0 specifications

The type is converted to java.lang.Integer.

Application server

The type is converted to java.lang.String.

6.2.9 Precautions for implementing JSPs of the JSP 1.2 specifications

This subsection describes the important points to be noted when implementing JSPs of the JSP 1.2 specifications.

These notes are not applicable to the JSP implementation of JSP 2.0 or later.

(1) Notes on using JSP documents

The notes on using JSP documents are as follows:

- In a JSP document, specify a valid value in the `xmlns:jsp` attribute of the *jsp:root* tag as described in the JSP 1.2 specifications. If you specify an invalid value outside the scope described in the JSP 1.2 specifications, that value is ignored.
- You must specify 1.2 in the version attribute of the *jsp:root* tag in the JSP document.
- In a JSP document, you must specify only the valid values for tag attributes as described in the JSP 1.2 specifications. Any invalid value specified in tag attributes is ignored.
- In a JSP document, you must specify all the tag attributes that are specified as mandatory in the JSP 1.2 specifications. If you omit any of the mandatory attributes, the operation is not guaranteed.
- In JSP documents, you must not code the tags as child tags if they are not allowed to be described as `child` tags. If you inappropriately code tags as `child` tags, operation is not guaranteed.

- A JSP document with the UTF-8 character code cannot be used if BOM is added to the document. If you want to use the UTF-8 character code for a JSP document, you must not add the BOM. A JSP document with the UTF-8 character code and BOM added to it results in a compilation error.

(2) Notes on using the pageEncoding attribute of the page directive

You need to only code the correct value for the `pageEncoding` attribute in the page directive of a JSP page or `<jsp:directive.page>` tag of a JSP document once

However, no error occurs even if you code the `pageEncoding` attribute in the page directive of a JSP page or *jsp:directive.page* tag of a JSP document more than once.

(3) Notes on using standard actions

You cannot code a tag that starts with "jsp:" and that is not a tag included in the JSP 1.2 specifications.

(4) Notes on using custom tags

You must not define duplicate attributes in a tag. The operation is not guaranteed if you define duplicate attributes within a tag in a JSP page.

(5) Notes on using a tag library

If you specify `webserver.xml.validate=false` for a property, the tag library descriptor (TLD file) is not validated. Although no error might be output and the operation is continued even if you use a TLD file that does not comply with the specifications (XML schema definitions), the operation is not guaranteed.

You must properly follow the specifications (XML schema definitions) when coding the TLD file.

(6) Notes on using the plugin action

You must specify only a value specified in the JSP 1.2 specifications for a plugin action in a JSP page, or for the type attribute of the *jsp:plugin* tag of a JSP document. If you specify a value that is not specified in the JSP 1.2 specifications, the output value of the type attribute will be NULL.

(7) Notes on using the params action

As coding the *jsp:param* tag is mandatory according to the specifications, if you code the params action or the *jsp:params* tag, you must also code the *jsp:param* tag.

However, if you code the params action in the JSP page of a JSP document, or if you code the *jsp:params* tag in a JSP document, no error occurs even if you do not code the *jsp:param* tag element.

6.2.10 Precautions related to the specifications that are added or changed in the EL2.2 specifications

This subsection describes the points to be noted when using added or changed EL2.2 specifications with Application Server. For details on the EL2.2 specifications, see *EL2.2 Specifications*.

- Although Application Server 09-00 supports EL2.2, both EL2.1 and EL2.2 implementations are available. You can switch the implementation by specifying the following parameter in the Easy Setup definition file. You must specify the parameter in the *configuration* tag of the logical J2EE server in the Easy Setup definition file.

  `webserver.jsp.el2_2.enabled`

  For details on the `webserver.jsp.el2_2.enabled` parameter, see *4.6 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.
- If multiple methods with the same number of parameters are specified in a single Bean, the type conversion is done based on the method that is defined first. Accordingly, if the type of the argument when calling a method matches a method that is defined first, the method is called without any problem; otherwise, if the argument type does not match, ELException is thrown.

6.2.11 Points to remember when upgrading the version of an existing Web application to the Servlet 3.0 specifications

This subsection describes tasks you need to perform and precautions you need to take when upgrading the version of a Web application from Servlet 2.5 to Servlet 3.0 specifications. For details on the Servlet 3.0 specifications, see *Servlet 3.0 Specifications*.

(1) Specifications added or changed in the Servlet 3.0 and JSP 2.2 specifications

For details on the precautions to be taken when using the additions or changes made in the Servlet 3.0 specifications, see *6.2.3 Precautions related to the specifications that are added or changed in Servlet 3.0 specifications*.

The specifications added based on the JSP 2.2 specifications are not supported. However, as the web.xml schema described in JSP 2.2 is supported, even if you add the JSP 2.2 tags in `web.xml`, the tags are ignored but no error is reported.

(2) Migrating web.xml

Modify the definitions in `web.xml` according to the Servlet 3.0 specifications. For details, see *Java Servlet Specification v3.0.* For specification related changes, see *6.2.3 Precautions related to the specifications that are added or changed in Servlet 3.0 specifications*.

6.2.12 Points to remember when upgrading the version of an existing Web application to the Servlet 2.5 specifications

This subsection describes the operations and precautions required for upgrading the version of Web applications corresponding to the Servlet 2.4 specifications to the Servlet 2.5 specifications. For details on the Servlet 2.5 specifications, see the Servlet 2.5 specifications.

(1) Specifications added and changed in Servlet 2.5 specifications and JSP 2.1 specifications

For details on the Servlet 2.5 specifications and precautions related to added and changed specifications in the JSP 2.1, see *6.2.4 Precautions related to added and changed specifications in the Servlet 2.5 specifications* or *6.2.7 Precautions related to added and changed specifications in the JSP 2.1 specifications* respectively.

(2) Migration of web.xml

`web.xml` is modified to the definition specified in the Servlet 2.5 specifications. For details, see Java Servlet Specification v2.5. Furthermore, for details on changing the specifications, see *6.2.4 Precautions related to added and changed specifications in the Servlet 2.5 specifications*.

(3) Size of classes generated from JSP

The functionality of JSP 2.1 specifications is applied to the classes generated from the JSP included in the Web applications compliant with Servlet 2.5 specifications. Therefore, the contents of the Java source files and class files generated from JSP are changed.

In this case, note that the number of lines in the Java source file, the size of the methods included in the classes generated from JSP, or the usage of the Permanent area may increase.

6.2.13 Precautions related to Web applications when migrating from a previous version of Application Server to 09-00

This subsection describes the precautions related to Web applications when migrating from a previous version of Application Server to 09-00.

(1) Version information acquiring API supported by the Web Container

In the Servlet API listed in the following table, you can acquire the version information of the Servlet specifications supported by the Web Container.

Table 6-38: Version information acquiring API supported by the Web Container

Interface/Class name	Method name
`javax.servlet.ServletContext`	`getMajorVersion`
	`getMinorVersion`
`javax.servlet.jsp.JspEngineInfo`	`getSpecificationVersion`

Even if the Web application version is earlier than Servlet 2.5/JSP 2.0, the version of the servlet specifications supported by the Web container is Servlet 3.0/JSP 2.1. Accordingly, the servlet APIs described above return Servlet 3.0/JSP 2.1 as the servlet version. Note that the APIs retrieve information irrespective of the operating mode of the server.

(2) Invocation of initCause(Throwable) for objects generated in javax.servlet.ServletException and javax.servlet.jsp.JspException

You cannot invoke `initCause(Throwable)` for the following objects:

- `ServletException` object generated in the constructors `ServletException(String, Throwable)` and `ServletException(Throwable)` of `javax.servlet.ServletException`
- `JspException` object generated in the constructors `JspException(String, Throwable)` and `JspException(Throwable)` of `javax.servlet.jsp.JspException`

If you want to invoke `initCause(Throwable)`, set up the compatibility properties in the Easy Setup definition file. For details on the settings, see *6.2.1 (22) Acquiring the root cause exception specified in the constructor of the javax.servlet.ServletException class*.

Operations when the setCharacterEncoding method of the javax.servlet.ServletRequest interface is invoked

After invoking the `getReader` method of the `javax.servlet.ServletRequest` interface, if you invoke the `setCharacterEncoding` method of the `javax.servlet.ServletRequest` interface, the return value of the `getCharacterEncoding` method is not changed. For details, see *6.2.4 (8) Disabling the invocation of the setCharacterEncoding method of the javax.servlet.ServletRequest interface*.

(3) Operations when the ServletConfig object maintained by the javax.servlet.GenericServlet class is null

If ServletConfig maintained by the `javax.servlet.GenericServlet` class as the instance variable is null, the exceptions thrown by the following methods differ in each version:

- `getInitParameter(String)`
- `getInitParameterNames()`
- `getServletContext()`
- `getServletName()`

The exceptions thrown by these methods will be described below for each version.

Versions prior to 07-60

`java.lang.NullPointerException`

08-00 or later

`java.lang.IllegalStateException`

'ServletConfig has not been initialized.' is output as the exception message.

These exceptions are thrown when both the following conditions are fulfilled:

- When the servlet that inherited javax.servlet.GenericServlet overrides init(ServletConfig)
- When super.init(ServletConfig) is not invoked in the overridden init(ServletConfig) or when super.init(ServletConfig) is invoked by specifying null in the argument

(4) Log output by the init method and destroy method of the javax.servlet.GenericServlet class

If you initialize or terminate the servlet that does not override the `init` method and `destroy` method of the `javax.servlet.GenericServlet` class, the log is output in the servlet log. For details on the log that is output, see *5.1.1(17) Messages output when the init method and destroy method are not overridden*.

However, the operations when the ServletConfig that has the `javax.servlet.GenericServlet` class in the instance variable is null differ with the versions prior to 07-60 and with 08-00. The differences in operations in each version are described below.

Versions prior to 07-60

Log output fails and the `java.lang.NullPointerException` exception is thrown.

08-00 or later

An exception is not thrown. The log is also not output.

(5) Log output by the init method and destroy method of the javax.servlet.GenericServlet class

If you initialize or terminate the servlet that does not override the `init` method and `destroy` method of the `javax.servlet.GenericServlet` class, the log is output in the servlet log. For details on the log that is output, see *6.2.1 (17) Messages output when the init method and destroy method are not overridden*.

However, the operations when the ServletConfig that has the `javax.servlet.GenericServlet` class in the instance variable is null differ with the versions prior to 07-60 and with 08-00or later. The differences in operations in each version are described below.

Versions prior to 07-60

Log output fails and the `java.lang.NullPointerException` exception is thrown.

08-00or later

An exception is not thrown. The log is also not output.

(6) Searching the TLD file in the library JAR

In 08-00or later, you can search the TLD file included in the library JAR and use the tag library present in the library JAR. If the library JAR contains the TLD file, you can use the TLD file that could not be used in the previous version, so the following operating changes occur:

- If the URI described in the `<uri>` element of the TLD file present in the library JAR is duplicated with the URI described in `web.xml` and another TLD file, a warning message is output. The URI and TLD file mapping described in the TLD file present in the library JAR is disabled.
- If the URI described in the TLD file present in the library JAR is specified in the uri attribute of the JSP `taglib` directive, you can perform the JSP translation. Note that with versions prior to 07-60, a translation error occurs during the JSP translation.

For details, see *2.3.4 Storing the tag library in the J2EE applications*.

(7) Searching the TLD file in the JAR file specified in the class path in the cjjspc command

In 08-00or later, during the JSP compilation using the `cjjspc` command, you can search the TLD file within the JAR file specified in the class path in the `-classpath` option and can use the tag library that exist within the JAR file. If the JAR file specified in the class path includes TLD files, you can use the TLD files that you could not use with the versions prior to the 07-60 version. As a result, the following events occur:

- If the URI described in the <uri> element of the TLD file that exists within the JAR file specified in the class path is duplicated with the URI described in web.xml and another TLD file, a warning message is output. The URI and TLD file mapping described in the TLD file that exists within the JAR file specified in the class path is disabled.
- If the URI described in the TLD file that exists within the JAR file specified in the class path is specified in the uri attribute of the JSP taglib directive, you can perform the JSP translation. Note that with versions prior to the 07-60 version, a translation error occurs during the JSP translation.

For details, see *2.3.4 Storing the tag library in the J2EE applications*.

(8) Deleting the HTTP Cookie indicating the session ID of the HTTP session

When the HTTP session is disabled in the Web application, the operations for the HTTP Cookie information that the Web client maintains differ with the versions prior to the 07-60 version and in the 08-00 or later. The differences in operations for each version are as follows:

08-00 or later

The HTTP Cookie information that the Web client maintains is deleted. Also, the sending of the session ID for a disabled HTTP session is controlled.

Versions prior to 07-60

The HTTP Cookie information that the Web client maintains is not deleted. Therefore, even after the HTTP session is disabled, sending of the disabled session ID might continue.

For details on deleting the HTTP Cookie, see *2.7.4 Deleting invalid session IDs maintained by the Web client*.

When performing upgrade installation from the versions prior to the 07-60 version and in the 08-00 or later, the settings for the following parameter are added to the settings of the already setup J2EE server:

```
webserver.session.delete_cookie.backcompat=true
```

By adding these settings, the HTTP Cookie information is no longer deleted and the operations are same as the versions prior to the 07-60 version. For details on the parameters, see *2.4 usrconf.properties (User property file for J2EE servers)* in the *uCosminexus Application Server Definition Reference Guide*.

6.2.14 Points to remember when upgrading the version of an existing Web application to the Servlet 2.4 specifications

This subsection describes the operations and precautions required for upgrading the version of Web applications corresponding to the Servlet 2.2 specifications or Servlet 2.3 specifications to the Servlet 2.4 specifications. For details on the Servlet 2.4 specifications, see the Servlet 2.4 specifications.

(1) Migration of web.xml

The web.xml file created according to the Servlet 2.2 specifications or Servlet 2.3 specifications is modified to the definition specified in the Servlet 2.4 specifications. For the points changed in the Servlet 2.4 specifications, see *5.2.5 Notes on specifications added or changed in Servlet 2.4 or later versions (web.xml)* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

(2) Modification to code corresponding to the Servlet 2.4 specifications

In Servlet 2.4 specifications, specifications are added and changed from Servlet 2.2 and Servlet 2.3. Check the added and changed points and modify in the code corresponding to the Servlet 2.4 specifications. For details on the points that are added and changed in the Servlet 2.4 specifications, see *6.2.5 Precautions related to added and changed specifications in the Servlet 2.4 specifications*.

Furthermore, even in the JSP 2.0 specifications, the specifications are added and changed from JSP 1.2. Check the added and changed points and modify to the code corresponding to the JSP 2.0 specifications. For details on the points that are added and changed in the JSP 2.0 specifications, see *6.2.8 Precautions related to added and changed specifications in the JSP 2.0 specifications*.

(3) Migration from J2EE server mode to servlet engine mode

The Web applications corresponding to the Servlet 2.4 specifications cannot be executed in the servlet engine mode. Consequently, if the servlet engine mode is used, the migration must be performed to the J2EE server mode.

For migrating from the servlet engine mode to the J2EE server mode, see *3. Servlet engine mode* in the *uCosminexus Application Server Compatibility Guide*.

(4) JSP syntax check

The JSP files contained in the Web applications corresponding to the Servlet 2.4 specifications conform to the JSP 2.0 specifications. In the JSP 2.0 specifications, the syntax check is performed even more strictly than in JSP 1.2 specifications. Therefore, errors that are not reported in the Web applications of the Servlet 2.3 specifications are now reported.

When you upgrade the versions of Web applications corresponding to the Servlet 2.2 specifications or Servlet 2.3 specifications to the Servlet 2.4 specifications, compile JSP using the `cjjspc` command and confirm that errors do not occur. If a compilation error is reported, make corrections according to the contents of the reported errors.

For details on the `cjjspc` command, see *2.5.2 Methods for performing JSP pre-compilation*.

6.2.15 Using annotations in servlets

In the Application Server, you can use annotations in the servlets. For annotations supported in Application Server, see *2. Annotations and Dependency Injections Supported in Application Server* in the *uCosminexus Application Server AP Reference Guide*.

6.2.16 Precautions related to size limitations for JavaVM methods

If a method exceeding 64 KB exists in JavaVM, an error occurs during the generation of the class file or the `java.lang.LinkageError` exception occurs when the class is loaded. Therefore, the byte code for one method must have a size within 64 KB.

Also, even if the size is within 64 KB, if a method with an extremely big size exists, the following adverse effects might occur:

- GC processing takes extremely long time.
- JIT compilation takes extremely long time.
- JIT compilation consumes extremely large amount of memory.

In the Web application, the byte code of one method might exceed 64 KB due to the auto-generated java source code. The following points describe the auto-generation of the java source code and the reviewing method when the method size is big.

(1) Auto-generation of the java source code

The auto-generation of the java source code is as follows:

- Auto-generation in the JSP specifications

  In the JSP specifications, the java source code is automatically generated in the `_jspService` method or the `doTag` method from the contents described in the JSP file or tag file.
- Auto-generation when the custom tag is used in the Application Server

  In the Application Server, if the custom tag is used, the processing of the custom tag and contents described in the body are changed into a method and the java source code is automatically generated.

  The custom tag can be converted into a method when the processing of the custom tag or all the custom tags included in the body fulfill the following conditions:

  - The attribute and body are scriptless.
  - The script variable is not defined.

(2) Reviewing method when the method size is big

When the number of lines of the methods for auto-generated java source code exceeds 1000 lines, including the comments and blank lines, the messages KDJE39231-W and KDJE39333-W are output.

If a message is output, review the body contents of the JSP file, tag file, or the custom tag.

The following are the reviewing methods for each location to be reviewed:

- **When the size of the JSP file contents is big**
  Divide the JSP file using dynamic include (include action).
- **When the size of the tag file contents is big**
  Implement one of the followings:
  - Divide the tag files used from the JSP file into multiple tag files.
  - Divide the tag files such that another tag file is invoked from the tag file.
- **When the size of the custom tag body is big**
  Divide the custom tag using dynamic include (include action).

6.3 Precautions for JSP migration

This section describes notes when moving JSP.

With the versions prior to 07-00 and with version 07-00 and later versions, the JSP compilation operations are different. On the Application Server, the JSP contents are checked according to the JSP specifications during the JSP translation, therefore, the errors might occur during the JSP translation and migration cannot be performed. In such a case, if you use the *JSP translation backward compatibility function*, you can set the same JSP compilation operations with the versions prior to 07-00 and with version 07-00 and later versions and ensure that the errors do not occur.

Specify the definition for the JSP translation backward compatibility function in the Easy Setup definition file. For the specified contents, see the following points. You can also define the JSP translation backward compatibility function in the options when executing the `cjjspc` command. For details on how to specify the options when executing the `cjjspc` command, see the *uCosminexus Application Server Command Reference Guide*.

The following table describes the organization of this section.

Table 6-39: Organization of this section (Precautions for JSP migration)

Title	Reference
Precautions related to the definition of script variables for the custom tag	*6.3.1*
Precautions related to the class attributes of the <jsp:useBean> tag	*6.3.2*
Precautions related to the Expression check of the tag attribute values	*6.3.3*
Precautions related to Expression specified in the tag attribute values	*6.3.4*
Precautions related to the prefix attribute of the taglib directive	*6.3.5*

6.3.1 Precautions related to the definition of script variables for the custom tag

This subsection describes the precautions when defining the script variables in the JSP custom tag, the differences in the operations of the JSP compilation depending on the usage of the JSP translation backward compatibility function, and also describes the definition of the JSP translation backward compatibility function.

(1) Notes

■ Notes for specifying the scope

If the script variable name and the script variable scope are repeated in multiple custom tags, the JSP compilation results differ depending on the versions of the Application Server. Specify the script variable scope using one of the followings:

- Subclass of `javax.servlet.jsp.tagext.TagExtraInfo` class
- Scope element in the variable element of the TLD file

The followings are the differences in operations in each Application Server version when script variables with the same names are specified in the JSP custom tag:

- **With versions prior to 07-00**
  Perform variable declaration of the script variable in the Java code generated from JSP corresponding to the custom tag from the second time onwards.
- **With version 07-00 and later versions**
  Do not perform variable declaration of the script variable in the Java code generated from JSP corresponding to the custom tag from the second time onwards.

The following is an example wherein the scope is `AT_BEGIN` and custom tag `<my:foo>` defines the script variable of the variable name (`var`) specified in the attribute id:

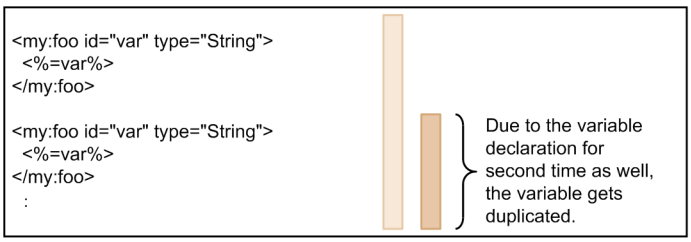
Example of definition

```
<my:foo id="var" type="String">
 <%=var%>
</my:foo>

<my:foo id="var" type="String">
 <%=var%>
</my:foo>
```

- With versions prior to 07-00

  Perform variable declaration with the script variable var in the Java code generated from JSP corresponding to the second custom tag. In this case, the range of scope 'AT_BEGIN' in each custom tag is as follows:



Legend:

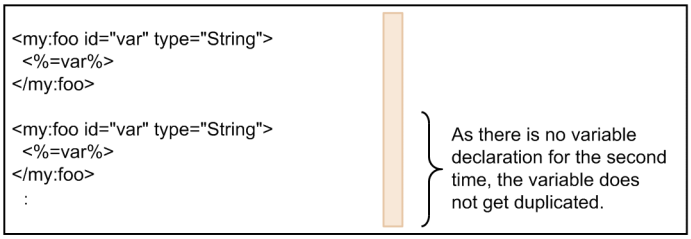
: Validity scope of the script variable defined in the custom tag for the first time

: Validity scope of the script variable defined in the custom tag for the second time

  Therefore, compilation error occurs during the compilation of the Java source generated from JSP.

- With version 07-00 and later versions

  Do not perform variable declaration with the script variable var in the Java code generated from JSP corresponding to the second custom tag. In this case, the range of scope 'AT_BEGIN' in each custom tag is as follows:



Legend:

: Validity scope of the script variable defined in the custom tag for the first time

  Therefore, even if the script variables of the same name are defined in the same scope, the compilation of the Java source generated from JSP is executed normally.

■ Notes for coding the scriptlet

When the Java source is generated from JSP, the Java code described in the scriptlet is not analyzed. Therefore, if you define the process for which the script variable scope is changed before and after the custom tag or in the body scriptlet, the JSP compiled results might return an error depending on the version of the Application Server version.

- **Example of compilation error when the scope is 'AT_BEGIN'**

  In this example, script variable of variable name (`var`) specified in the attribute id is defined in the custom tag `<my:foo>` and scope 'AT_BEGIN'.

```
<% if(flag) { %>

 <my:foo id="var" type="String">
  <%=var%>
 </my:foo>

<% } else { %>

 <my:foo id="var" type="String">
  <%=var%>
 </my:foo>
```

```
<% } %>
```

The operations for each Application Server version in this example are as follows:

- With versions prior to 07-00

  Perform variable declaration with script variable var in the Java code generated from JSP corresponding to the second custom tag. Therefore, error does not occur even when referencing the second script variable `var.JSP` compilation is executed normally.

- With version 07-00 and later versions

  In the second custom tag, the analysis is that the script variable var is already declared. In this case, variable declaration will not be performed with variable var in the Java code generated from JSP corresponding to the custom tag. Therefore, an error occurs in the reference of the second script variable var.

(2) Differences in JSP compilation operations depending on the usage of the JSP translation backward compatibility function

The differences in the compilation operations when the JSP translation backward compatibility function is used and when the function is not used with version 07-00 and later versions are as follows:

When the script variable name and the script variable scope is repeated in multiple custom tags:

- **Use the JSP translation backward compatibility function**

  Perform variable declaration of script variables in Java code generated from JSP corresponding to the custom tag from the second time onwards.

- **Do not use the JSP translation backward compatibility function**

  Do not perform variable declaration of script variables in Java code generated from JSP corresponding to the custom tag from the second time onwards.

(3) Defining the JSP translation backward compatibility function

To define the JSP translation backward compatibility function, specify the following parameter in the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file:

`webserver.jsp.translation.backcompat.customAction.declareVariable`

Specifies whether or not to output the variable declaration of the script variables corresponding to the second custom tag in the Java code generated from the JSP file.

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

6.3.2 Precautions related to the class attributes of <jsp:useBean> tag

This subsection describes the precautions related to the class attributes of the `<jsp:useBean>` tag, differences in JSP compilation operations depending on the usage of the JSP translation backward compatibility function, and the definition of the JSP translation backward compatibility function.

(1) Notes

In the `<jsp:useBean>` tag, JSP specifications define that class attribute is the implementation class name of the object. Complying with these specifications, the following checks are implemented during the JSP translation for the classes specified in the class attribute, with the version 07-00 and later versions:

- The class modifier is public.
- The class modifier is not an interface.
- The class modifier is not abstract.
- The method modifier is public and a constructor that is not an argument exists.

Therefore, when a class that does not meet these check items is specified in the class attribute, the JSP compilation results are different with the versions prior to 07-00 and with version 07-00 and later versions.

The following is an example of JSP wherein the implementation class is specified in the include source and an interface is specified in the include destination:

- `test1.jsp` (include source)

```
<jsp:useBean id="bean" scope="request" class="test.TestBean"/>
<jsp:include page="test1_included.jsp"/>
```

- `test1_included.jsp` (include destination)

```
<jsp:useBean id="bean" scope="request" class="test.TestBeanIF"/>
```

Note:

`test.TestBean` specified in the class attribute is an implementation class compliant with the JSP specifications and test.TestBeanIF becomes the interface of `test.TestBean`.

The operations for each Application Server version in this example are as follows:

- With versions prior to 07-00

  Since the check is not implemented during JSP translation, the servlet generated from the JSP file is executed.

  In the servlet generated from the JSP file, the existing script variables are searched with "bean" specified in the `id` attribute. In the above description, since the same script variables are already defined in the include source (`test1.jsp`), the existing object (instance of the `test.TestBean` class) is used without being instantiated from the class (`test.TestBeanIF` interface) specified in the class attribute. Therefore, the operations are executed normally.

- With version 07-00 and later versions

  The checks for the class specified in the class attribute are implemented for `test1_included.jsp` during JSP translation, therefore, an error occurs in JSP translation.

(2) Differences in JSP compilation operations depending on the usage of the JSP translation backward compatibility function

The differences in the compilation operations when the JSP translation backward compatibility function is used and when the function is not used with version 07-00 and later versions are as follows:

When a class name that cannot be instantiated is specified in the class attribute

- Use the JSP translation backward compatibility function

  Bean can be acquired without the `id` attribute value specified in the `<jsp:useBean>` tag from the second time onwards without resulting in an error.

- Do not use the JSP translation backward compatibility function

  An error occurs during JSP translation.

(3) Defining the JSP translation backward compatibility function

To define the JSP translation backward compatibility function, specify the following parameter in the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file:

`webserver.jsp.translation.backcompat.taglib.noCheckPrefix`

Specifies whether or not to check the class property value of the `<jsp:useBean>` tag during JSP translation.

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

6.3.3 Precautions related to the Expression check of the tag attribute values

This subsection describes the precautions related to the Expression check of the tag attribute values, differences in JSP compilation operations depending on the usage of the JSP translation backward compatibility function, and the definition of the JSP translation backward compatibility function.

(1) Notes

In the JSP specifications, the tag attributes specified in the Expression have been limited. With the version 07-00 and later versions, if Expression is specified in the attributes other than the attributes that can specify Expression without using the JSP translation backward compatibility function, an error will occur during the JSP translation. However, the versions prior to 07-00 do not check whether the Expression can be specified during the JSP translation. As a result, `<%=` and `%>` indicating Expression are recognized as strings and an error does not occur. Therefore, if Expression is specified in attributes other than the attributes that can specify the Expression, the JSP compiled results are different with the version 07-00 and its prior and later versions.

When using JSP that specifies Expression in the attributes other than the attributes that can specify Expression, make sure that you set up the JSP translation backward compatibility function.

(2) Differences in JSP compilation operations depending on the usage of the JSP translation backward compatibility function

The following are the differences in the compilation operations when the JSP translation backward compatibility function is used and when the function is not used with version 07-00 and later versions:

When Expression is specified in the attribute value of the tag that does not permit the specification of the Expression

- **Use the JSP translation backward compatibility function**
  The Expression specified in the attribute value of the tag that does not permit the specification of the Expression is handled as a string.
- **Do not use the JSP translation backward compatibility function**
  An error occurs during JSP translation.

(3) Defining the JSP translation backward compatibility function

To define the JSP translation backward compatibility function, specify the following parameter in the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file:

`webserver.jsp.translation.backcompat.tag.noCheckRtexprvalue`

Specifies whether or not to determine if the Expression is specified in the attribute value of the tag that cannot specify the Expression.

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

6.3.4 Precautions related to Expression specified in the tag attribute values

This subsection describes the precautions related to Expression specified in the tag attribute values, differences in JSP compilation operations depending on the usage of the JSP translation backward compatibility function, and the definition of the JSP translation backward compatibility function.

(1) Notes

When specifying the Expression in the attribute value of the tag, specify as `"<%= scriptlet_expr %>"` or `'<%= scriptlet_expr %>'`.

If the tag attribute value starts with "<%= (or '<%=) and does not end with %>" (or %>'), the value enclosed within " (or ') is handled as a string with versions prior to the 07-00 version. For example, if an optional string exists between %> and ", the value enclosed within " is handled as a string. However, with version 07-00 and later versions, %>" (or %>') is handled as the end of the attribute value, therefore, an error occurs during JSP translation.

If the tag attribute value starts with "<%= (or '<%=) and does not end with %>" (or %>'), make sure that you set up the JSP translation backward compatibility function.

(2) Differences in JSP compilation operations depending on the usage of the JSP translation backward compatibility function

The differences in the compilation operations when the JSP translation backward compatibility function is used and when the function is not used with version 07-00 and later versions are described below:

When the tag attribute value starts with "<%= (or '<%=) and does not end with %>" (or %>')

- **Use the JSP translation backward compatibility function**
  The attribute value enclosed within " (or ') is processed as a string.
- **Do not use the JSP translation backward compatibility function**
  The attribute value enclosed within " (or '), is processed as an Expression.

(3) Defining the JSP translation backward compatibility function

To define the JSP translation backward compatibility function, specify the following parameter in the <configuration> tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file.

webserver.jsp.translation.backcompat.tag.rtexprvalueTerminate

Specifies whether or not to handle as a string the value enclosed within " (or ') of the attribute value in which the tag attribute value starts with "<%= or '<%= and does not end with %>" (%>' when started with '<%).

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

6.3.5 Precautions related to the prefix attribute of the taglib directive

This subsection describes the precautions related to the prefix attribute of the taglib directive, differences in JSP compilation operations depending on the usage of the JSP translation backward compatibility function, and the definition of the JSP translation backward compatibility function.

(1) Notes

In the JSP specifications, you cannot use a custom tag that uses the prefix specified in the taglib directive before the taglib directive. The version 07-00 and later versions check whether the custom tag that uses the prefix specified in the taglib directive is coded before the taglib directive according to the JSP specifications. If the custom tag that uses the prefix specified in the taglib directive is coded before the taglib directive, an error occurs during translation. However, with versions prior to 07-00, this check is not performed, therefore, the coded custom tag is handled as a string.

Therefore, if the custom tag is coded using the prefix specified in the taglib directive before the taglib directive, the JSP compiled results are different with the versions prior to 07-00 and with version 07-00 and later versions.

If the custom tag that uses the prefix specified in the taglib directive is coded before the taglib directive, make sure that you set up the JSP translation backward compatibility function.

(2) Differences in JSP compilation operations depending on the usage of the JSP translation backward compatibility function

The differences in the compilation operations when the JSP translation backward compatibility function is used and when the function is not used with version 07-00 and later versions are as follows:

When the custom tag that uses the prefix specified in the taglib directive is coded before the taglib directive

- **Use the JSP translation backward compatibility function**
  The custom tag is handled as a string and not as a custom tag.
- **Do not use the JSP translation backward compatibility function**
  An error occurs during JSP translation.

(3) Defining the JSP translation backward compatibility function

To define the JSP translation backward compatibility function, specify the following parameter in the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file:

`webserver.jsp.translation.backcompat.useBean.noCheckClass`

Specifies whether or not to execute the check process of the class attribute value of the `<jsp:useBean>` tag during JSP translation.

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

Appendixes

A. Error Status Code

This appendix describes the error status codes returned by the Web container, redirector, and in-process HTTP server.

The location where the error occurs depends on the used Web server. Reference the error status code according to the location where the error has occurred. The following table describes the used Web servers and the corresponding locations where the error has occurred.

Table A-1: Used Web server and the corresponding location where the error occurred

Cosminexus HTTP Server to be used	Location where the error occurred		
	Web container	Redirector	HTTP In-process HTTP server
Hitachi Web Server or Microsoft IIS	Y	Y	N
In-process HTTP server	Y	N	Y

Legend:

Y: Error occurs

N: Error does not occur

A.1 Error status codes returned by the Web container

When the client accesses a non-existent resource or a servlet in which an exception occurred, the Web container returns an error status code. The following table describes the error status codes returned by the Web container, and the conditions for returning the error status codes.

Table A-2: Error status codes returned by the Web container and conditions for returning the error status codes

Error status codes	Conditions for returning the error status code
400 Bad Request	The error status code 400 is returned in either of the following cases: • When a client directly sends a request to a resource that is specified as the login page used for Form authentication, and the user is successfully authenticated from the login page that is displayed as a result of the request • When the access satisfies all the following three conditions: 1. The version of HTTP is "HTTP/1.0". 2. The servlet to be accessed inherits `javax.servlet.http.HttpServlet`. 3. The HTTP method at the access is not overwritten by the servlet. • When an access is made by a request header with a Content-Length header value of greater than 2147483647 or smaller than 0 • When an access is made by a request header with a non-numeric Content-Length header value • When an access is made by a request header containing multiple Content-Length headers • When request URIs cannot be normalized
401 Unauthorized	The error status code 401 is returned when a resource that requires Basic authentication is accessed as follows: • The access uses an invalid user ID or password. • The access does not include authentication information (the Authorization header).
403 Forbidden	Error status code 403 is returned, when any of the following conditions is applicable: • When a resource that requires the Basic or Form authentication is accessed using an unauthenticated user name

Error status codes	Conditions for returning the error status code
403 Forbidden	• When a resource that does not permit any access is accessed without any role-name element for the auth-constraint element being specified in `web.xml`[#1] • When static contents are accessed by using the PUT or DELETE method • When a resource whose `<transport-guarantee>` element in `web.xml` is set to INTEGRAL or CONFIDENTIAL is accessed via http[#2]
404 Not Found	Error status code 404 is returned when any of the following is accessed: • When a non-existent resource is accessed • When a servlet or a JSP file in which `javax.servlet.UnavailableException` occurs, is accessed[#3]
405 Method Not Allowed	Error status code 405 is returned in the case of an access that satisfies all of the following three conditions: • When the HTTP version is "HTTP/1.1" • When the servlet to be accessed inherits `javax.servlet.http.HttpServlet` • When the HTTP method during access does not get overridden by the corresponding servlet
412 Precondition Failed	Error status code 412 is returned, when the static contents that do not match the conditions specified in If-Match header or If-Unmodified-Since header, are accessed.
413 Request Entity Too Large	Error status code 413 is returned when the size of the request body exceeds the upper-limit value.
416 Requested Range Not Satisfiable	Error status code 416 is returned, when the static contents that use the value of an invalid Range header applicable to any of the following cases, are accessed: • The value of Range header does not begin with "byte" • A numeric character and "-" is not used in range definition • The specified range is not appropriate
500 Internal Server Error	Error status code 500 is returned, when any of the following conditions is applicable: • When a servlet or a JSP file in which an exception occurs, is accessed[#4] • When a JSP file whose compilation failed is accessed • When deleted static contents are accessed[#5] • When I/O error occurs when accessing the static contents • When a resource protected by `<auth-constraint>` element is accessed, when the definition of `web.xml` is invalid[#6]
501 Not Implemented	Error status code 501 is returned, when the static contents or the servlet that inherits `javax.servlet.http.HttpServlet` is accessed by an HTTP method other than the GET, HEAD, POST, PUT, DELETE, OPTIONS, and TRACE method.
503 Service Unavailable	Error status code 503 is returned, when any of the following conditions is applicable: • When there is no space in the pending queue of requests[#7] • When a servlet or JSP file in which `javax.servlet.UnavailableException` occurs, is accessed[#8] • When a Web container being shut down is accessed • When a Web application that is in abnormal state due to an unexpected error or exception, is accessed • When only the Web container server is running and any Web applications are not running in the Web container server, and the Web application that was not started is accessed.

#1

Applicable when the Web application is version 2.4 or later.

#2

This applies to the case when the port number of the https, used by the URL an access is forwarded to, is not set for the `webserver.connector.redirect_https.port` key in usrconf.properties.

#3

Applicable when the version of the Web application is 2.4 or later, and `javax.servlet.UnavailableException` indicating the permanent unavailability occurs, and the exception is not caught in the servlet and JSP file.

#4

Applicable in the following cases:

- When the version of the Web application is 2.4 or later
  When exception is not caught in servlets or JSPs
- When the version of the Web application is 2.3
  When the error page is not specified in the `<error-page>` tag of `web.xml`, or in the page directive of a JSP file, and the exception is not caught in the servlet or JSP file

#5

Applicable when the reload functionality of the Web application, re-compilation functionality of the JSP file, or the reload functionality of J2EE application is not used.

#6

Applicable when the `<role-name>` element is defined in the `<auth-constraint>` element of `web.xml`, and the `<login-config>` element is not defined. If the application is started in this state, warning message KDJE39150-W is output to the console window, and in the message log.

#7

Applicable when the settings to control the number of concurrently executing threads in the Web application, or in the URL group are specified.

#8

Applicable in the following cases:

- When the version of Web application is 2.4 or later
  When `javax.servlet.UnavailableException` indicating the temporary unavailability occurs, and the exception is not caught in the servlets or JSPs
- When the version of Web application is 2.3
  When the error page is not specified in `<error-page>` tag of `web.xml`, or in the page directive of JSP file, and the exception is not caught in the servlet or JSP file

A.2 Error status codes returned by the Redirector

When a timeout occurs during a data transaction with the Web container, and when the coding of the definition file contains an error, the redirector returns an error status code. The following tables describe the error status codes returned by the redirector and the conditions for returning the error status code, for each type of Web server.

Table A-3: Error status codes returned by the redirector and the occurrence conditions (for Cosminexus HTTP Server)

Error status codes	Conditions for returning the error status code
400 Bad Request	The error status code 400 is returned in either of the following cases: • When the port number of Host header of the request is invalid • When the request method is not POST[#1] • When the request does not have a Content-Length header (for a POST request, the body is a chunk format)[#1] • When the Content-Length header value of the request exceeds the upper limit of the POST data size set in the POST request sending destination worker [#1]
500 Internal Server Error	Error status code 500 is returned, when any of the following conditions is applicable: • When there is a coding error in the contents of `mod_jk.conf`[#2] • When there is a failure in reading, or a coding error in the contents of `workers.properties`[#1]

Error status codes	Conditions for returning the error status code
500 Internal Server Error	• When the request header exceeds 16 KB[#3] • When failed to establish connection with the Web container • When timeout occurs during establishment of a connection to the Web container • When an error occurs during sending data to the Web container • When timeout occurs during sending data to the Web container • When an error occurs during receiving data from the Web container • When timeout occurs during receiving data from the Web container • When timeout occurs in reading the POST data from the client • When an unsupported HTTP method[#4] is specified in the request

#1

Applicable when the default worker is not specified in the distribution by the POST data size.

#2

Applicable only in Windows. The web server fails to start in UNIX.

#3

Might be applicable when a request header of a total size of 16 KB or more can be received according to the settings for limitations of request headers in Cosminexus HTTP Server.

#4

For details on whether the HTTP method is supported or not, see *Table A-5*.

Table A-4: Error status codes returned by the redirector and the occurrence conditions (for Microsoft IIS)

Error status codes	Conditions for returning the error status code
400 Bad Request	Error status code 400 is returned, when any of the following conditions is applicable: • When the request URL contains a `%` (percent sign) and the two characters after the `%` (percent sign) do not express hexadecimals (characters other than A-F, a-f, or 0-9) • When the port number of Host header of the request is invalid
403 Forbidden	Error status code 403 is returned, when any of the following conditions is applicable: • When the request URL begins with "hitachi_ccfj"[#1] • When the request URL contains "%2F"[#1]
500 Internal Server Error	Error status code 500 is returned, when any of the following conditions is applicable: • When there is a coding error in the contents of `isapi_redirect.conf` • When there is failure in reading, or a coding error in the contents of `workers.properties` • When the request header exceeds 16 KB[#2] • When failed to establish connection with the Web container • When timeout occurs during establishment of a connection to the Web container • When an error occurs during sending data to the Web container • When timeout occurs during sending data to the Web container • When an error occurs during receiving data from the Web container • When timeout occurs during receiving data from the Web container • When timeout occurs in reading the POST data from the client • When an unsupported HTTP method[#3] is specified in the request

#1

Not case-sensitive.

#2

Might be applicable when a request header of a total size of 16 KB or more can be received according to the settings for limitations of request headers in Microsoft IIS.

#3

For details on whether the HTTP method is supported or not, see *Table A-5*.

Support for the request HTTP methods in the redirector

The following table lists the request HTTP methods that are supported in the redirector.

Table A-5: Support for the request HTTP methods in the redirector

HTTP method	Supported or not
OPTIONS	Y
GET	Y
HEAD	Y
POST	Y
PUT	Y
DELETE	Y
TRACE	Y
CONNECT	--#
PROPFIND	Y
PROPPATCH	Y
MKCOL	Y
COPY	Y
MOVE	Y
LOCK	Y
UNLOCK	Y
ACL	Y
REPORT	Y
VERSION-CONTROL	Y
CHECKIN	Y
CHECKOUT	Y
UNCHECKOUT	Y
SEARCH	Y
Methods available in HTTP1.1 other than above-mentioned methods	--#

Legend:

Y: Supported

--: Not supported

#

The redirector returns a status 500 error to a request that specifies an unsupported HTTP method. Furthermore, the message KDJE41001-E is output.

A.3 Error status codes returned by the in-process HTTP server

When the size of the request from the client exceeds the upper limit and if the value is invalid, the in-process HTTP server returns the error status code. The following table describes the error status codes returned by the in-process HTTP server, and the conditions for returning the error status codes.

Table A-6: Error status codes returned by the in-process HTTP server and the occurrence conditions

Error status codes	Conditions for returning the error status code
400 Bad Request	Error status code 400 is returned, when any of the following conditions is applicable: • When the request HTTP version is 1.1 and the Host header does not exist • When the port number of Host header of the request is invalid • When the size of the request header exceeds the upper limit • When the number of request headers exceeds the upper limit • When the request URI is invalid • When an attempt to decode the request URI has failed • When the request URI cannot be normalized • When the Content-Length header value of the request is greater than 2147483647 or smaller than or 0 • When the Content-Length header value of the request is a non-numeric value • When multiple Content-Length headers are specified for the request • When the HTTP version of the request line is not supported
403 Forbidden	The error status code 403 is returned when a resource whose `<transport-guarantee>` element in `web.xml` is set to INTEGRAL or CONFIDENTIAL is accessed via http.
405 Method Not Allowed	Error status code 405 is returned when access is made from an HTTP method that is not permitted
413 Request entity too large	Error status code 413 is returned when the request body size exceeds the upper limit.
414 Request-URI too large	Error status code 414 is returned when the length of the request line exceeds the upper limit.
500 Internal Server Error	Error status code 500 is returned when an attempt to read the file fails and the file is returned with the status code 200 by the redirect functionality.
501 Not Implemented	Error status code 501 is returned when the transfer-encoding header value of the request is not supported.
503 Service Unavailable	Error status code 503 is returned when an attempt is made to process the requests exceeding the upper limit of flow control.

B. Precautions related to Cosminexus HTTP Server Settings

This appendix describes the precautions related to the Cosminexus HTTP Server settings.

B.1 Precautions for restarting Cosminexus HTTP Server

If the cause of a failure that occurred while restarting Cosminexus HTTP Server exists in the Easy Setup definition file (if the Smart Composer functionality is not used, the redirector definition file (`mod_jk.conf`) or the workers file (`workers.properties`)), the message is output to one of the following:

- Start command execution window or event log of Cosminexus HTTP Server
  - When Cosminexus HTTP Server is started, stopped, or restarted from the Command Prompt, the message is output to the Start command execution window.
  - When Cosminexus HTTP Server is started, stopped, or restarted from the service, the message is output in the event log. This is applicable only in Windows.
- Error log files for Cosminexus HTTP Server

  By default, error log files are output to the following locations:
  - In Windows

    Cosminexus HTTP Server -installation-directory`\logs\error.log`
  - In UNIX

    `/opt/hitachi/httpsd/logs/error.log`

The following table describes each of the output messages:

Table B-1: Messages output in the Start command execution window or event log of Cosminexus HTTP Server

Message	Cause and Action
JkWorkersFile file_name invalid#	The file name specified in the `JkWorkersFile` key is invalid. Correct the value specified in the `JkWorkers`File key, and then restart the Web server.
Can't find the workers file specified#	The specified workers file cannot be found. Check whether the file specified in the `JkWorkersFile` key exists, and then restart the Web server.
Content should start with /#	The first character in the URL pattern is not forward slash (/). Correct the first character of the URL pattern specified in the `JkMount` key to forward slash (/), and then restart the Web server.
JkOptions: Illegal option '*a...a*'	The value (*a...a*) specified in the `JkOption` key is invalid. Correct the value specified in the `JkOption` key, and then restart the Web server.
a...a takes *b...b* arguments,	The number of values that can be specified in the key shown in *a...a* is *b...b*. Correct the number of values that are specified in the key shown in *a...a*, and then restart the Web server.
a...a must be On or Off	The value that can be specified in the key shown in *a...a* is either `On` or `Off`. Change the value specified in the key shown in *a...a* to `On` or `Off`, and then restart the Web server.
JkModulePriority: Invalid value specified.*a...a*	The value *a...a* specified in the `JkModulePriority` key is invalid. Specify the correct value in the `JkModulePriority` key, and then restart the Web server.

#

This message is output only in UNIX.

Table B-2: Error log files of Cosminexus HTTP Server

Message	Description and action
[Time] [emerg] Memory error	The memory is insufficient. Secure the memory that the system can use, and then restart the Web server.
[Time] [emerg] Error while opening the workers#	One of the following problems occurred: • Opening of the workers file failed. Check whether read permission is included in the access permission of the file specified in the `JkWorkersFile` key, and then restart the Web server. • The contents of the workers file are invalid. Correct the contents of the workers file, and then restart the Web server.
[Time] [emerg] Error while checking the `mod_jk.conf`#	The description in the redirector definition file is inappropriate. Check the message output in the redirector, and then restart the Web server.

#
This message is output only in UNIX.

B.2 Precautions related to the redirector log

- If there is no permission to write the Cosminexus HTTP Server execution account in the log output destination directory, the request is processed normally, but the log is not output.
- In Windows, by default the log output destination directory (*Cosminexus-installation-directory*`\CC\web\redirector\logs`) of the redirector does not exist. Therefore, when Cosminexus HTTP Server starts, the access permission of the directory that is one level higher (redirector directory) is inherited, the logs directory is generated, and an attempt is made to output the log. At this time, if there is no write permission for the Cosminexus HTTP Server execution account, the log is not output. In this case, generate the logs directory and set the access permission or set the access permission for the directory that is one level higher (redirector directory).
  Furthermore, when the log output destination directory of the redirector is changed and the specified path exists only halfway, set the access permission for the existing lowest directory or create the directory corresponding to the specified path completely and set the access permission for the lowest directory.

B.3 Precautions for upgrading Cosminexus HTTP Server

When Cosminexus HTTP Server is upgraded, you must restart Cosminexus HTTP Server to reflect the changes in the redirector for Cosminexus HTTP Server. For details on how to restart Cosminexus HTTP Server, see the *uCosminexus Application Server HTTP Server User Guide*.

C. Microsoft IIS Settings

This appendix describes how to specify the settings when integrating with a Web server using Microsoft IIS.

C.1 Microsoft IIS 7.0 or Microsoft IIS 7.5 settings

This subsection describes how to specify the settings for Microsoft IIS 7.0 or Microsoft IIS 7.5 when integrating with a Web server using Microsoft IIS.

To integrate a J2EE server with Microsoft IIS 7.0 or Microsoft IIS 7.5:

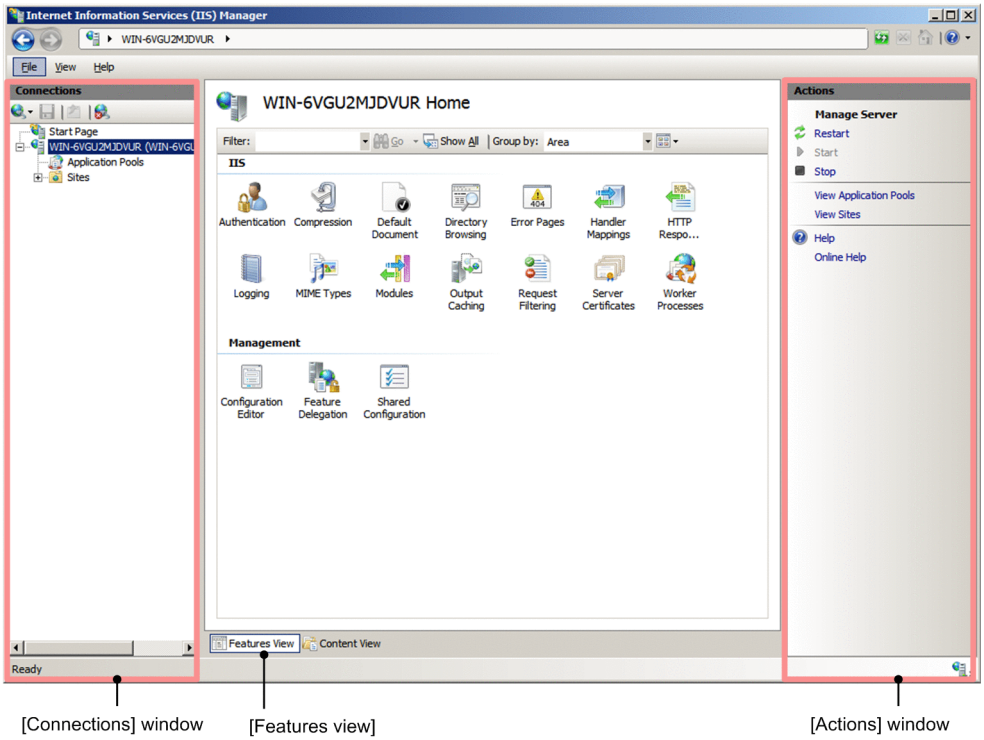
1. Install a role service
2. Add the ISAPI and CGI limitations
3. Add an ISAPI filter
4. Set the handler mapping
5. Add a virtual directory
6. Set an application pool in the **Server** node
7. Set an application pool in the **Site** node
8. Set the access permissions for the log output destination directory of the redirector
9. Start the Web site

Reference note

- Some of the window notations are different in Microsoft IIS 7.0 and Microsoft IIS 7.5. In the procedures hereafter, the description is based on the notations for Microsoft IIS 7.5. If you are using Microsoft IIS 7.0, read the notations as described in the following table:

Window notations in Microsoft IIS 7.5	Window notations in Microsoft IIS 7.0
Server Manager	Server Manager
Internet Information Service (IIS) Manager	Internet Information Service (IIS) Manager
ISAPI Filter	ISAPI Filter
Filter Name	Filter Name
Handler Mapping	Handler Mapping
Worker Process	Worker Process

- The window configuration of **Internet Information Service (IIS) Manager** used in the settings of Microsoft IIS 7.0 or Microsoft IIS 7.5 is as follows:



Specify the settings in the Connections window, Features View, and Actions window according to the following procedure:

(1) Installing a role service

When using Microsoft IIS 7.0 or Microsoft IIS 7.5, you must install a role service according to the usage. To install a role service:

1. In the **Start** menu, from **All Programs - Management Tools**, start **Server Manager**.
2. Under **Server Manager**, select **Roles**, and then under **Web Server (IIS)**, click **Add Role Service**.
3. In the **Add Role Service** dialog box, select the following role services:
   - Static contents
   - Existing documents
   - Reference directory
   - HTTP error
   - ISAPI extension
   - ISAPI filter
   - IIS management console
4. Click **Install**.
5. Click **Close**.
   Installation is complete.

(2) Adding the ISAPI and CGI limitations

To add the limitations of ISAPI and CGI:

1. In the **Start** menu, from **All Programs - Management Tools**, start **Internet Information Service (IIS) Manager**.
2. In the **Features View** server, from the **Home** page, double-click **ISAPI and CGI Limitations**.

3. In the **ISAPI and CGI Limitations** page, under the Actions window, click **Add**.
4. In the **Add ISAPI or CGI Limitations** dialog box, perform the following operations:
   - In **ISAPI or CGI Path**, specify the DLL (*Cosminexus-installation-directory*`\CC\web\redirector\isapi_redirect.dll`) of the redirector.
   - In **Description**, enter **ISAPI**.
   - Check the **Allow Execution of Extension Path** checkbox.
5. Click the **OK** button.
   Close the **Add ISAPI or CGI limitations** dialog box to apply the settings.

(3) Adding an ISAPI filter

To add an ISAPI filter:

1. In the **Features View** site, from the **Home** page, double-click **ISAPI Filter**.
2. In the **ISAPI Filter** page, under the Actions window, click **Add**.
3. In the **Add ISAPI Filter** dialog box, perform the following operations:
   - In **Filter Name**, enter `hitachi_ccfj`.
   - In **Executable File**, specify the DLL (*Cosminexus-installation-directory*`\CC\web\redirector\isapi_redirect.dll`) of the redirector.
4. Click the **OK** button.
   The **Add ISAPI Filter** dialog box closes, and the settings are applied.

(4) Setting the handler mapping

To set the handler mapping:

1. In the **Features View** site, from the **Home** page, double-click **Handler Mapping**.
2. In the **Handler Mapping** page, select **ISAPI-dll**, and then in the Actions window, click **Edit**.
3. In the **Edit Module Map** dialog box, perform the following operations:
   - In the **Requested Path**, enter ***.dll**.
   - In **Executable File**, specify the DLL (*Cosminexus-installation-directory*`\CC\web\redirector\isapi_redirect.dll`) of the redirector.

   If the handler mapping is already set, the Edit Script Map dialog box will run; however, the operation contents are the same.
4. Click the **OK** button.
5. In the message dialog box for confirming whether or not to enable the editing of the module map, click the **Yes** button to enable the ISAPI extension functionality.
6. In the **Handler Mapping** page, select **ISAPI-dll**, and then in the Actions window, click **Edit Access Permission of Functionality**.
7. In the Edit Access Permission of Functionality dialog box, check all access permissions including **Read**, **Script**, and **Execute**.
8. Click the **OK** button.
   The Edit Access Permission of Functionality dialog box closes, and the settings are applied.

(5) Adding a virtual directory

Add a virtual directory named `hitachi_ccfj`. To add the virtual directory:

1. In the Connections window, expand the **Site** node, and click the site for adding the virtual directory.
2. In the Actions window, click **Display Virtual Directory**.
   The **Virtual Directory** page appears.

3. In the **Virtual Directory** page, under the Actions window, click **Add Virtual Directory**.
4. In the Add Virtual Directory dialog box, perform the following operations:
   - In **Alias**, enter `hitachi_ccfj`.
   - In **Physical path**, specify the directory in which the DLL (*Cosminexus-installation-directory*`\CC\web\redirector\isapi_redirect.dll`) of the redirector is saved.
5. Click the **OK** button.
   The Add Virtual Directory dialog box closes, and the settings are applied.

(6) Setting an application pool in the Server node

To set an application pool in the **Server** node:

1. In the Connections window, expand the **Server** node, and then click **Application Pool**.
2. In the **Application Pool** page, select the application pool to be used, and then in the Actions window, click **Detailed Settings**.
3. In the Detailed Settings dialog box, perform the following operation:
   - In **Enable 32-bit Application**, specify **True** (This specification is necessary only to run the Redirector of Windows x86 on Windows x64 OS).
   - In **Maximum Number of Worker Processes**#, specify the maximum number of processes for processing requests in Microsoft IIS.

   #
   In this manual, the execution processes of the Web container are also referred to as *Worker processes*; however, the *Worker processes* set here are the processes used to process requests in Microsoft IIS.
4. Click the **OK** button.
   The Detailed Settings dialog box closes, and the settings are applied.

(7) Setting an application pool in the Site node

To set an application pool in the **Site** node:

1. In the Connections window, expand the **Site** node, and then click the site in which to specify the application pool.
2. In the Actions window, click **Detailed Settings**.
3. In the Detailed Settings dialog box, enter **Application Pool**.
   In Application Pool, specify the name of the application pool set in *(6) Setting an application pool in the Server node*.
4. Click the **OK** button.
   The Detailed Settings dialog box closes, and the settings are applied.

(8) Setting access permissions for the log output destination directory of the redirector

In the log output destination directory of the redirector, you must add the permission for writing to the execution account of the application pool of Microsoft IIS. Set the access permissions for the log output destination directory of the redirector from **Explorer**.

Specify the execution account of the application pool in the Detailed Settings dialog box of the application pool, under **ID**. If the default ID is specified, add the write permission for the **IIS_IUSRS** group.

The default IDs are as follows:

- Microsoft IIS 7.0: `NetworkService`
- Microsoft IIS 7.5: `ApplicationPoolIdentity`

Note that during new installation, by default, the log output destination directory of the redirector (*Cosminexus-installation-directory*`\CC\web\redirector\logs`) does not exist. Therefore, either create the logs directory and

set the access permission, or set the access permission for the directory that is one level higher (`redirector` directory).

Also, when the log output destination directory of the redirector is changed and the specified path exists up to only halfway, either set the access permission for the existing lowermost directory, or create all directories corresponding to the specified path and set the access permission for the lowermost directory.

(9) Starting the Web site

To start the Web site of Microsoft IIS:

1. In the Connections window, expand the **Site** node, and then click the site to be started.
2. In the Actions window, click **Start**. If the site is already started, click **Restart**.
   The Web site either starts or is restarted.

(10) Notes

This subsection describes the notes on setting Microsoft IIS 7.0 or Microsoft IIS 7.5.

(a) Notes on replication of configuration settings in multiple environments

In Microsoft IIS 7.0 or Microsoft IIS 7.5, you can save the configuration settings in the `web.config` file. Also, based on the saved `web.config` file, you can replicate the configuration settings in multiple environments using `xcopy`.

However, for a configuration environment in which the redirector is used, you cannot set the redirector when replicating the settings in multiple environments using `xcopy`. Even when you specify the same configuration settings for multiple environments using `xcopy`, set the redirector manually in each environment.

(b) Notes on customizing the error page

If you have specified the settings for returning a custom error page in the error page of Microsoft IIS 7.0 or Microsoft IIS 7.5, the customization of the error page by the **<error-page>** tag of `web.xml` might be disabled. If you want to enable the customization of the error page by the **<error-page>** tag of `web.xml`, disable the settings of custom error page in the error page of Microsoft IIS 7.0 or Microsoft IIS 7.5.

(c) Points to be noted when using along with other filters

If Microsoft IIS receives requests that are to be forwarded to the Web container, the Redirector for Microsoft IIS changes the request URL information to be used in the ISAPI filter. Therefore, the request URL received by Microsoft IIS cannot be retrieved in the ISAPI filter if the ISAPI filter is executed after the Redirector for Microsoft IIS. Accordingly, you must make the settings in such a way that the filter is executed before the Redirector for Microsoft IIS, and the filter can retrieve the URL requests received by Microsoft IIS. To change the execution order, you can set the execution priority of the Redirector for Microsoft IIS by setting the value of the filter_priority key in the `isapi_redirect.conf` file (Operation Definition file of the Redirector for Microsoft IIS) to "Medium" or "Low". For details on the filter_priority key of the `isapi_redirect.conf` file (Operation Definition file of the Redirector for Microsoft IIS) see *9.2 isapi_redirect.conf file (Operation Definition file of the Redirector for Microsoft IIS)* in the *uCosminexus Application Server Definition Reference Guide*.

D. Main Functionality Changes in Each Version

This subsection describes the main functionality changes in each version of Application Server earlier than version 09-50. A purpose oriented description of changes is given here. For the main functionality changes in version 09-50, see *1.4 Main updates in the functionality of Application Server 09-50*.

The descriptions are as follows:

- The main functionality that is changed in Application Server 08-00 and the overview are described. For details on the functionality, see the description in the *Reference* column of the following table. The *Reference* column specifies the main locations where the functionality is described.
- If the details on the changed items are described in this manual, the *Reference* column lists the description location (section) of this manual. If the description is in another manual, the name of that manual is mentioned. Note that *uCosminexus Application Server* is omitted in the description of each manual name.

D.1 Main functionality changes in 09-00

(1) Simplifying implementation and setup

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

Table D-1: Changes made for simplifying implementation and setup

Item	Overview of changes	Reference manual	Reference
Changed the unit of the setup and operation target in the virtual environment	The units to be operated when you set up and operate the virtual environment have been changed from the virtual server to the virtual server group. You can now define the information of a virtual server group in a file and register multiple virtual servers to a management unit in a batch.	*Virtual System Setup and Operation Guide*	*1.1.2*
Removed the restrictions on environments that can be built by using the Setup Wizard	Removed the restrictions on the environment that can be built by using the Setup Wizard. You can now unset up and set up any of the existing environments set up with a different functionality, by using the Setup Wizard.	*System Setup and Operation Guide*	*2.2.3*
Simplification of the procedure for removing an installed environment	Added functionality (`mngunsetup` command) for removing the system environment that is set up by using Management Server, thereby simplifying the removal procedure.	*System Setup and Operation Guide*	*4.1.37*
		Management Portal User Guide	*3.6*, *5.4*
		Command Reference Guide	*mngunsetup (Deleting the Management Server configuration environment)*

(2) Supporting standard and existing functionality

The following table describes the items that were changed to support standard and existing functionality:

Table D-2: Changes made for supporting standard and existing functionality

Item	Overview of changes	Reference manual	Reference
Supporting Servlet 3.0	Servlet 3.0 is now supported.	This manual	*Chapter 6*
Supporting EJB 3.1	EJB 3.1 is now supported.	*EJB Container Functionality Guide*	*Chapter 2*

Item	Overview of changes	Reference manual	Reference
Supporting JSF 2.1	JSF 2.1 is now supported.	This manual	*Chapter 3*
Supporting JSTL 1.2	JSTL 1.2 is now supported.	This manual	*Chapter 3*
Supporting CDI 1.0	CDI 1.0 is now supported.	*Common Container Functionality Guide*	*Chapter 9*
Using Portable Global JNDI names	You can now look up objects for which Portable Global JNDI names are used.	*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, 16.2.16, 16.2.17, 16.2.18, 16.2.20, 16.2.22, 19.1, 19.2.3, 37.2, 37.6.1, 37.6.2, 37.6.3*
Supporting JAX-RS 1.1	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 were changed for maintaining and improving reliability:

Table D-3: Changes for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference
Using TLSv1.2 for SSL/TLS communication	You can now use RSA BSAFE SSL-J to execute the SSL/TLS communication with a security protocol containing TLSv1.2.	*Security Management Guide*	*7.3*

(4) Maintaining and improving operation performance

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

Table D-4: Changes for maintaining and improving operation performance

Item	Overview of changes	Reference manual	Reference
Monitoring the total pending queues of the entire Web container	You can now monitor the total pending queues of the entire Web container when the total is output to the operation information.	*Operation, Monitoring, and Linkage Guide*	*Chapter 3*

Item	Overview of changes	Reference manual	Reference
Output of the trace based performance analysis for applications (user extended trace)	The trace based performance analysis used for analyzing the processing performance of user-developed applications can now be output without changing the applications.	*Maintenance and Migration Guide*	*Chapter 7*
Operations performed by using the user script in a virtual environment	The user-created script (user script) can now be executed on a virtual server at any time	*Virtual System Setup and Operation Guide*	*7.8*
Improving the management portal	Changes have been made so that the messages describing the procedure are now displayed on the following management portal windows: • Deploy the preference information window • Start window for the Web server, J2EE server, and SFO server • Batch start, batch restart, and startup windows for Web server cluster and J2EE server cluster	*Management Portal User Guide*	*10.11.1, 11.9.2, 11.10.2, 11.11.2, 11.11.4, 11.11.6, 11.12.2, 11.13.2, 11.13.4, 11.13.6*
Adding restart functionality for operation management functionality	You can now set the automatic restart in the operation management functionality (Management Server and Administration Agent). Due to the automatic restart functionality, it is now possible to continue operations even if an error occurs in the operation management functionality. The procedure for automatic start setting has also been changed.	*Operation, Monitoring, and Linkage Guide*	*2.4.1, 2.4.2, 2.6.3, 2.6.4*
		Command Reference Guide	*mngautorun (Set up/ canceling the set up of autostart and autorestart)*

(5) Other purposes

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

Table D−5: Changes due to other purposes

Item	Overview of changes	Reference manual	Reference
Changing the unit for switching log output files	You can now change log output files by date.	*Maintenance and Migration Guide*	*3.2.1*
Changing the Web server name	The name of the Web server included in Application Server has been changed to uCosminexus HTTP Server.	*HTTP Server User Guide*	--
Supporting a direct connection with the API (SOAP architecture) in BIG-IP	Direct connection is now supported by using APIs (SOAP architecture) in BIG-IP (load balancer). Note that the procedure for setting up the connection environment of the load balancer has been changed for using a direct connection through APIs.	*System Setup and Operation Guide*	*4.7.3, Appendix K*
		Virtual System Setup and Operation Guide	*2.1, Appendix C*
		Security Management Guide	*8.2, 8.4, 8.5, 8.6, 18.2, 18.3, 18.4*

Legend:
--: Reference the entire manual.

D.2 Main functionality changes in 08-70

(1) Simplifying implementation and setup

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

Table D-6: Changes made for simplifying implementation and setup

Item	Overview of changes	Reference manual	Reference
Improving the Management portal	The changes have been made to enable the user to set the property (settings of the Connector property file) for defining the resource adapter attributes and perform the connection test in the management portal window. Also, you can now use the Management portal window to upload J2EE applications (ear file and zip file) on Management Server.	*First Step Guide*	*3.5*
		Management Portal User Guide	--
Adding functionality for implicitly importing the import property for the `page/tag` directive	You can now use the functionality for implicitly importing the import property of the `page/tag` directive.	This manual	*2.3.7*
Support for automating the environment settings corresponding to the JP1 products in a virtual environment	The changes have been made so that when Application Server is set up on a virtual server, the environment settings of JP1 products can be automatically set for the virtual server by using the hook script.	*Virtual System Setup and Operation Guide*	*7.7.2*
Improving the Integrated user management functionality	When using a database in a user information repository, you can now connect to the database with the JDBC driver of database products. The database connection through the JDBC driver of Cosminexus DABroker Library is not supported anymore. You can now set the integrated user management functionality using the Easy Setup definition file and the management portal windows. The Active Directory now supports double byte characters such as Japanese language in DN.	*Security Management Guide*	*Chapter 5*, *14.3*
		Management Portal User Guide	*3.5*, *10.9.1*
Enhancing HTTP Server settings	You can now directly set the directive (settings of `httpsd.conf`) that defines the operation environment of HTTP Server using the Easy Setup definition file and the management portal windows.	*System Setup and Operation Guide*	*4.1.21*
		Management Portal User Guide	*10.10.1*
		Definition Reference Guide	*4.13*

Legend:
--: Reference the entire manual.

(2) Supporting standard and existing functionality

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

Table D-7: Changes made for supporting standard and existing functionality

Item	Overview of changes	Reference manual	Reference
Adding the items to be specified in `ejb-jar.xml`	You can now specify a class level interceptor and a method level interceptor in `ejb-jar.xml`.	*EJB Container Functionality Guide*	*2.15*
Supporting the parallel copy garbage collection	You can now select the parallel copy garbage collection.	*Definition Reference Guide*	*16.5*
Supporting the global transaction of the Inbound resource adapter conforming to the Connector 1.5 specifications	You can now use `Transacted Delivery` in resource adapters conforming to the Connector 1.5 specifications. This enables the participation of EIS invoking the Message-driven Bean in the global transaction.	*Common Container Functionality Guide*	*3.16.3*
Supporting MHP of a TP1 inbound adapter	You can now use MHP as the OpenTP1client that invokes Application Server by using the TP1 inbound adapter.	*Common Container Functionality Guide*	*Chapter 4*

Item	Overview of changes	Reference manual	Reference
Supporting the FTP inbound adapter of the `cjrarupdate` command	An FTP inbound adapter has been added to the resource adapters that can be upgraded by using 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 D-8: Changes for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference
Improving the database session failover functionality	The user can now select a mode that does not obtain the lock of the database in which the global session information is stored in a performance-centric system. Also, exclusive requests for references can now be defined without updating the database.	*Expansion Guide*	*Chapter 6*
Expansion of a process for the OutOfMemory handling functionality	A process for the OutOfMemory handling functionality has been added.	*Maintenance and Migration Guide*	*2.5.7*
		Definition Reference Guide	*16.2*
Adding the memory saving functionality for the Explicit heap used in an HTTP session	A functionality to minimize the amount of the Explicit heap memory used in the HTTP session has been added.	*Expansion Guide*	*8.11*

(4) Maintaining and improving operability

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

Table D-9: Changes with the purpose of maintaining and improving the operability

Item	Overview of changes	Reference manual	Reference
Supporting an user authentication using JP1 products in the virtual environment (handling cloud operations)	The administration and authentication of users using a virtual server manager can now be performed by using the authentication server of JP1 products when integrating JP1.	*Virtual System Setup and Operation Guide*	*1.2.2*, *Chapter 3*, *Chapter 4*, *Chapter 5*, *Chapter 6*, *7.9*

(5) Other purposes

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

Table D-10: Changes for other purposes

Item	Overview of changes	Reference manual	Reference
Supporting a direct connection using APIs (REST Architecture) to the load balancing functionality	A direct connection using APIs (REST architecture) is now supported as a method to connect to the Load balancing functionality. ACOS (AX2500) has been added in the types of available load balancing functions.	*System Setup and Operation Guide*	*4.7.2*, *4.7.3*
		Virtual System Setup and Operation Guide	*2.1*
		Definition Reference Guide	*4.5*
Improving response timeout when collecting snapshot logs and collection targets	You can now stop the snapshot log collection (timeout) at a specified time. The contents collected as primary delivery data have been changed.	*Maintenance and Migration Guide*	*Appendix A*

D.3 Main functionality changes in 08-53

(1) Simplifying implementation and setup

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

Table D-11: Changes made for simplifying implementation and setup

Item	Overview of changes	Reference manual	Reference
Setting up a virtual environment supporting various hypervisors	You can now set up Application Server on a virtual server implemented by using various hypervisors. An environment in which multiple hypervisors co-exist is also supported now.	*Virtual System Setup and Operation Guide*	*Chapter 2*, *Chapter 3*, *Chapter 5*

(2) Supporting standard and existing functionality

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

Table D-12: Changes made for supporting the standard and existing functionality

Item	Overview of changes	Reference manual	Reference
Invocation from OpenTP1 supporting the transaction integration	You can now integrate transactions when the Message-driven Bean running on Application Server is invoked from OpenTP1	*Common Container Functionality Guide*	*Chapter 4*
JavaMail	The mail receiving functionality, which uses the APIs conforming to JavaMail 1.3 by integrating with the mail server conforming to POP3, is now available.	*Common Container Functionality Guide*	*Chapter 8*

(3) Maintaining and improving reliability

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

Table D-13: Changes made for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference
Enhancing the JavaVM troubleshooting functionality	You can now use the following functionality as the JavaVM troubleshooting functionality: • You can now change the operations when OutOfMemoryError occurs. • You can now set up an upper limit for the amount of the C heap allocated during the JIT compilation. • You can now set up the maximum thread count. • Output items of the extended verbosegc information have been extended.	*Maintenance and Migration Guide*	*Chapter 4*, *Chapter 5*, *Chapter 9*

(4) Maintaining and improving operability

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

Table D-14: Changes made for maintaining and improving operability

Item	Overview of changes	Reference manual	Reference
Supporting JP1/ITRM	JP1/ITRM, a product that uniformly manages the IT resources, is now supported.	*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 D-15: Changes for other purposes

Item	Overview of changes	Reference manual	Reference
Supporting Microsoft IIS 7.0 and Microsoft IIS 7.5	Microsoft IIS 7.0 and Microsoft IIS 7.5 are now supported as Web servers.	--	--
Supporting HiRDB Version 9 and SQL Server 2008	The following products are now supported as the database: • HiRDB Server Version 9 • HiRDB/Developer's Kit Version 9 • HiRDB/Run Time Version 9 • SQL Server 2008 Also, SQL Server JDBC Driver is now supported as the JDBC driver corresponding to SQL Server 2008.	*Common Container Functionality Guide*	*Chapter 3*

Legend:
--: Not applicable.

D.4 Main functionality changes in 08-50

(1) Enhancing ease of installation and construction

The following table describes the items that are changed with the purpose of enhancing ease of installation and construction:

Table D-16: Changes with the purpose of enhancing ease of installation and construction

Item	Overview of changes	Reference manual	Reference
Changing the tags with mandatory specification in `web.xml` at the Web service provider side	Changed the specification of the `listener` tag, `servlet` tag, and `servlet-mapping` tag in `web.xml` at the Web Service provider side from mandatory to optional.	*Definition Reference Guide*	*2.4*
Using the network resources of the logical server	Added a functionality for accessing the network resources and network drives on other hosts from the J2EE application.	*Operation, Monitoring, and Linkage Guide*	*1.2.3*, *5.2*, *5.7*
Simplifying the execution procedure of sample programs	Simplified the execution procedure of sample programs by providing some sample programs in the EAR format.	*First Step Guide*	*3.5*
		System Setup and Operation Guide	*Appendix M*
Improving the operation of the window of management portal	Changed the update interval of the window from "Do not update" to "Three seconds".	*Management Portal User Guide*	*7.4.1*
Improving the Setup Wizard completion window	Changes have been made to display the Easy Setup definition file and the HITACHI Connector Property file used for setup in the window displayed during completion of the Setup Wizard.	*System Setup and Operation Guide*	*2.2.6*
Setting up the virtual environment	Added the procedure for setting up Application Server on a virtual server implemented by using hypervisors.#	*Virtual System Setup and Operation Guide*	*Chapter 3*, *Chapter 5*

#
When setting up in the 08-50 mode, see *Appendix D Settings for Using the Virtual Server Manager in the 08-50 Mode* in the *uCosminexus Application Server Virtual System Setup and Operation Guide*.

(2) Supporting standard and existing functionality

The following table describes the items that are changed with the purpose of supporting standard and existing functionality:

Table D-17: Changes with the purpose of supporting standard and existing functionality

Item	Overview of changes	Reference manual	Reference
Supporting invocation from OpenTP1	Enabled the invocation of the Message-driven Bean operating on the Application Server from OpenTP1.	*Common Container Functionality Guide*	*Chapter 4*
Supporting JMS	Enabled the use of the Cosminexus JMS provider functionality compliant with JMS1.1 specifications.	*Common Container Functionality Guide*	*Chapter 7*
Supporting Java SE 6	Enabled the use of the Java SE 6 functionality.	*Maintenance and Migration Guide*	*5.5*, *5.8.1*
Supporting the use of generics	Enabled the use of generics in EJB.	*EJB Container Functionality Guide*	*4.2.19*

(3) Maintaining and improving reliability

The following table describes the items that are changed with the purpose of maintaining and improving reliability:

Table D-18: Changes with the purpose of maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference
Improving the usability of the Explicit Memory Management functionality	Enabled easy usage of the Explicit Memory Management functionality by using the Automatic Deployment setup file.	*System Design Guide*	*7.1.1*, *7.6.3*, *7.10.5*, *7.11.1*
		Expansion Guide	*Chapter 8*
Blocking the database session failover functionality in the URI unit	Enabled specification of requests that are to be set outside the scope of the database session failover functionality during the use of the functionality in the URI unit.	*Expansion Guide*	*5.6.1*
Monitoring failures in the virtual environment	Enabled the monitoring of virtual servers and detecting the occurrence of failures in a virtual system.	*Virtual System Setup and Operation Guide*	*Appendix D*

(4) Maintaining and improving operability

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

Table D-19: Changes with the purpose of maintaining and improving operability

Item	Overview of changes	Reference manual	Reference
Omitting the management user account	Enabled the omission of the input of the user login ID and password in the management portal, commands of the Management Server, and commands of the Smart Composer functionality.	*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, and setting up the Management Server)*, *mngsvrutil (Management commands of the Management Server)*, *8.3*, *cmx_admin_passwd (Setting the management user account of the Management Server)*
Operating the virtual environment	Added the procedure for executing batch startup, batch stop, scale-in, and scale-out of multiple virtual servers in a virtual system.#	*Virtual System Setup and Operation Guide*	*Chapter 4*, *Chapter 6*

#

When operating in the 08-50 mode, see *Appendix D Settings for Using the Virtual Server Manager in the 08-50 Mode* in the *uCosminexus Application Server Virtual System Setup and Operation Guide*.

(5) Other purposes

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

Table D-20: Changes for other purposes

Item	Overview of changes	Reference manual	Reference
Statistical functionality for unused objects within the Tenured area	Enabled the identification of only unused objects within the Tenured area.	*Maintenance and Migration Guide*	*9.8*
Base point object list output functionality for Tenured increment factors	Enabled the output of information of the object that acts as the base point of unused objects identified using the statistical functionality for unused objects within the Tenured area.		*9.9*
Class-wise statistical information analysis functionality	Enabled the output of class-wise statistical information in the CSV format.		*9.10*
Cluster node switching due to detection of excess auto restart frequency of the logical server	Enabled node switching when the logical server stops abnormally (when the auto restart frequency is exceeded, or when a failure is detected when the auto restart frequency is set to 0) in the case of cluster configuration in which Management Server is a target for monitoring node switching.	*Operation, Monitoring, and Linkage Guide*	*18.4.3*, *18.5.3*, *20.2.2*, *20.3.3*, *20.3.4*
Node switching system for the host unit management model	Enabled node switching for the host unit management model during system operation linked with cluster software.		*Chapter 20*
Supporting ACOS (AX2000, BS320)	Added ACOS (AX2000, BS320) in the types of available load balancers.	*System Setup and Operation Guide*	*4.7.2, 4.7.3, 4.7.5, 4.7.6, Appendix K, Appendix K. 2*
		Definition Reference Guide	*4.5*, *4.6.2*, *4.6.4*, *4.6.5*, *4.6.6*, *4.10.1*
Adding transaction attributes that can be specified in a Stateful Session Bean (SessionSynchronization) when performing transaction management in CMT	Changes have been made to specify `Supports`, `NotSupported`, and `Never` as transaction attributes in a Stateful Session Bean (SessionSynchronization) when performing transaction management in CMT.	*EJB Container Functionality Guide*	*2.7.3*
Forced termination of the Administration Agent during the occurrence of OutOfMemoryError	Enabled forced termination of the Administration Agent during the occurrence of OutOfMemoryError in JavaVM.	*Maintenance and Migration Guide*	*2.5.8*
Asynchronous parallel processing of threads	Enabled the implementation of the asynchronous timer processing and asynchronous thread processing using TimerManager and WorkManager.	*Expansion Guide*	*Chapter 10*

D.5 Main functionality changes in 08-00

This section describes the main functionality changes in Application Server 08-00. These changes are described below with reference to the purpose of change.

The description contents are as follows:

- The main functionality that have been changed in Application Server 08-00 and an overview of the changes is described below. For details about the functionality, check the *Reference*. The *Reference* column describes the main description locations of the functionality.
- *Cosminexus Application Server* has been omitted from the manual names described in the *Reference* column.

(1) Improvement of development productivity

The following table describes the items changed with the purpose of improving the development productivity:

Table D-21: Changes made with the purpose of improving the development productivity

Item	Overview of changes	Reference manual	Reference
Simplification of migration from other Application Server products	Enabled the use of the following functionality for smooth migration from other Application Server products: • Enabled the judgment of upper limit of the HTTP sessions through an exception. • Enabled the inhibition of occurrence of a translation error when the ID of JavaBeans is duplicate, and when the upper-case characters and lower-case characters are different in the attribute name of the custom tag and in the TLD definition.	This manual	*2.3*, *2.7.5*
Provision of `cosminexus.xml`	Enabled the start of J2EE applications without setting the properties after importing them into the J2EE server by describing the properties unique to the Cosminexus Application Server in `cosminexus.xml`.	*Common Container Functionality Guide*	*11.3*

(2) Support of standard functionality

The following table describes the items changed with the purpose of supporting the standard functionality:

Table D-22: Changes made with the purpose of supporting the standard functionality

Item	Overview of changes	Reference manual	Reference
Servlet 2.5 support	Supported Servlet 2.5.	This manual	*2.2*, *2.5.4*, *2.6*, *Chapter 6*
JSP 2.1 support	Supported JSP 2.1.	This manual	*2.3.1*, *2.3.3*, *2.5*, *2.6*, *Chapter 6*
JSP debug	Enabled the execution of JSP debugging in the development environment using MyEclipse. #	This manual	*2.4*
Storage of the tag library in the library JAR, and TLD mapping	Enabled the search of TLD files within the library JAR by the Web container during the start of the Web application, and their subsequent automatic mapping, when the tag libraries are stored in the library JAR.	This manual	*2.3.4*
Omission of `application.xml`	Enabled the omission of `application.xml` in a J2EE application.	*Common Container Functionality Guide*	*11.4*
Combined use of annotations and DDs	Enabled the combined use of annotations and DDs, and also enabled the update of annotation contents in the DD.	*Common Container Functionality Guide*	*12.5*
Conformance of annotations to Java EE 5 standard (default interceptor)	Enabled the storage of the default interceptor in the library JAR. Also enabled the execution of DI from the default interceptor.	*Common Container Functionality Guide*	*11.4*
Reference resolution of `@Resource`	Enabled the reference resolution of resources with `@Resource`.	*Common Container Functionality Guide*	*12.4*

Item	Overview of changes	Reference manual	Reference
JPA support	Supported JPA specifications.	*Common Container Functionality Guide*	*Chapter 5*, *Chapter 6*

#

In version 09-00 and later, you can use the JSP debug functionality in the development environment using WTP.

(3) Maintenance and improvement of reliability

The following table describes the items changed with the purpose of maintaining or improving reliability:

Table D-23: Changes made with the purpose of maintaining or improving reliability

Item	Overview of changes	Reference manual	Reference
Persistence of session information	Enabled the inheritance of session information of an HTTP session by saving the information in the database.	*Expansion Guide*	*Chapter 5*, *Chapter 6*
Inhibition of a full garbage collection	Enabled the inhibition of occurrence of a full garbage collection by deploying the objects responsible for the full garbage collection outside the Java heap.	*Expansion Guide*	*Chapter 8*
Client performance monitor	The time required for client processing can now be checked and analyzed.	--	--

Legend:

--: This functionality has been deleted in version 09-00.

(4) Maintenance and improvement of operability

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

Table D-24: Changes made with the purpose of maintaining and improving operability

Item	Overview of changes	Reference manual	Reference
Improving the operability of applications on the management portal	The server management commands and management portal can now be interoperated for application and resource operations.	*Management Portal User Guide*	*1.1.3*

(5) Other purposes

The following table describes the items changed with some other purpose:

Table D-25: Changes made with other purposes

Item	Overview of changes	Reference manual	Reference
Deletion if disabled HTTP Cookies	Enabled the deletion of disabled HTTP Cookies.	This manual	*2.7.4*
Failure detection in the Naming Service	Enabled prompt detection of the error by the EJB client, when a failure occurs in the Naming Service.	*Common Container Functionality Guide*	*2.9*
Connection failure detection timeout	Enabled the specification of the timeout period for a connection failure detection timeout.	*Common Container Functionality Guide*	*3.15.1*
Oracle11g support	Enabled the use of Oracle11g as a database.	*Common Container Functionality Guide*	*Chapter 3*
Scheduling of batch processing	Enabled the scheduling of execution of batch applications by CTM.	*Expansion Guide*	*Chapter 4*

Item	Overview of changes	Reference manual	Reference
Batch processing log	The retry frequency and retry interval can now be specified for the size and number of log files of the batch execution command and the failure of exclusive processing of the log.	*Definition Reference Guide*	*3.6*
snapshot log	Changed the collection contents of the snapshot log.	*Web Container Functionality Guide*	*Appendix A. 1*, *Appendix A.2*
Publication of protected area of method cancellation	Published the contents of protected area list that is outside the scope of method cancellation.	*Operation, Monitoring, and Linkage Guide*	*Appendix C*
Pre-statistical garbage collection selection functionality	Enabled the selection of whether or not to execute a garbage collection before the output of class-wise statistical information.	*Web Container Functionality Guide*	*9.7*
Tenuring distribution information output functionality of the Survivor area.	Enabled the output of tenuring distribution information of Java objects of the Survivor area to the Hitachi JavaVM log file.	*Web Container Functionality Guide*	*9.11*
Finalize retention cancellation functionality	Enabled the cancellation of retention of the finalize processing of JavaVM after monitoring its status.	--	--
Change of the maximum heap size of server management commands	Changed the maximum heap size used by server management commands.	*Definition Reference Guide*	*5.2*, *5.3*
Action for cases when un-recommended display names are specified	Provided the output of messages when un-recommended display names are specified in J2EE applications.	*Messages*	*KDJE42374-W*

Legend:

--: This functionality is deleted in 09-00.

E. 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

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X