uCosminexus Application Server

Security Management Guide

3020-3-Y09-10(E)

■ Relevant program products

See the manual *uCosminexus Application Server Overview*.

■ Export restrictions

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

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

■ Trademarks

■ Microsoft product name abbreviations

This manual uses the following abbreviations for Microsoft product names.

Abbreviation		Full name or meaning
Active Directory		Microsoft(R) Active Directory(R)
Microsoft IIS	Microsoft IIS 7.0	Microsoft(R) Internet Information Services 7.0
	Microsoft IIS 7.5	Microsoft(R) Internet Information Services 7.5
SQL Server	SQL Server 2005	Microsoft(R) SQL Server 2005
	SQL Server 2008	Microsoft(R) SQL Server 2008
		Microsoft(R) SQL Server 2008 R2
	SQL Server 2012	Microsoft(R) SQL Server 2012
JDBC driver for SQL Server	SQL Server JDBC Driver	Microsoft(R) SQL Server JDBC Driver 3.0
		Microsoft(R) JDBC Driver 4.0 for SQL Server

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

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

■ Issued

Aug 2013: 3020-3-Y09-10(E)

■ Copyright

Summary of amendments

The following table lists changes in the manual 3020-3-Y09-10(E) for uCosminexus Application Server 09-50, uCosminexus Application Server(64) 09-50, uCosminexus Client 09-50, uCosminexus Developer 09-50, uCosminexus Service Architect 09-50, uCosminexus Service Platform 09-50, uCosminexus Service Platform(64) 09-50 and product changes related to the manual:

Changes	Location
Changes were made to the notes about disabling J2EE server runtime protection.	*2.2.5*
`WebPasswordJDBCLoginModule` now supports SQL Server 2012 databases. The JDBC driver for SQL Server was changed to the SQL Server JDBC Driver that is applicable to all SQL Server versions.	*5.3.5*
Load balancer BIG-IP v11 was added.	*8.2*, *8.5*, *18.3*, *18.4*
Notes from the *Release Notes* were moved.	*2.4*, *3.2*, *5.3.3*, *5.6.1*, *5.17.1*, *6.2.1*, *7.2.3*, *8.4*, Appendix *B*

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

Preface

For information you need to know before reading this manual, see the *uCosminexus Application Server Overview*.

Non-supported functionality

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

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

Non-supported compatibility functionality

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

- Basic mode
- Check of JSP source compliance (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

Chapter 1. Application Server Functionality

This chapter describes the types and purposes of the application server's functions. It shows the associations between the functions and the manuals, and explains the modifications to the functions in this server version.

1.1 Classifications of functionality

This application server can be used for such purposes as configuring application execution environments on J2EE servers that are compatible with Java EE 6 and developing applications that run in those execution environments. It offers various types of functionality including Java EE functionality and extended functionality specific to application servers. By selecting and using functions that are appropriate to your purposes, you can build and operate a highly reliable and capable system.

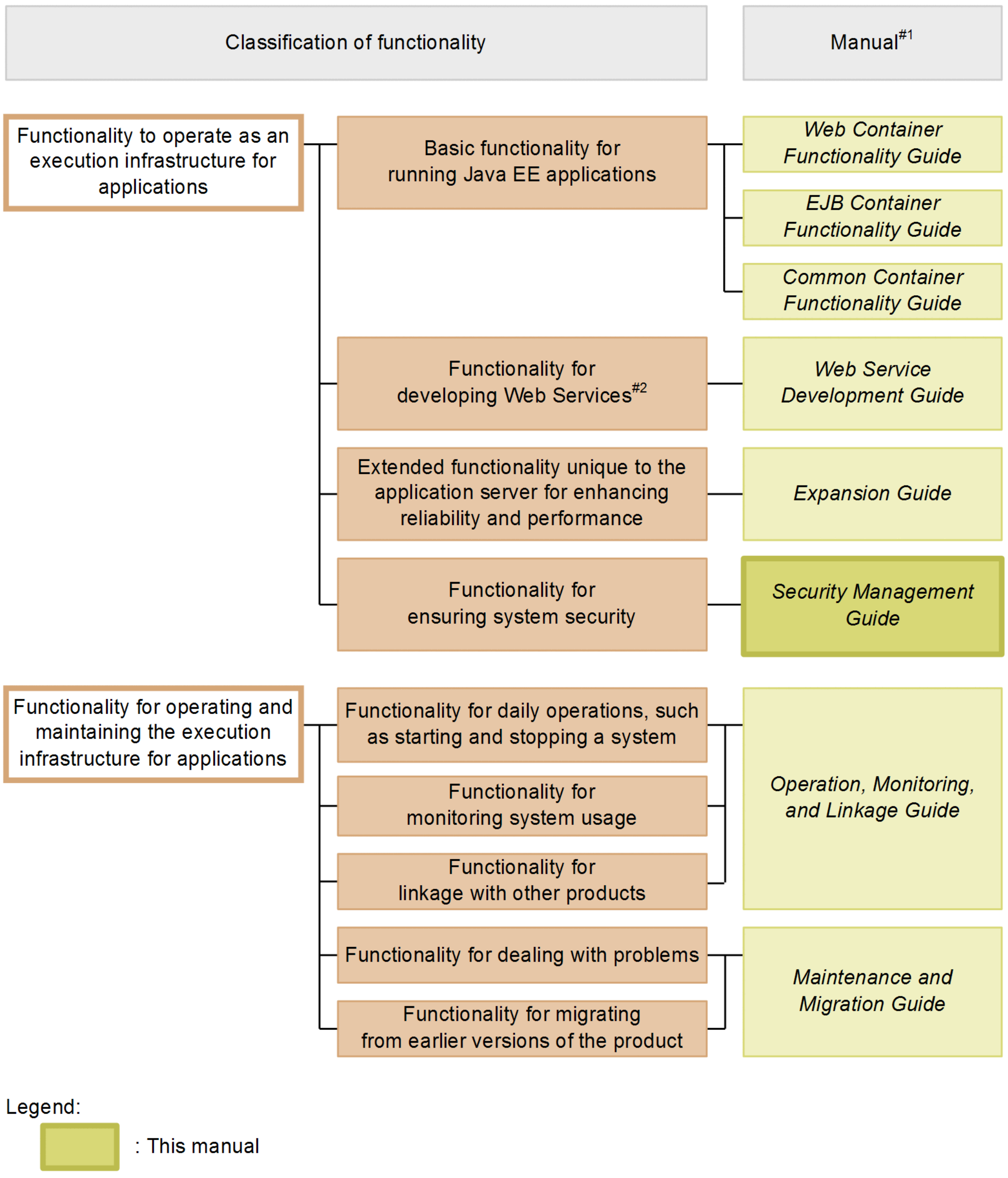
The application server has two major types of functionality:

- Functionality as an execution infrastructure for applications
- Functionality for operating and maintaining the execution infrastructure for applications

These two types of functionality can be further subdivided according to their positioning and uses. The application server manuals are classified according to the types of functionality they cover.

The following figure shows the types of functionality provided by the application server and the associated manuals.

Figure 1-1: Types of functionality provided by the application server and the associated manuals


Classification of functionality
Manual#1
Functionality to operate as an execution infrastructure for applications
Basic functionality for running Java EE applications
Web Container Functionality Guide
EJB Container Functionality Guide
Common Container Functionality Guide
Functionality for developing Web Services#2
Web Service Development Guide
Extended functionality unique to the application server for enhancing reliability and performance
Expansion Guide
Functionality for ensuring system security
Security Management Guide
Functionality for operating and maintaining the execution infrastructure for applications
Functionality for daily operations, such as starting and stopping a system
Functionality for monitoring system usage
Functionality for linkage with other products
Operation, Monitoring, and Linkage Guide
Functionality for dealing with problems
Functionality for migrating from earlier versions of the product
Maintenance and Migration Guide
Legend:
: This manual

#1

The words *uCosminexus Application Server* are omitted from the manual titles.

#2

The application server enables you to run SOAP Web Service and RESTful Web Service. In addition to the *uCosminexus Application Server Web Service Development Guide*, also see the following manual if necessary:

For details about XML processing:

- *XML Processor User's Guide*

The following section describes the types of functionality covered by the manuals.

1.1.1 Functionality for an application execution infrastructure

Functionality for an application execution infrastructure refers to the basic functionality for executing online and batch jobs implemented as applications. Select the appropriate functions to meet the purpose and requirements of the system.

Before building a system or developing an application, determine the basic functionality to be used.

Such functionality includes the following:

(1) Basic functionality for running applications (basic development functionality)

This functionality provides the capability to run applications (J2EE applications). It mainly comprises J2EE server functionality.

The application server includes a J2EE server that complies with the Java EE 6 specifications. The J2EE server offers functionality that complies with not only the standard specifications but also the unique functionality of the application server.

The basic development functionality can be further divided into three types according to the way the J2EE application uses this functionality. The functional description manuals for the application server are associated with these three types of functionality.

These three types of functionality are outlined below.

■ Functionality (Web container) for running Web applications

This functionality includes Web container functionality to provide an execution infrastructure for Web applications. It also includes functionality implemented by linking the Web container to a Web server.

■ Functionality (EJB container) for running an enterprise bean

This functionality includes EJB container functionality to provide an execution infrastructure for enterprise beans. It also includes EJB client functionality for calling an enterprise bean.

■ Functionality (container common functionality) for both Web applications and enterprise beans

This is functionality that can be used by both a Web application running in a Web container and an enterprise bean running in an EJB container.

(2) Functionality for developing Web Services

This functionality provides environments for running and developing Web Services.

The application server provides the following engines:

- JAX-WS engine that binds JAX-WS-compliant SOAP messages
- JAX-RS engine that binds JAX-RS-compliant RESTful HTTP messages

(3) Extended functionality unique to the application server for enhancing reliability and performance (extended functionality)

This functionality refers to extended functionality that is unique to the application server. It includes functionality implemented through the use of non-J2EE server processes such as batch server, CTM, and database processes.

The application server has various types of extended functionality for enhancing the reliability of the system and ensuring its stable operation. For example, it has an extended functionality for running non-J2EE applications (batch applications) on the Java platform.

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

The intention of this functionality is to ensure the security of a system built around the application server. It includes authentication for preventing unauthorized user access and encryption for preventing information leakage on communication paths.

1.1.2 Functionality for operating and maintaining the execution infrastructure for applications

The application server provides functionality to facilitate the efficient operation and maintenance of the execution infrastructure for applications. Use the provided functions as necessary after the start of system operation. Note that certain functionality requires configuration of settings or implementation of applications prior to system operation.

This functionality includes the following:

(1) Functionality for daily operations such as starting and stopping a system (operation functionality)

This is functionality for daily operations such as starting and stopping a system and starting, stopping, and replacing applications.

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

This functionality includes monitoring of system operation and resource shortages. It also includes the output of system operation history and any other information necessary for audit activities.

(3) Functionality for linkage with other products (linkage functionality)

This is functionality that enables the application server to be linked with other products such as JP1 and cluster software.

(4) Functionality for dealing with problems (maintenance functionality)

This functionality is for troubleshooting, and includes the output of reference information that is necessary for troubleshooting.

(5) Functionality for migrating from earlier versions of the product (migration functionality)

This functionality is designed to facilitate migration from earlier versions of the application server to the latest version.

(6) Functionality for achieving compatibility with earlier versions of the product (compatibility functionality)

This is functionality designed to provide compatibility with earlier versions of the application server. For this purpose, we recommend you migrate to a version of the application server that supports the recommended functionality.

1.1.3 Functionality and associated manuals

The functional description manuals for the application server are divided according to the types of functionality.

The table below lists the types of functionality provided by the application server, and the associated manuals.

Table 1-1: Types of functionality and associated manuals

Classification	Functionality	Manual[#1]
Basic development functionality	Web container	*Web Container Functionality Guide*
	Use of JSF and JSTL	
	Web server integration	
	In-process HTTP server	
	Servlet and JSP implementation	
	EJB container	*EJB Container Functionality Guide*
	EJB client	

Classification	Functionality	Manual[#1]
	Notes on enterprise bean implementation	
	Naming management	*Common Container Functionality Guide*
	Resource connection and transaction management	
	Application server call from OpenTP1 (TP1 inbound integrated function)	
	Use of JPA on an application server	
	Cosminexus JPA provider	
	Cosminexus JMS provider	
	Use of JavaMail	
	Use of CDI on an application server	
	Use of Bean Validation on an application server	
	Application attribute management	
	Use of annotations	
	Format and deployment of J2EE applications	
	Container extension library	
Extended functionality	Application execution with a batch server	*Expansion Guide*
	Request scheduling and load balancing with CTM	
	Batch application scheduling	
	Session information transfer between J2EE servers (session failover functionality)	
	Database session failover functionality	
	EADs session failover functionality	
	Full garbage collection prevention with explicit-management heap functionality	
	Application user log output	
	Asynchronous parallel processing of threads	
Security management functionality	Authentication through integrated user management	*Security Management Guide*[#2]

Classification	Functionality	Manual[#1]
	Authentication through application setup	
	Use of TLSv1.2 for SSL/TLS communication	
	Use of an API for direct access to load balancers, and control of load balancers using operation management functionality	
Operation functionality	System start and stop	*Operation, Monitoring, and Linkage Guide*
	J2EE application operation	
Monitoring functionality	Monitoring of operation information (collection of operation information)	
	Resource shortage monitoring	
	Audit log output functionality	
	Database audit trail linkage functionality	
	Output of operation information via operation management commands	
	Automatic process execution through management event notification and management actions	
	CTM operation statistics collection	
	Console log output	
Linkage functionality	Operation of systems linked using JP1	
	System configuration definition and system management (linkage with JP1/IM-CM)	
	Centralized monitoring of the system (linkage with JP1/IM)	
	Automatic operation of the system by job (linkage with JP1/AJS)	
	Audit log collection and centralized management (linkage with JP1/NETM/Audit)	
	Linkage with cluster software	
	1-to-1 node switching system (linkage with cluster software)	

Classification	Functionality	Manual[#1]
	Mutual node switching system (linkage with cluster software)	
	N-to-1 recovery system (linkage with cluster software)	
	Node switching system for per-host management models (linkage with cluster software)	
Maintenance functionality	Troubleshooting functionality	*Maintenance and Migration Guide*
	Performance analysis with performance analysis traces	
	The product's JavaVM functionality (abbreviated hereafter to *JavaVM*)	
Migration functionality	Migration from earlier versions of the application server	
	Migration to versions of the application server with the recommended functionality	

#1: The phrase *uCosminexus Application Server* is omitted from the manual titles.

#2: *Security Management Guide* refers to this manual.

1.2 Functionality and associated system purposes

You should select the functionality of the application server in accordance with the purpose of the system that you wish to build and operate.

This section describes the system purposes for which the application server's security functions should be used. Different types of functionality are available for different systems, as shown below.

- Reliability

  This functionality is available for systems that require a high level of reliability.

  It includes functions aimed at increasing availability and fault tolerance and enhancing security through user authentication.

- Performance

  This functionality is available for performance-focused systems.

  It includes functions designed to provide system performance tuning.

- Operation and maintenance

  This functionality is aimed at facilitating efficient operation and maintenance.

- Expandability

  This functionality is designed to facilitate expanding or shrinking the system size and to flexibly accommodate configuration changes.

- Other

  This functionality is included to accommodate other needs.

The application server's security functionality includes Java EE functionality and extended functionality unique to the application server. When selecting functions, check their compliance with the Java EE standard as necessary.

1.2.1 Authentication functionality

The table below shows the types of authentication functionality. Select the functionality that best suits the purpose of your system. For functional details, see the relevant information.

Table 1-2: Authentication functionality and associated system purposes

Functionality	System purpose					Compliance with Java EE standard		Relevant information
	Rel.	Per.	Op. and maint.	Expand	Other	Std.	Extd.	
Integrated user management	S	--	S	--	--	S	S	Chapter 5
Authentication by application setup	S	--	--	--	--	S	S	Chapter 6

Legend:

Rel.: Reliability

Per.: Performance

Op. and maint.: Operation and maintenance

Expand.: Expandability

Std.: Standard

Extd.: Extended

S: Supported

--: Not supported

Note: An *S* entered (in the same row) in both the *Standard* and *Extended* columns below *Compliance with Java EE standard* indicates cases where Java EE functionality has been extended to create functionality that is unique to the application server. An *S* entered in the *Extended* column next to a blank *Standard* column indicates cases where another functionality has been extended to create functionality that is unique to the application server.

1.2.2 Encryption functionality

The table below shows the application server's encryption functionality. Select the functionality that best suits the purpose of your system. For functional details, see the relevant information.

Table 1-3: Encryption functionality and associated system purposes

Functionality	System purpose					Compliance with Java EE standard		Relevant information
	Rel.	Per.	Op. and maint.	Expand	Other	Std.	Extd.	
SSL/TLS encryption of authentication information and data	S	--	--	--	--	S	S	Manual *HTTP Server User Guide* Chapter 7

Legend:

Rel.: Reliability

Per.: Performance

Op. and maint.: Operation and maintenance

Expand.: Expandability

Std.: Standard

Extd.: Extended

S: Supported

--: Not supported

Note 1: An *S* entered (in the same row) in the same row in both the *Standard* and *Extended* columns below *Compliance with Java EE standard* indicates cases where Java EE functionality has been extended to create functionality that is unique to the application server. An *S* entered in the *Extended* column next to a blank *Standard* column indicates cases where another functionality has been extended to create functionality that is unique to the application server.

Note 2: The phrase *uCosminexus V9 Application Server* is omitted from the manual title.

1.2.3 Invalid processing prevention functionality

The table below shows the application server's invalid processing prevention functionality. Select the functionality that best suits the purpose of your system. For functional details, see the relevant information.

Table 1-4: Invalid processing prevention functionality and associated system purposes

Functionality	System purpose					Compliance with Java EE standard		Relevant information
	Rel.	Per.	Op. and maint.	Expand	Other	Std.	Extd.	
Web container execution-time protection with Security Manager	S	--	--	--	--	S	S	2.2.5

Legend:

Rel.: Reliability

Per.: Performance

Op. and maint.: Operation and maintenance

Expand.: Expandability

Std.: Standard

Extd.: Extended

S: Supported

--: Not supported

Note: An *S* entered (in the same row) in both the *Standard* and *Extended* columns below *Compliance with Java EE standard* indicates cases where Java EE functionality has been extended to create functionality that is unique to the application server. An *S* entered in the *Extended* column next to a blank *Standard* column indicates cases where another functionality has been extended to create functionality that is unique to the application server.

1.2.4 Other functionality

The table below shows the application server's functionality for complying with requests for secure communication with other programs via linkage. Select the functionality that best suits the purpose of your system. For functional details, see the relevant information.

Table 1-5: Functionality for complying with requests for secure communication with other programs via linkage and associated system purposes

Functionality	System purpose					Compliance with Java EE standard		Relevant information
	Rel.	Per.	Op. and maint.	Expand	Other	Std.	Extd.	
Use of an API for direct access to load balancers, and control of load balancers using the operation management functionality	--	--	--	--	S	--	--	Chapter 8

Legend:

Rel.: Reliability

Per.: Performance

Op. and maint.: Operation and maintenance

Expand.: Expandability

Std.: Standard

Extd.: Extended

S: Supported

--: Not supported

1.3 Format of functional descriptions in this manual

This section describes the format of functional descriptions in this manual. It also gives an example table showing the parts of that format.

1.3.1 Parts of the descriptions

The functional descriptions in this manual are divided into the five parts described below. You can select and read any of these parts according to your purpose in referencing this manual.

- Description

  Functional description. Describes the purpose, features, and mechanism of the functionality. Read this part if you want to obtain an overview of the functionality.

- Implementation

  Describes how to code the program and how to create DD. Read this part if you want to develop an application.

- Setup

  Describes how to configure the properties necessary for system creation. Read this part if you want to create a system.

- Operation

  Operation method description. Describes the operation procedure and gives an example of executing the commands to be used. Read this part if you want to operate the system.

- Precautions

  Provides general precautions that should be observed when using the functionality. Read this part without fail.

1.3.2 Parts of the functional descriptions - example table

The following chapters contain tables showing the parts of the functional description. The title of each table is either *Organization of this chapter* or *Organization of this section*.

Below is an example table showing the parts of the functional description.

Example table showing the parts of the functional description

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

Part	Title	Relevant information
Description	What is the XX functionality?	X.1
Implementation	Application implementation	X.2
	Definitions in DD and `cosminexus.xml`#	X.3
Setup	Setup in execution environment	X.4
Operation	Operation using the XX functionality	X.5
Precautions	Precautions for using the XX functionality	X.6

#: For details about `cosminexus.xml`, see *11. Application Attribute Management* in the *uCosminexus Application Server Common Container Functionality Guide*.

Hint:

Configuring the properties of an application that does not contain cosminexus.xml

If an application does not contain `cosminexus.xml`, configure or change its properties after importing it into the execution environment. After configuration, you can change the properties in the execution environment.

To set up the application in the execution environment, use server management commands and the attributes file. For details about their use, see *3.5.2 Procedure for configuring the J2EE application properties*, in the *uCosminexus Application Server Application Setup Guide*.

The tags specified in the attributes file correspond to the DD file or `cosminexus.xml`. For details about their correspondence, see *2. Cosminexus Application Attributes File (cosminexus.xml)*, in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

Note that the properties to be configured in each attributes file can also be configured in the HITACHI Application Integrated Property File.

1.4 Major functional changes in Application Server 09-50

This section describes the major functional changes in Application Server 09-50, organized by the purpose for each modification.

- This section outlines the major functional changes in Application Server 09-50. For functional details, see the relevant information. The columns *Reference manual* and *Relevant information* indicate where to find relevant information about functionality.
- The words *uCosminexus Application Server* are omitted from the manual titles listed in the *Reference manual* column.

(1) Improving development productivity

The following table outlines the changes made to improve development productivity.

Table 1-6: Changes made to improve development productivity

Item	Overview of changes	Reference manual	Relevant information
Simplifying setup of Eclipse	The GUI can now be used to set up an Eclipse environment.	*Application Development Guide*	*1.1.5, 2.4*
Support for debugging using user extended performance analysis traces	User extended performance analysis trace configuration files can now be created in the development environment.	*Application Development Guide*	*1.1.3, 6.5*

(2) Implementing standard and existing functionality

The table below outlines the changes made to enable implementation of standard and existing functionality.

Table 1-7: Changes made to enable implementation of standard and existing functionality

Item	Overview of changes	Reference manual	Relevant information
Support for JDBC 4.0	DB Connector now supports HiRDB Type4 JDBC Driver and SQL Server JDBC drivers that comply with the JDBC 4.0 specifications.	*Common Container Functionality Guide*	*3.6.3*
Easing of portable global JNDI naming rules	Characters permitted for portable global JNDI names have been added.	*Common Container Functionality Guide*	*2.4.3*

Item	Overview of changes	Reference manual	Relevant information
Support for Servlet 3.0 specifications	Servlet 2.5 and earlier versions can now rename HTTP cookies and URL path parameters from Servlet 3.0.	*Web Container Functionality Guide*	*2.7*
Addition of applications that can link with Bean Validation	CDI and user applications can now use Bean Validation.	*Common Container Functionality Guide*	Chapter *10*
Support for JavaMail	Email sending and receiving functionality that uses JavaMail 1.4-compliant API functions is now supported.	*Common Container Functionality Guide*	Chapter *8*
Addition of OSs that are supported by the `javacore` command	The `javacore` command can now be used to obtain Windows thread dumps.	*Command Reference Guide*	*javacore (obtains thread dumps (Windows))*

(3) Maintaining and enhancing reliability

The table below outlines the changes made to maintain and enhance reliability.

Table 1-8: Changes made to maintain and enhancing reliability

Item	Overview of changes	Reference manual	Relevant information
Avoiding a shortage of space in code cache areas	A shortage of space in code cache areas can now be avoided by checking the size of a code cache area being used by the system and then changing the threshold values before a shortage of space occurs in the area.	*System Design Guide*	*7.1.2*
		Maintenance and Migration Guide	*5.7.2, 5.7.3*
		Definition Reference Guide	*16.1, 16.2, 16.4*
Support for efficient application of the Explicit Memory Management functionality	Functions for controlling objects that are moved to the Explicit heap have been added to reduce the automatic release processing time and apply the Explicit Memory Management functionality efficiently: • Function for controlling the moving of objects to Explicit memory blocks • Function for specifying the classes that are not subject to the Explicit Memory Management functionality • Output of object release rate information to 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*

Item	Overview of changes	Reference manual	Relevant information
Extension of output range for statistical information by class	Referential relationships based on the `static` field can now be output to extended thread dumps that contain statistical information by class.	*Maintenance and Migration Guide*	*9.6*

(4) Maintaining and enhancing availability

The table below outlines the changes made to maintain and enhance availability.

Table 1-9: Changes made to maintain and enhancing availability

Item	Overview of changes	Reference manual	Relevant information
Support for the EADs session failover functionality	Functionality that links with EADs and achieves session failover is now supported.	*Expansion Guide*	Chapters *5* and *7*
Operations by using WAR	WAR applications that consist of only WAR files can now be deployed on J2EE servers.	*Web Container Functionality Guide*	*2.2.1*
		Common Container Functionality Guide	*13.9*
		Command Reference Guide	*cjimportwar (imports WAR applications)*
Synchronously starting and stopping the operation management functionality	An option for synchronously starting and stopping the operation management functionality (Management Server and Administration Agent) has been added.	*Operation, Monitoring, and Linkage Guide*	*2.6.1, 2.6.2, 2.6.3, 2.6.4*
		Command Reference Guide	*adminagentctl (starts and stops Administration Agent), mngautorun (configures and cancels the configuration of automatic start and automatic restart), mngsvrctl (starts, stops, and sets up Management Server)*

Item	Overview of changes	Reference manual	Relevant information
Forced release of Explicit memory blocks by using the Explicit Memory Management functionality	The `javagc` command can now be used to release the Explicit memory blocks at any time.	*Expansion Guide*	*8.6.1, 8.9*
		Command Reference Guide	*javagc (forcibly performs garbage collection)*

(5) Other purposes

The table below outlines changes made for other purposes.

Table 1-10: Changes made for other purposes

Item	Overview of changes	Reference manual	Relevant information
Acquiring definition information	The `snapshot` (snapshot log collection) command can now be used to collect only definition files.	*Maintenance and Migration Guide*	*2.3*
		Command Reference Guide	*snapshotlog (collects snapshot logs)*
Output of logs by the `cjenvsetup` command	Information about the execution of Component Container Administrator setup (`cjenvsetup` command) is now output to the message logs.	*System Setup and Operation Guide*	*4.1.4*
		Maintenance and Migration Guide	*4.20*
		Command Reference Guide	*cjenvsetup (Component Container Administrator setup)*
Support for BIG-IP v11	BIG-IP v11, a type of load balancer, is now supported.	*System Setup and Operation Guide*	*4.7.2*
Output of CPU time to the event logs of the Explicit Memory Management functionality	The CPU time required for releasing Explicit memory blocks is now output to the event logs of the Explicit Memory Management functionality.	*Maintenance and Migration Guide*	*5.11.3*

Item	Overview of changes	Reference manual	Relevant information
Functional enhancement of user extended performance analysis traces	The following functions associated with user extended performance analysis traces have been added: • Trace targets can now be specified in units of packages and classes, in addition to units of methods. • The available range of event IDs has been expanded. • The limitation on the number of lines that can be specified in the user extended performance analysis trace configuration file has been eased. • The trace collection level can now be specified in the user extended performance analysis trace configuration file.	*Maintenance and Migration Guide*	*7.5.2, 7.5.3, 8.28.1*
Improvement of information analysis when asynchronous Session Bean calls are used	The source and target requests can now be matched by using the root application information in PRF traces.	*EJB Container Functionality Guide*	*2.17.3*

Chapter

2. Security Management with the Application Server

This chapter describes the functionality and methods for managing security with the application server. Based on the contents of this chapter, determine which functionality or method to use in order to establish the desired grade or level of security.

2.1 Organization of this chapter

The application server offers the functionality to manage system security. By using this functionality appropriately for your purpose, you can build and operate a system that meets your security needs. The table below shows how the chapter is organized.

Table 2-1: Organization of this chapter (Security Management with the Application Server)

Part	Title	Relevant information
Description	Measures for ensuring security	*2.2*
	Details about the methods and functionality for ensuring security	*2.3*
Notes	Notes about using the methods and functionality for ensuring security	*2.4*

Note: This chapter does not include information on implementation, setup, or operation.

2.2 Measures for ensuring security

To ensure security with the application server, you should take the measures outlined in the following sections:

- Realizing a system configuration that will ensure security
- Operating the system securely
- Preventing unauthorized users from accessing the system
- Ensuring communication path security
- Preventing invalid processing
- Taking other actions

2.2.1 Realizing a system configuration that will ensure security

Properly install security hardware or software on the system to prevent unauthorized access to the system from the outside.

By using a firewall, you can control access between the external and internal networks. You can prevent unauthorized access from the external network by pre-specifying the clients to be granted access to the system and then enabling or disabling communication according to the established rules. In addition, by using intrusion detection system (IDS), you can monitor the communication lines and, based on the communication pattern, detect and prevent unauthorized access.

If you deploy reverse proxy server, you can prevent malicious clients from directly accessing a Web server that contains important content. The reverse proxy server can receive requests from clients and access the Web server.

If communication path security is ensured through the use of encryption, SSL accelerator can handle the encryption and decryption processes, avoiding placing any load on the Web server and application server.

2.2.2 Operating the system securely

You can ensure security by operating properly the system after you build it.

Before the system operates, determine which users should be able to operate it, what information should be managed, and how the system should be physically arranged.

To operate the system properly in accordance with your wishes, prepare procedure manuals as necessary and then check whether the system runs correctly.

2.2.3 Preventing unauthorized users from accessing the system (authentication functionality)

To safely manage information handled by the system and ensure security, you should prevent unauthorized users from accessing the system. Authentication functionality is effective for this purpose.

The application server offers the following authentication functionality:

- User login authentication through the integrated user management framework

  This functionality uses integrated management of information about users who log into the system to enable multiple applications to be accessed with a single login.

- Web container-based authentication by `<security-constraint>` element setup

  This functionality uses a Web container to allow only authorized users to access a given application. Information for authentication is defined by using the `<security-constraint>` element in the DD file (`web.xml`).

- EJB container-based authentication by `<security-identity>` element setup

  This functionality uses an EJB container to allow only authorized users to access a given application. Information for authentication is defined by using the `<security-identity>` element in the DD file (`ejb-jar.xml`) or `cosminexus.xml`.

  The method of execution, specified using the `<method-permission>` element, can be controlled depending on the role assigned to each user.

  If you implement an EJB client application by using the application server's API, you can authenticate access from the EJB client application.

2.2.4 Ensuring communication path security (encryption functionality)

Information encryption is an effective way to prevent information leakage on the communication path between the clients and the application server.

The application server offers the following encryption functionality:

- Encryption with SSL

  Using a Web server and J2EE server, this functionality encrypts information passed along the communication path.

- SOAP message encryption with the Web Services Security functionality

  Web Services Security allows you to encrypt SOAP messages sent and received via the Web service and assign an XML signature to SOAP messages.

2.2.5 Preventing invalid processing

To prevent invalid processing from being executed on the J2EE server, you can use J2EE server runtime protection provided by the *SecurityManager* functionality of Java SE.

J2EE server runtime protection enables prevention of the following phenomena:

- The entire J2EE server terminates abnormally due to an invalid servlet or EJB that internally issues `System.exit()`.
- The J2EE server executes abnormally due to an invalid servlet or EJB that rewrites system properties without permission.

Note that J2EE server runtime protection is enabled by default. If you do not wish to use it, specify the `-nosecurity` option in the `cjstartsv` command that starts the J2EE server.

If you disable J2EE server runtime protection, the `setSecurityManager` method of `java.lang.System` cannot be used. If an attempt is made to use this method, J2EE server operations might be affected adversely. Note also that dynamic class loading cannot be used when EJB is called from a corresponding J2EE server process.

2.2.6 Taking other actions

If secure communication is requested for linkage with other programs, configure the settings to meet the requirements of the programs.

2.3 Details about the methods and functionality for ensuring security

The information listed in the table below provides relevant details about the methods and functionality for the security measures described in this chapter.

Table 2-2: Details about the methods and functionality for ensuring security

Measure	Functionality	Relevant information
Realizing a system configuration that will ensure security	--	Chapter 3, Chapter 4
Operating the system securely	--	Chapter 4
Preventing unauthorized users from accessing the system (authentication functionality)	User login authentication through the integrated user management framework	Chapter 5
	Web container-based authentication with DD settings	6.2
	Authentication with security identities	6.3
Ensuring communication path security (encryption functionality)	SSL encryption (on the Web server)	Manual *HTTP Server User Guide*
	SSL/TLS encryption of authentication information and data	Chapter 7
Preventing invalid processing	Web container runtime protection provided by SecurityManager	2.2.5
Taking other actions	Use of an API for direct access to load balancers, and control of load balancers using the operation management functionality	Chapter 8

--: Not available

2.4 Notes about using the methods and functionality for ensuring security

(1) About certificates

The `cacerts` certificates file that is provided in Application Server does not include the certificates. If you need certificates, obtain them and then import them. For details about importing certificates, see the following sites:

Windows:

`http://docs.oracle.com/javase/jp/6/technotes/tools/windows/keytool.html`

UNIX:

`http://docs.oracle.com/javase/jp/6/technotes/tools/solaris/keytool.html`

Chapter

3. System Configurations for Ensuring Security

This chapter describes various system configurations that can be used to ensure security in J2EE application execution infrastructures. Using examples of system configurations, this chapter provides information about deploying firewalls to suit different types of components and deploying reverse proxies in a DMZ.

3.1 Organization of this chapter

This chapter describes various system configurations for ensuring security. The table below shows how the chapter is organized.

Table 3-1: Organization of this chapter (System Configurations for Ensuring Security)

Part	Title	Relevant information
Description	System configurations using a firewall	3.2
	Deployment of reverse proxies in a DMZ	3.3

Note: This chapter does not include information on implementation, setup, operation, or precautions.

3.2 System configurations using a firewall

This section describes system configurations that employ a firewall to ensure security. It shows the positions of firewalls to suit different types of components that serve as access points. For information about other security concepts, see *4. Considerations in the Design of a Secure System*.

3.2.1 Deployment of a firewall for servlets and JSPs

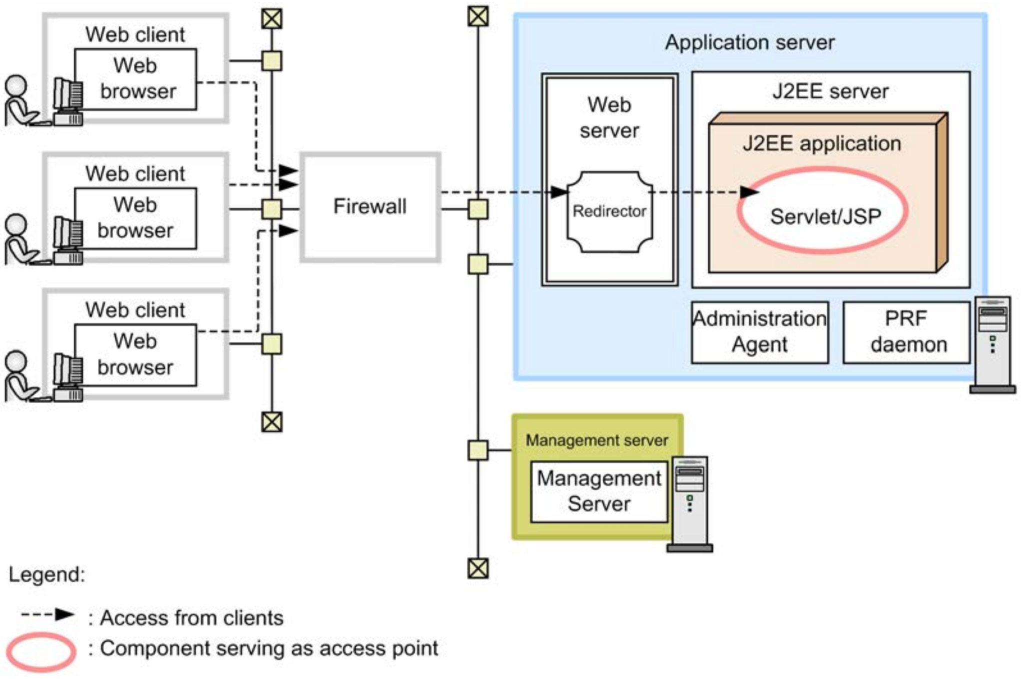
The following is an example of a system configuration that provides access to a servlet and JSP via a firewall.

(1) System configuration features

In this configuration, the firewall is installed before the servlet and JSP as seen from the Web clients.

The figure below shows this configuration. Note that this configuration is for Web server integration.

Figure 3-1: Example configuration providing access to a servlet and JSP via a firewall



For other examples, see *3.2 System configurations*, in the *uCosminexus Application*

Server System Design Guide.

Features

Because access to the servlet and JSP goes through the firewall, this system prevents unauthorized third party access to the system, leakage of information handled by applications, and illegal operation by third parties.

Access from the clients

Access to the servlet and JSP from all the clients goes through the firewall.

(2) Machine software required and processes to be activated

When using the firewall, the necessary software and processes to be activated on the application server machine and client machine are the same as those for system configurations that use servlets and JSPs as access points. See *3.3.1 Deployment of reverse proxies for Web server integration* or *3.3.2 Deployment of reverse proxies for using an in-process HTTP server*.

3.2.2 Deployment of a firewall for Session and Entity Bean

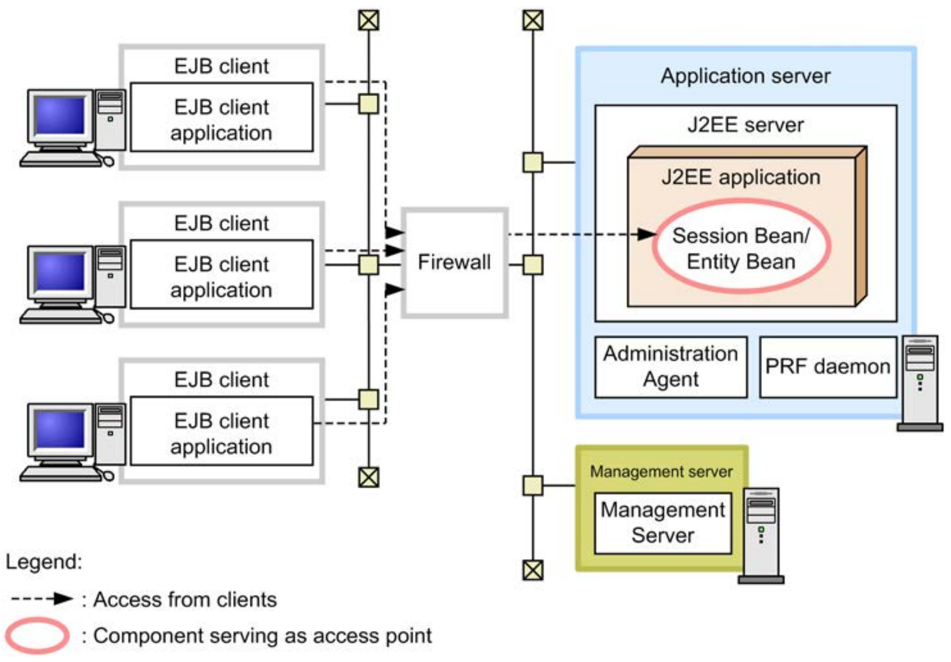
The following is an example of a system configuration that provides access to Session and Entity Bean via a firewall.

(1) System configuration features

In this configuration, the firewall is installed before the Session and Entity Bean as seen from the EJB clients.

The figure below shows this configuration.

Figure 3-2: Example configuration providing access to Session and Entity Bean via a firewall



For other examples, see *3.2 System configurations*, in the *uCosminexus Application Server System Design Guide*.

Features

Because access to the Session and Entity Bean goes through the firewall, this system prevents unauthorized third party access to the system, leakage of information handled by applications, and illegal operation by third parties.

Access from the clients

Access to the Session and Entity Bean from all EJB clients goes through the firewall.

(2) Machine software required and processes to be activated

When using the firewall, the necessary software and processes to be activated on the application server machine and client machine are the same as those for system configurations that use Session and Entity Bean as access points. See *3.4.3 System configuration with Session and Entity Bean serving as access points*, in the *uCosminexus Application Server System Design Guide*.

3.2.3 Firewall deployment with Resource Manager

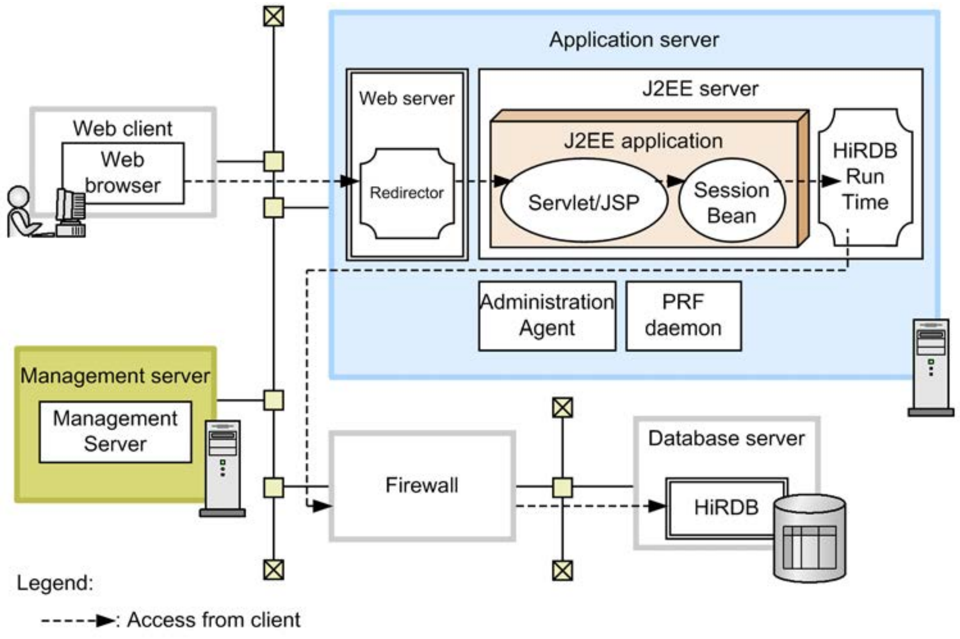
The following is an example of a system configuration that provides access to Resource Manager via a firewall.

(1) System configuration features

In this configuration, the firewall is installed before Resource Manager as seen from the application.

The figure below shows this configuration.

Figure 3-3: Example configuration providing access to Resource Manager via a firewall



For other examples, see *3.2 System configurations*, in the *uCosminexus Application Server System Design Guide*.

Features

Because access to Resource Manager goes through the firewall, this system prevents unauthorized third party access to the system, leakage of information handled by Resource Manager, and illegal operation by third parties.

Access from the clients

Requests from Web browsers on client machines are sent via the Web server to the servlet and JSP. The servlet and JSP call the Session Bean locally. Access to the database from the Session Bean goes through the firewall.

(2) Machine software required and processes to be activated

Activate the software and processes that are appropriate for transaction usage. For details, see *3.6 Types of transactions*, in the *uCosminexus Application Server System Design Guide*.

3.3 Deployment of reverse proxies in a DMZ

This section describes system configurations that involve a reverse proxy deployed in a DMZ to ensure security.

If your system is connected to the Internet, refer to the system configuration details provided here to deploy a reverse proxy.

Note that this section provides information about deploying reverse proxies that are suitable for various types of Web servers that may be used. For information about other security concepts, see *4. Considerations in the Design of a Secure System*.

3.3.1 Deployment of reverse proxies for Web server integration

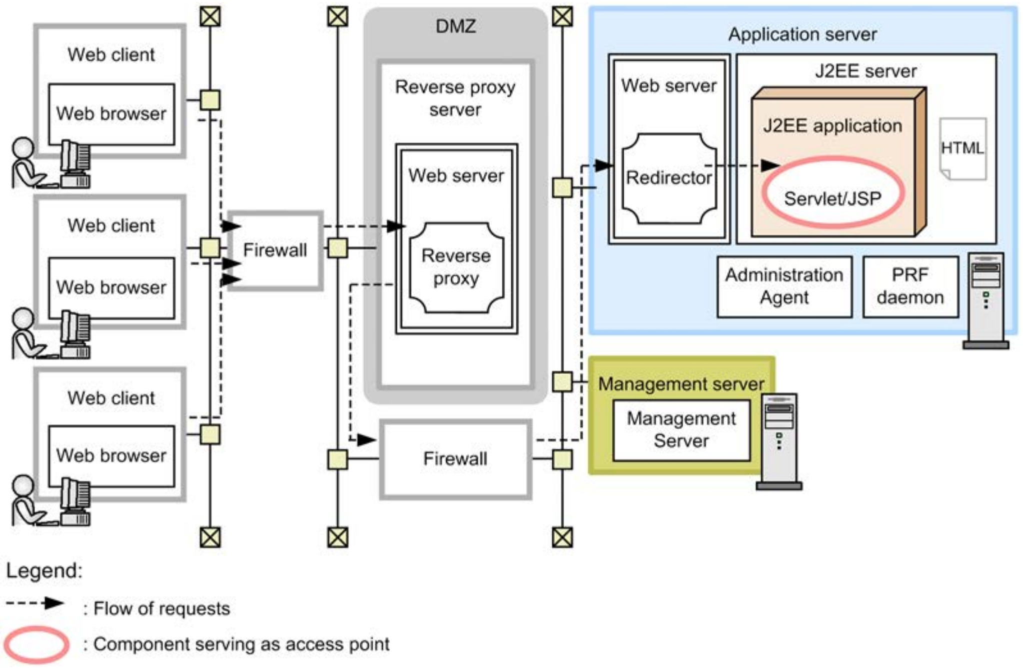
The following are examples of system configurations that employ a reverse proxy for Web server integration.

(1) System configuration features

In this configuration, the reverse proxy server is deployed in a DMZ between the Web browsers and the application server.

The figure below shows an example configuration that uses a reverse proxy deployed in a DMZ for Web server integration.

Figure 3-4: Example configuration using a reverse proxy deployed in a DMZ for Web server integration



For other examples, see *3.2 System configurations*, in the *uCosminexus Application Server System Design Guide*.

Features

- Only the reverse proxy server accesses the application server, preventing direct access to it from Web browsers.
- Usually, the reverse proxy does not store static content such as HTML.

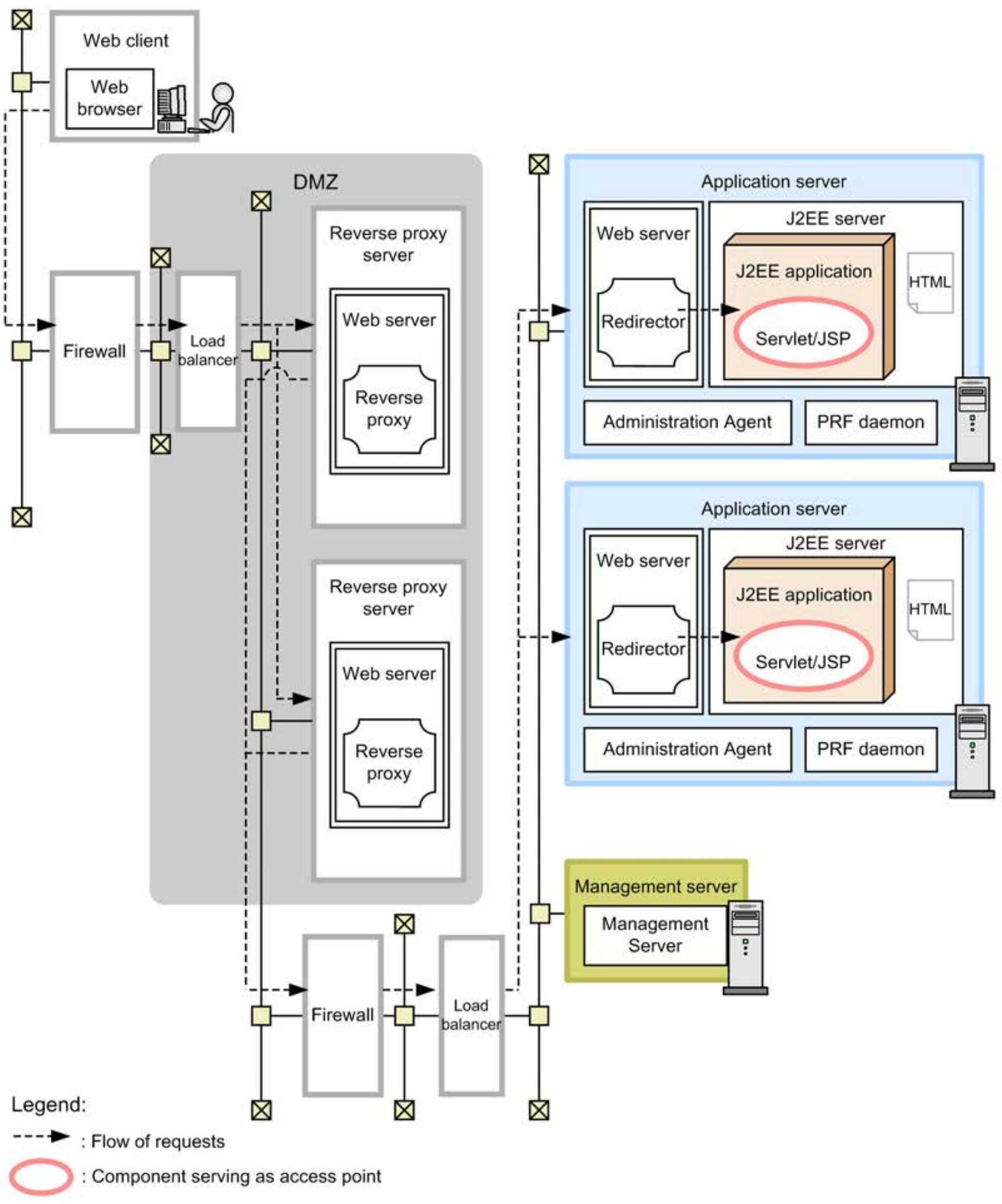
Flow of requests

Access to the servlet and JSP from the clients goes through one Web server containing the reverse proxy module and another containing a redirector module.

A load balance cluster can be used for load balancing by using a load balancer (layer 5 switch) for the reverse proxy server and the application server.

The following figure shows an example load balance cluster configuration using reverse proxies deployed in a DMZ.

Figure 3-5: Example configuration (load balance cluster) using reverse proxies deployed in a DMZ for Web server integration



For other examples, see *3.2 System configurations*, in the *uCosminexus Application Server System Design Guide*.

Features

- Only the reverse proxy servers access the application servers, preventing direct access to them from Web browsers.
- Usually, the reverse proxies do not store static content such as HTML.
- Scalability and availability can be ensured by distributing loads between the reverse proxy server and the application server.

Flow of requests

Access to servlets and JSPs from the client goes through the first load balancer, the Web servers containing the reverse proxy modules, the second load balancer, and then the Web server containing the redirector module.

For access from Web browsers, the first load balancer distributes the load between the two reverse proxy servers. For access from the reverse proxy servers, the second one distributes the load between the two application servers. The second load balancer also manages HTTP session such as affinity or sticky.

Note that when using HTTPS, you need to install an SSL accelerator in front of the first load balancer.

(2) Machine software required and processes to be activated

The following section describes the software and processes required for the machines.

(a) Reverse proxy server machines

Install Cosminexus HTTP Server on the reverse proxy servers. The process shown below should always be activated.

- ■ Web servers

Each Web server should incorporate a reverse proxy module.

(b) Application server machines, management server machine, and client machine

The necessary software and processes to be activated on the application server machines, the management server machine, and the client machine are the same as those for system configurations that use a servlet and JSP as access points. See *3.4.1 System configuration with a servlet and JSP serving as access points (for Web server integration),* in the *uCosminexus Application Server System Design Guide*.

3.3.2 Deployment of reverse proxies for using an in-process HTTP server

The following are examples of system configurations that employ a reverse proxy to use an in-process HTTP server.

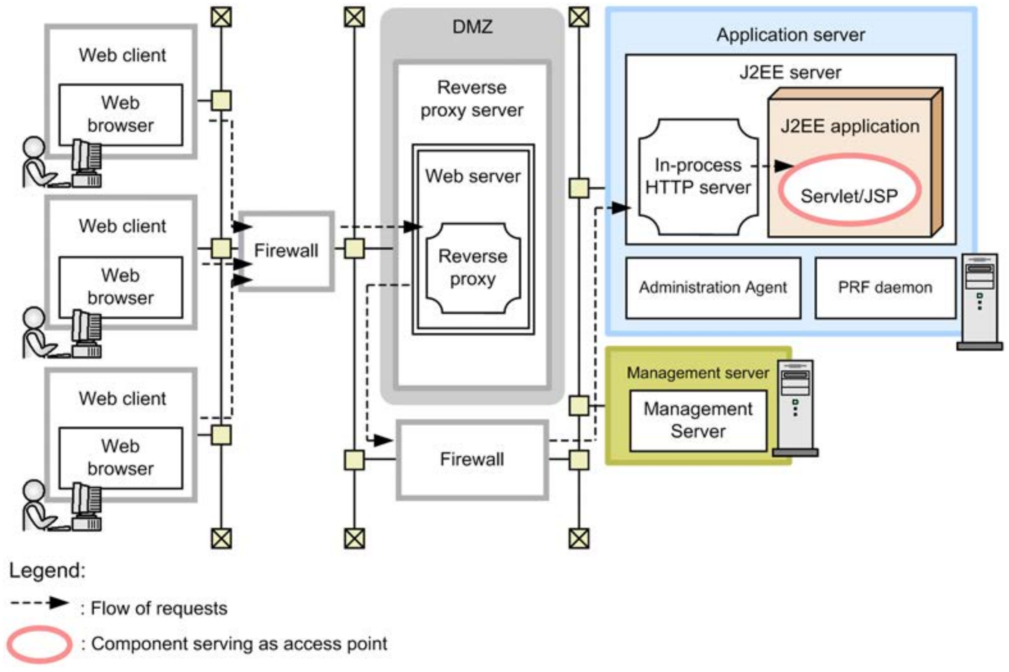
If your system is connected to the Internet and uses an in-process HTTP server, make sure you create a DMZ and install a reverse proxy in that DMZ by referring to the system configuration details provided here.

(1) System configuration features

In this configuration, the reverse proxy is deployed in a DMZ between the Web browsers and the application server.

The figure below shows an example configuration with a reverse proxy deployed in a DMZ to use an in-process HTTP server.

Figure 3-6: Example configuration with a reverse proxy deployed in a DMZ to use an in-process HTTP server



For other examples, see *3.2 System configurations*, in the *uCosminexus Application Server System Design Guide*.

Features

- Only the reverse proxy server accesses the application server, preventing direct access to it from Web browsers.
- Usually, the reverse proxy does not store static content such as HTML.

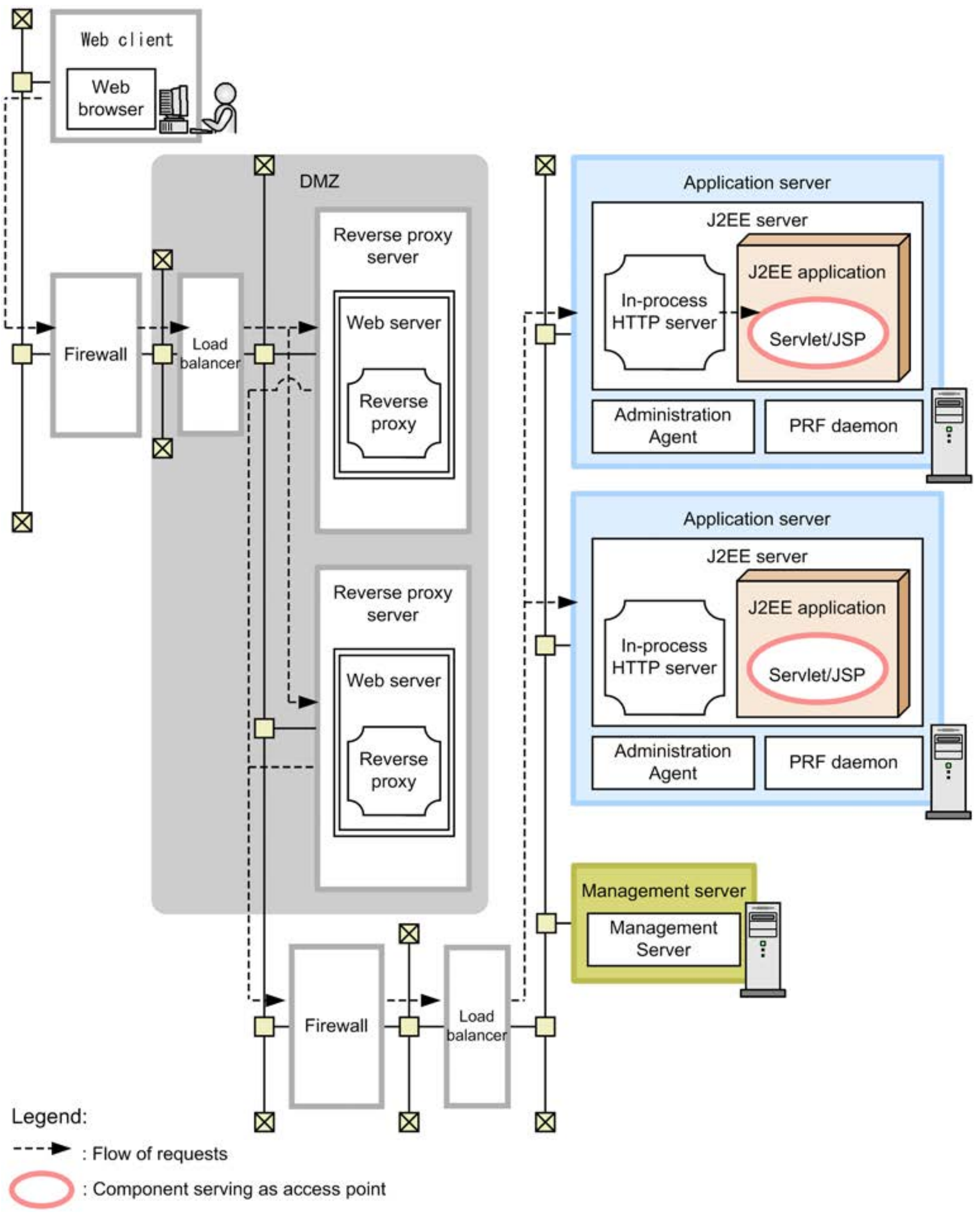
Flow of requests

Access to the servlet and JSP from the clients goes through the Web server containing the reverse proxy module.

A load balance cluster can be used for load balancing by using a load balancer (layer 5 switch) for the reverse proxy server and the application server.

The next figure shows an example load balance cluster configuration with reverse proxies deployed in a DMZ.

Figure 3-7: Example configuration (load balance cluster) with reverse proxies deployed in a DMZ to use an in-process HTTP server



For other examples, see *3.2 System configurations*, in the *uCosminexus Application Server System Design Guide*.

Features

- Only the reverse proxy servers access the application servers, preventing direct access to them from Web browsers.
- Usually, the reverse proxies do not store static content such as HTML.
- Scalability and availability can be ensured by distributing the load between the reverse proxy server and the application server.

Flow of requests

Access to servlets and JSPs from the client goes through the first load balancer, the Web servers containing the reverse proxy modules, and then the second load balancer.

For access from Web browsers, the first load balancer distributes the load between the two reverse proxy servers. For access from the reverse proxy servers, the second one distributes the load between the two application servers. The second load balancer also manages HTTP session such as affinity or sticky.

Note that when using HTTPS, you need to install an SSL accelerator in front of the first load balancer.

(2) Machine software required and processes to be activated

The following section describes the software and processes required for the machines.

(a) Reverse proxy server machines

Install Cosminexus HTTP Server on the reverse proxy server machines. The process shown below should always be activated.

- ■ Web servers

Each Web server should incorporate a reverse proxy module.

(b) Application server machines, management server machine, and client machine

The necessary software and processes to be activated on the application server machines, the management server machine, and the client machine are the same as those for system configurations that use a servlet and JSP as access points. See *3.4.2 System configuration with a servlet and JSP serving as access points (using an in-process HTTP server)*, in the *uCosminexus Application Server System Design Guide*.

Chapter

4. Considerations in the Design of a Secure System

In order to ensure that a business system is running safely and that the data it handles is protected, it is necessary to consider security thoroughly during the system design phase. This chapter describes how to approach the design of the system and what procedures and audit methods are necessary and appropriate in order to configure and operate a secure system.

It also describes how to clarify the security threats to be expected when the system uses an external network and how to use hardware and software to protect against such threats.

Refer to this chapter when the system is executing J2EE applications. This chapter does not apply to systems that execute batch applications.

4.1 Organization of this chapter

This chapter describes how to approach the design of the system and what procedures and audit methods are necessary and appropriate in order to configure and operate a secure system. The table below shows how the chapter is organized.

Table 4-1: Organization of this chapter (Considerations in the Design of a Secure System)

Part	Title	Relevant information
Description	Overview of considerations in the design of a secure system	4.2
	Considering the configuration of a secure system	4.3
	Considering the users of the system	4.4
	Considering the resources handled by the system	4.5
	Checking the preconditions for a secure system	4.6
	Analyzing expected threats	4.7
	Considering countermeasures	4.8
	Considering work procedures	4.9
	Checking how to audit the system	4.10
	Considering the security of systems that use external networks	4.11

Note: This chapter does not include information on implementation, setup, operation, or precautions.

4.2 Overview of considerations in the design of a secure system

A system is expected to encounter various security threats. Such threats might come in the course of its configuration and operation in the hands of the users who manage or operate it. They might also come in the course of end-users' use of the services the system provides. To protect the system against such threats, it is necessary to implement countermeasures such as designing a physically secure system and establishing operation rules for workers.

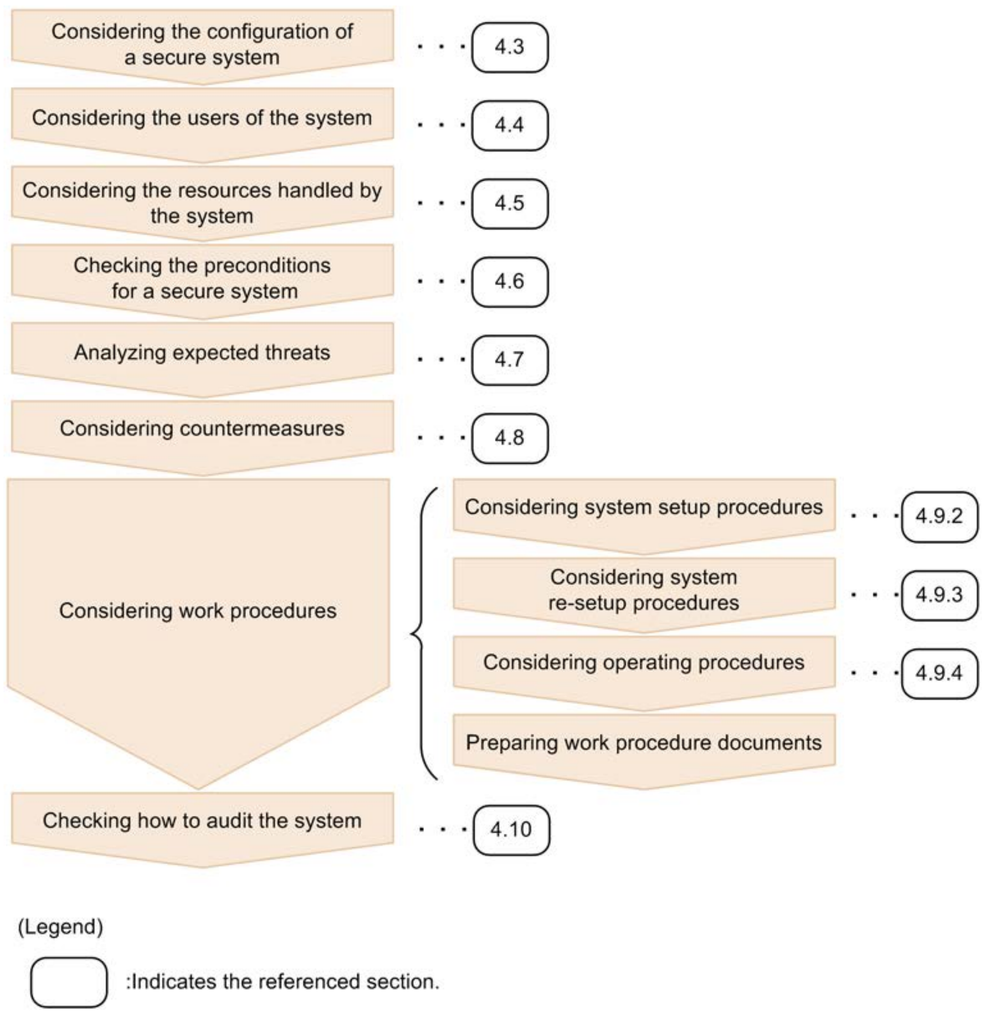
Recent years have seen an increase in the importance of internal control within organizations, from the perspectives of ensuring healthy organizational operations and safely configuring and operating increasingly complicated and diverse IT systems. Internal control requires an organization to prove to Auditors that it maintains the security of its systems. To achieve this, it is necessary to log the operations performed on the system, including who performed the operations and when, and to provide an auditing mechanism to verify that the operations were properly performed by employees who are duly authorized to use the system.

To implement such a secure system, it is necessary to clarify expected threats during system design and consider a system in which appropriate countermeasures against the threats can be implemented.

This chapter describes the points that must be considered during system design. It clarifies expected system threats and then describes how to approach the design of the system and what procedures are necessary in order to configure and operate a secure system.

Considering the design of a secure system involves the steps shown in the following flowchart.

Figure 4-1: Flow of considerations in the design of a secure system


Considering the configuration of a secure system
4.3
Considering the users of the system
4.4
Considering the resources handled by the system
4.5
Checking the preconditions for a secure system
4.6
Analyzing expected threats
4.7
Considering countermeasures
4.8
Considering work procedures
Considering system setup procedures
4.9.2
Considering system re-setup procedures
4.9.3
Considering operating procedures
4.9.4
Preparing work procedure documents
Checking how to audit the system
4.10
(Legend)
:Indicates the referenced section.

This figure shows a work flow for ensuring the security of a system that is used within a company. For details about countermeasures against external threats, see *4.11 Considering the security of systems that use external networks*.

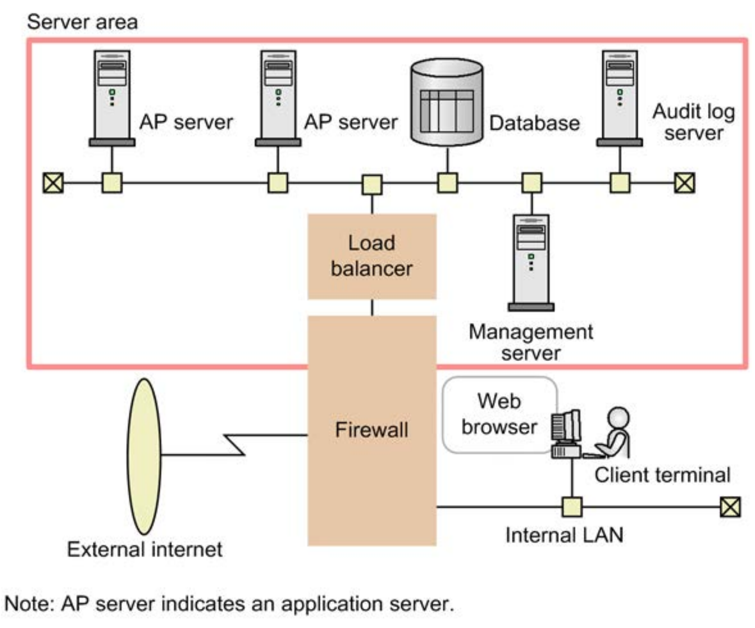
4.3 Considering the configuration of a secure system

This section describes the configuration for a secure system. In its consideration of a secure system, this manual makes the following assumptions:

- The system is running continuously and is used within a large company.
- All system components are deployed on an internal LAN.
- A user uses a Web browser from an internal client terminal to use services provided by the system.
- To use services from such a client terminal, the user is required to log in. A user who is not registered in the system cannot use a service.

The figure below shows the configuration of the system.

Figure 4-2: Secure system configuration



These system components are described below. For the definitions of *System administrator*, *System operator*, *Auditor*, and *end-user* that appear in the following description, see *4.4 Considering the users of the system*.

Server area

This area is a physically isolated space used to manage hardware. The hardware within the server area is managed by the System administrator. Only the System administrator, the System operators, and the Auditor are allowed to enter the server area.

Application (AP) servers

An application server is a machine running a Web server, a service-providing J2EE application, and a server program necessary to run the J2EE application. Multiple application servers are installed in the server area, and the load between the servers is balanced by a load balancer.

Database

This is a database machine that stores user information and information processed by services. It is installed in the server area.

Audit log server

The audit log server collects audit logs for auditing. Only the Auditor can use this server. It is installed in the server area.

Management server

The management server is a machine running a management program that manages the application servers. It is used by the System administrator to configure the system and by System operators to operate the system. It is installed in the server area.

Load balancer

The load balancer is a machine that is used to balance load if multiple application servers are installed. It is installed in the server area.

Firewall

Firewalls are installed between the server area, the internal LAN, and the external Internet.

Client terminal

A client terminal is used to access the services provided by the system. An end-user uses a Web browser on a client terminal to access an application server via an internal LAN.

4.4 Considering the users of the system

To consider a secure system, define the system users first. Clarify which users are expected to access the system and clearly define the purpose and scope of work for each user. This will provide a base for verifying whether a given operation is performed by an authorized user who is permitted to carry out that operation -- one of the objectives of system auditing.

The work procedures of each user must be defined by preparing work procedure documents. Such work procedure documents might include *System Setup Procedure*, *System Operating Procedure*, *End-User Operating Procedure*, and *Entry and Exit Procedure* documents. For a description of considerations for work procedure documentation, see *4.9 Considering work procedures*.

For this example system, the following users are defined:

System administrator

The System administrator is responsible for configuring and managing the system according to a *System Setup Procedure* document. Specifically, the System administrator mainly performs the following types of work:

- Installing and configuring hardware, software, and networks within the server area
- Updating software
- Starting and stopping the system

A user selected from the Information Systems department of the company serves as the System administrator.

System operator

System operators are responsible for operational work within the server area, including registration and deletion of end-users, according to a *System Operating Procedure* document. Users entrusted by the Information Systems department of the company serve as System operators.

End-user

End-users access system-provided services according to an *End-User Operating Procedure* document. End-users access services via a Web browser on a client terminal connected to an internal LAN.

Auditor

The Auditor is responsible for entering and exiting the server area to collect audit logs according to an *Entry and Exit Procedure* document. The Auditor examines the collected audit logs and verifies whether the system has been configured in the appropriate manner by a trusted System administrator in accordance with the *System Setup Procedure* document. The Auditor also verifies whether the system is operated and used in the appropriate manner in accordance with the *System Operating Procedure* and *End-User Operating Procedure* documents. A user selected from the Compliance department of the company responsible for internal auditing serves as the Auditor.

4.5 Considering the resources handled by the system

When approaching the design of a secure system, it is necessary to clearly determine what types of resources or data handled by the system need to be protected.

For the example system given in this section, it is determined that, among the types of resources or data handled by the system, the following need to be protected:

- User information of system administrators
- User information of end-users
- Configuration files used for system configuration
- J2EE applications
- Information sent by end-users and processed by J2EE applications during service use
- Audit logs

For those resources determined to require protection, it is necessary to take some measures including access permission control. For details about such measures, see *4.8 Considering countermeasures*.

4.6 Checking the preconditions for a secure system

This section describes the preconditions for a secure system.

To build a secure system, it is necessary to check the preconditions regarding hardware installation methods and workers. After getting a grasp of such preconditions, use the functions provided by the application server and the OS to implement countermeasures against expected threats.

This section assumes the following two types of preconditions:

- Physical preconditions
- Operational preconditions

4.6.1 Physical preconditions

The physical preconditions to be met in configuring a secure system are described below.

- The hardware running the system, the firewall, the servers, and the internal network should be installed within a server area that is physically isolated from the outside.
- Appropriate controls such as entry and exit management should be applied to prevent non-authorized users from entering the server area.
- No hardware or software that is not necessary for running the system should be allowed to be brought into the server area.

4.6.2 Operational preconditions

The operational preconditions to be met in configuring a secure system are described below.

Operational preconditions include those for work procedure document, system management, system operation, and system auditing. These preconditions are described below.

Preconditions for work procedure documents

Procedures for the configuration, management, and operation of the system should be described in the *System Setup Procedure*, *System Operating Procedure*, and *End-User Operating Procedure* documents respectively.

Preconditions for system management

The hardware, software, and J2EE applications that are necessary within the server area to run the system should be configured and set up by the System administrator in accordance with the *System Setup Procedure* document. The

System administrator should be selected from among trusted staff.

Preconditions for system operation

The hardware, software, and J2EE applications that are necessary within the server area to run the system should be operated by the System operators.

Preconditions for system auditing

The Auditor who audits the system should be selected from among trusted staff.

4.7 Analyzing expected threats

This section analyzes the threats that can be expected for the system, based on the information examined in *4.4 Considering the users of the system* and *4.5 Considering the resources handled by the system,* and checked in *4.6 Checking the preconditions for a secure system*.

Threats expected for the system are as follows.

- Service use by an unauthorized user

  An end-user who is not registered in the system is able to use a service.

- Service use by a user who does not follow the procedure document

  An end-user who obtains a user ID and password registered in the system might not follow the *End-User Operating Procedure* document, and exploit a vulnerability in the system in order to use a service.

  Also, a user registered in the system might use a service that he or she is not authorized to use.

- System configuration by an unauthorized System administrator

  A user who is not a System administrator might enter the server area and configure the system illegally and without following the *Entry and Exit Procedure* document.

- System operation by an unauthorized System operator

  A user who is not a System operator might enter the server area and operate the system illegally and without following the *Entry and Exit Procedure* document.

- System operation by a System operator who does not follow the procedure document

  A user might use the Management Server management user account of a System operator and operate the system in a manner that does not comply with the *System Operating Procedure* document.

To protect the system against these threats, implement the countermeasures described in *4.8 Considering countermeasures.*

For details about the *Entry and Exit Procedure*, *System Setup Procedure*, *System Operating Procedure,* and *End-User Operating Procedure* documents, see *4.9 Considering work procedures*.

4.8 Considering countermeasures

This section describes the countermeasures that should be implemented and the behavior of the system when these countermeasures are implemented.

Countermeasures to be implemented are classified into the following two types:

- Countermeasures to be implemented against preconditions

  This type of measure is for dealing with the preconditions checked in *4.6 Checking the preconditions for a secure system*.

- Countermeasures to be implemented against expected threats

  This type of measure is for dealing with the expected threats described in *4.7 Analyzing expected threats*.

The following subsection describes these types of measures.

4.8.1 Countermeasures to be implemented against preconditions

This subsection describes countermeasures to be implemented against the preconditions checked in *4.6 Checking the preconditions for a secure system*.

The table below shows the preconditions checked in *4.6 Checking the preconditions for a secure system* and the countermeasures to be implemented.

Table 4-2: Preconditions and countermeasures to be implemented

Precondition type	Countermeasure
Physical preconditions	• Physical countermeasures
Operational preconditions	• Measures for the System administrator • Measures for System operators • Measures for the System auditor

These countermeasures are outlined below.

(1) Countermeasures for physical preconditions

The countermeasures for physical preconditions are as follows.

- ■ Physical countermeasures
  - The System administrator should install the hardware running the system, the firewall, the servers, and the internal network within a server area that is physically isolated from the outside.
  - The System administrator should not bring into the server area any hardware or software that is not necessary for running the system.

- The System administrator, System operator, and Auditor should enter and exit the server area in accordance with the *Entry and Exit Procedure* document.

For details about the *Entry and Exit Procedure* document, see *4.9 Considering work procedures*.

(2) Countermeasures for operational preconditions

The countermeasures for operational preconditions are described below.

- ■ Measures for the System administrator
    - For the position of System administrator, a trusted user who will be responsible for the entire system and will not conduct malicious acts should be selected.
    - The System administrator should be trained about system configuration and management and should be familiar with system configuration and management methods. The System administrator also should be familiar with methods for configuring and managing the hardware that will be used in the system.
    - The System administrator should configure and manage the system, taking security precautions into consideration.
    - The System administrator should set difficult-to-guess, highly secure OS and Management Server management passwords for him or herself and for the System operators.

    For details about the *System Setup Procedure* document, see *4.9 Considering work procedures*.
- ■ Measures for System operators
    - The System operators should be trained in system operation and be familiar with system operation methods.
    - The System operators should take security precautions into consideration when operating the system.
    - The System operator should set difficult-to-guess, highly secure passwords for end-users.

    For details about the *System Operating Procedure* document, see *4.9 Considering work procedures*.
- ■ Measures for the System auditor
    - For the position of Auditor, a trusted user who will be responsible for the entire system and will not conduct malicious acts should be selected.
    - The Auditor should be a user who is not a System administrator.

- The Auditor should verify the validity of the system setup procedures. The Auditor also audits the validity of the operating procedures.

4.8.2 Countermeasures to be implemented against expected threats

This subsection describes countermeasures to be implemented against the expected threats described in *4.7 Analyzing expected threats*.

The table below shows the threats expected for the system, and countermeasures against them for each target user type. For details about these threats, see *4.7 Analyzing expected threats*.

Table 4-3: Expected threats and countermeasures to be implemented

Target user	Threat	Countermeasure
System administrator	System configuration by an unauthorized System administrator	• OS-based user identification and authentication
System operator	System operation by an unauthorized System operator	• OS-based user identification and authentication • User identification and authentication for System operators
	System operation by the System operator, not in compliance with the procedure document	• System audit log output • J2EE application audit log output
End-user	Service use by an unauthorized user	• J2EE application audit log output • J2EE application-based user identification and authentication
	Service use by a user who does not follow the procedure document	• J2EE application audit log output • J2EE application-based access control

These countermeasures are outlined below.

Measures for the System administrator

- OS-based user identification and authentication

  Configure user identification and authentication on the OS running the system to control command execution permissions so that the system can only be managed by the System administrator.

Measures for System operators

- OS-based user identification and authentication

  Configure user identification and authentication on the OS running the system to control command execution permissions so that the system can be operated by a System operator.

- User identification and authentication of System operators

  Configure user identification and authentication on the system so that the system can be operated by the System operator.

- System audit log output

  In order to audit whether the system has been operated in accordance with the relevant procedure documents, output system audit logs.

- J2EE application audit log output

  In order to audit whether end-users have been managed in accordance with the relevant procedure documents, use the audit log output API provided by the application server to implement J2EE applications and output J2EE application audit logs. For details about how to implement a J2EE application using the audit log output API, see *6.8 Implementation process for outputting the application's audit log entries*, in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

Measures for end-users

- J2EE application audit log output

  In order to audit whether authorized end-users have used services in accordance with the relevant procedure documents, use the audit log output API provided by the application server to implement J2EE applications and output J2EE application audit logs. For details about how to implement a J2EE application using the audit log output API, see *6.8 Implementation process for outputting the application's audit log entries*, in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

- J2EE application-based user identification and authentication

  Implement user identification and authentication for J2EE applications so that services can only be used by authorized end-users.

- J2EE application-based access control

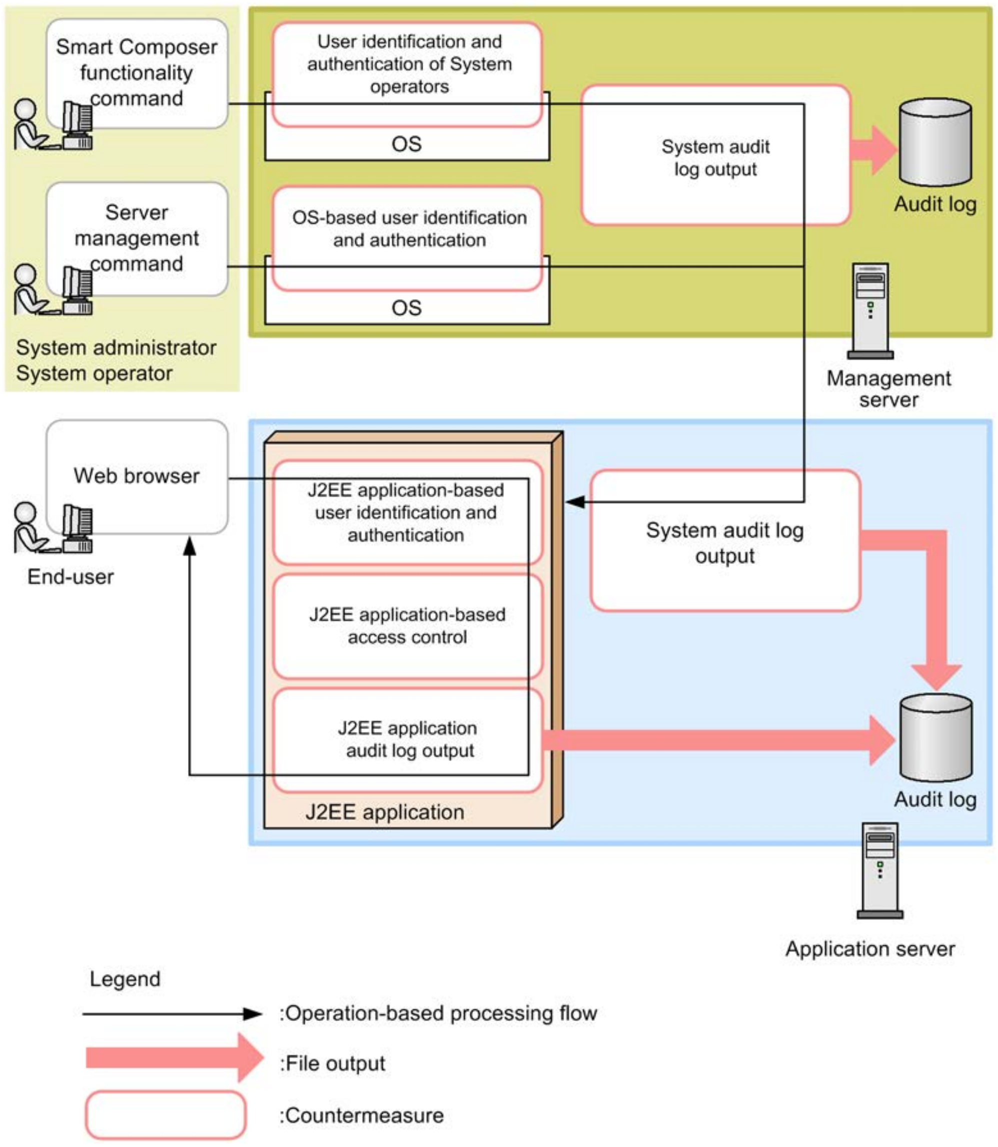
  Implement access control for J2EE applications so that protected data can only be accessed by end-users who have access permission.

4.8.3 Secure system behavior with the countermeasures implemented

This subsection describes the behavior of a secure system with the countermeasures implemented.

The figure below outlines the behavior of the system when the countermeasures are implemented. Each countermeasure shown in this figure corresponds to one described in *4.8.2 Countermeasures to be implemented against expected threats*.

Figure 4-3: System administrator operations and system behavior


Smart Composer functionality command
Server management command
System administrator
System operator
User identification and authentication of System operators
OS
OS-based user identification and authentication
OS
System audit log output
Audit log
Management server
Web browser
End-user
J2EE application-based user identification and authentication
J2EE application-based access control
J2EE application audit log output
J2EE application
System audit log output
Audit log
Application server
Legend
:Operation-based processing flow
:File output
:Countermeasure

The behavior of a system in which the countermeasures shown in this figure are implemented is described below for each user type.

(1) System administrator and System operator operations, and system behavior

The operations of the System administrator and System operator and the system behavior are outlined below.

System administrator's operations

- Uses Smart Composer functionality commands to configure the application server. However, for setup work for J2EE applications, resources, etc., server management commands are used.
- Deploys and starts the application in which the user identification and authentication functions, the access control function, and the audit log function are implemented.

System operator's operations

Uses Smart Composer functionality commands to operate the system. However, for log collection in the case of an error, the `snapshotlog` command is used.

System behavior

Creates an audit log entry for each operation performed by the execution of a command.

Hint:

Some commands do not create an audit log entry. Before using a command, check whether it creates an audit log entry. For details about the commands that create audit log entries, see *6.6 List of commands and operations for outputting audit log entries*, in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

(2) End-user operations and system behavior

System (J2EE application) behavior and end-user operations for using system-provided services are outlined below.

End-users' operations

Uses a Web browser on a client terminal to send an HTTP request to the application server.

System (J2EE application) behavior

- Identifies the user information included in the HTTP request to authenticate the user.
- Uses the Access Control function to check whether the authenticated user has permissions.
- Executes the J2EE application service in accordance with the requests permitted by the Access Control function.
- Compiles an audit log during processing.

4.9 Considering work procedures

This section describes the work procedures to be considered for each system in order to configure and operate a secure system.

To configure and operate a secure system, it is necessary to clarify the work procedures to be performed by each type of worker. In system auditing, audit logs are compared with the relevant work procedure documents to check for consistency. A work procedure document is a written document that describes the authorized procedures and methods for work such as configuring or operating the system or for service use by end-users. A work procedure document needs to be prepared for each system. Checking audit logs, which are records of workers and operations, against the relevant work procedure documents helps to clarify whether authorized workers have performed operations in accordance with the authorized methods and procedures. This helps to maintain the security of the system.

To prepare each work procedure document, it is necessary to consider what work needs to be included in the document, as well as the work procedure and method. From the point of view of auditing, clarify what work needs to be specified so that the work is done by an authorized user in accordance with an authorized procedure and method. Also, specify in each work procedure that a command that creates an audit log entry must be used to perform the work.

4.9.1 Overview of work procedure documents to be prepared

This subsection outlines the work procedure documents that need to be prepared.

The work procedure documents to be prepared depend on the system. For this system the following work procedure documents are prepared.

- Entry and Exit Procedure document

  This document specifies the entry and exit management procedures for entering and exiting the server area.

- System Setup Procedure document

  This document specifies the procedures for configuring the system. Procedures to be written should be based on those described in *4.9.2 Considering the system setup procedures* and *4.9.3 Considering the system re-setup procedures*.

  To perform the operations in the procedures described in *4.9.2 Considering the system setup procedures* and *4.9.3 Considering the system re-setup procedures*, the System administrator uses Smart Composer functionality commands and server management commands. In addition, commands that create an audit log entry should be used for all operations.

- System Operating Procedure document

This document specifies the procedures for operating the system. Procedures to be written should be based on those described in *4.9.4 Considering system operating procedures*.

To perform the operations in the procedures described in *4.9.4 Considering system operating procedures*, the System operator uses Smart Composer functionality commands and the `snapshotlog` command.

- End-user Operating Procedure document

  This document specifies the procedure for using services provided by the system.

4.9.2 Considering the system setup procedures

This subsection gives examples of the system setup procedures to be written in the *System Setup Procedure* document. When preparing a *System Setup Procedure* document, refer to these sample procedures.

To set up a secure system, Smart Composer functionality commands and server management commands are used. In addition, commands that create an audit log entry should be used for all operations. When adding an operation other than those described here to the work procedure document, use commands that create audit log entries. For details about commands that create audit log entries, see *6.6 List of commands and operations for outputting audit log entries*, in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

Note that all procedures described in this section are to be performed by the System administrator.

(1) Installing hardware

Hardware should be installed by the System administrator. The procedure for installing hardware is as follows:

1. Enter the server area, which is physically isolated from the outside, in accordance with the *Entry and Exit Procedure* document.
2. Install the hardware for running the system and a firewall within the server area.

In the *System Setup Procedure* document, include detailed procedures for installing the hardware and the firewall.

(2) Installing the OS

The OS to be used on the system should be installed by the System administrator. The procedure for installing the OS is as follows:

1. Install the OS and configure the settings necessary for network connection, including the IP address and host name settings.
2. Apply the necessary security patches to the system.

3. Install the software necessary for the system and set the environment variables.
4. Create an OS account for the System administrator and assign administrator privileges to the account.
5. Create an OS account for the Auditor and use a secure means to notify the Auditor of the account.

(3) Starting system management

The System administrator logs into the OS using the System administrator account created in step *(2) Installing the OS*.

(4) Configuring settings for audit log output

On the management server and application server terminals, the System administrator configures the audit log output settings. The procedure for configuring the audit log output settings is as follows:

1. Decide the size of the audit log file, based on the system configuration.
2. Assign the System administrator and System operator permission to read and write to the audit log file. Also, assign the Auditor permission to read the audit log file.
3. Update the audit log definition file (`auditlog.properties`) using the information determined or set in steps 1 and 2.
4. Create the audit log output directory specified in the audit log definition file.
5. Assign the System administrator and System operator permission to read and write to the audit log output directory created in step 4. Also, assign the Auditor permission to read the audit log output directory created in step 4.
6. Execute the setup command (`auditsetup` command).

(5) Configuring the load balancer and database

The System administrator should install the load balancer and database within the server area and configure the machines.

In the *System Setup Procedure* document, include detailed procedures for configuring the load balancer and database.

(6) Configuring the management server

The System administrator should initialize the management server. The procedure for configuring the management server is as follows:

1. In the `mngsvrctl` command, specify the argument `setup` to set up Management Server, and then configure the management user account for Management Server.
2. In the `mngautorun` command, specify the argument `server` and the `-sync` option to configure Management Server to start automatically.

(7) Defining the configuration of the Web system

The System administrator should define the configuration of the Web system. The procedure for defining the configuration of the Web system is as follows:

1. In the `mngsvrctl` command, specify the argument `start` and the `-sync` option to start Management Server.
2. Edit and save the Easy Setup definition file.
3. In the `adminagentctl` command, specify the `-sync` option to start Administration Agent on each application server.
4. On the management server, use the `cmx_build_system` command to set up the Web system.

(8) Preparing the Web system

The System administrator should use the appropriate Smart Composer functionality commands on the administrator terminal of the management server to prepare the Web system. The procedure for preparing the Web system is as follows:

1. Use the `cmx_start_target` command to set the Web system to the standby state.
2. Use the `cmx_list_status` command to confirm that the service unit in the Web system is in the standby state.

(9) Configuring the resource adapter

The System administrator should use server management commands on the administrator terminal of the management server to configure the resource adapter that is necessary for applications to link with the database. The procedure for configuring a resource adapter is as follows:

1. From the following directory, copy a `Connector` property file template for the resource adapter to be used.

   In Windows

   Cosminexus-installation-directory`\CC\admin\templates\`

   In UNIX

   `/opt/Cosminexus/CC/admin/templates/`
2. Edit the `Connector` property file template copied in step 1.
3. Use the `cjimportres` command to import the resource adapter.
4. Use the `cjsetresprop` command to incorporate the edited contents of the `Connector` property file into the resource adapter.
5. Use the `cjdeployrar` command to deploy the resource adapter.

6. Use the `cjtestres` command to test the resource adapter connection.

(10) Verifying the J2EE application

The System administrator should verify that the countermeasures described in *4.8.2 Countermeasures to be implemented against expected threats* are implemented in the J2EE application. The countermeasures to be verified are as follows:

- J2EE application audit log output
- J2EE application-based user identification and authentication
- J2EE application-based access control

Specifically, confirm that the J2EE application meets the following specifications:

- Provides the capability for the System operator to register and delete the user IDs and passwords of end-users.
- Provides the capability to identify and authenticate user IDs and passwords.
- Provides the capability to control access to the services it provides.
- Provides the capability to create entries in the audit log when a user is using its services.

(11) Configuring the J2EE application

The System administrator should use server management commands on the administrator terminal of the management server to configure the J2EE application. The procedure for configuring the J2EE application is as follows:

1. Use the `cjimportapp` command to import the J2EE application.
2. Use the `cjgetappprop` command to obtain the application integrated property file.
3. Edit the application integrated property file obtained in step 2.
4. Use the `cjsetappprop` command to incorporate the edited contents of the application integrated property file into the J2EE application.

Note:

This describes how to configure a J2EE application without runtime information. To configure a J2EE application that contains runtime information, after importing the J2EE application in step 1, use the `cjstopapp` command to stop the J2EE application, and then proceed to step 2.

(12) Starting the Web system

The System administrator should use the appropriate Smart Composer functionality commands and server management commands on the administrator terminal of the management server to start the Web system. The procedure for starting the Web system is as follows:

1. Use the `cjstartrar` command to start the resource adapter.
2. Use the `cjstartapp` command to start the J2EE application.
3. Use the `cmx_start_target` command to set the service unit within the Web system to a running state.

(13) Disabling unnecessary functions

To prevent unauthorized users from using an unnecessary function, disable it. Specifically, the System administrator should change the command execution permissions or delete the files that are necessary to execute the command. The table below shows the functions that need to be disabled for Windows and UNIX respectively.

Table 4-4: Functions that need to be disabled (for Windows)

Function name	Target directory	Target file	Action
Cosminexus HTTP Server function for GUI server management	*Cosminexus-installation-directory*`\httpsd`	`adm-httpsd.exe`	Deny execution permission to any user other than the System administrator.
Cosminexus HTTP Server command for editing password files	*Cosminexus-installation-directory*`\httpsd\bin`	`htpasswd.exe`	Deny execution permission to any user other than the System administrator.
CTM function for changing the number of concurrent schedule queues	*Cosminexus-installation-directory*`\CTM\bin`	`ctmchpara.exe`	Deny execution permission to any user other than the System administrator.
CTM function for displaying and deleting CTM domain information	*Cosminexus-installation-directory*`\CTM\bin`	`ctmdminfo.exe`	Deny execution permission to any user other than the System administrator.
CTM function for blocking schedule queues	*Cosminexus-installation-directory*`\CTM\bin`	`ctmholdque.exe`	Deny execution permission to any user other than the System administrator.
CTM function for outputting version information on executable files and libraries	*Cosminexus-installation-directory*`\CTM\bin`	`ctmjver.exe`	Deny execution permission to any user other than the System administrator.

Function name	Target directory	Target file	Action
CTM function for editing and outputting messages	*Cosminexus-installation-directory*`\CTM\bin`	`ctmlogcat.exe`	Deny execution permission to any user other than the System administrator.
CTM function for outputting schedule queue information	*Cosminexus-installation-directory*`\CTM\bin`	`ctmlsque.exe`	Deny execution permission to any user other than the System administrator.
CTM function for unblocking schedule queues	*Cosminexus-installation-directory*`\CTM\bin`	`ctmrlesque.exe`	Deny execution permission to any user other than the System administrator.
CTM function for editing and outputting operating statistics	*Cosminexus-installation-directory*`\CTM\bin`	`ctmstsed.exe`	Deny execution permission to any user other than the System administrator.
CTM function for forced output of buffer contents to a file	*Cosminexus-installation-directory*`\CTM\bin`	`ctmstsflush.exe`	Deny execution permission to any user other than the System administrator.
CTM function for outputting version information on executable files and libraries	*Cosminexus-installation-directory*`\CTM\bin`	`ctmver.exe`	Deny execution permission to any user other than the System administrator.
PRF function for editing and outputting performance analysis trace information	*Cosminexus-installation-directory*`\PRF\bin`	`cprfed.exe`	Deny execution permission to any user other than the System administrator.
PRF function for forced output of buffer contents to a file	*Cosminexus-installation-directory*`\PRF\bin`	`cprfflush.exe`	Deny execution permission to any user other than the System administrator.
PRF function for displaying and changing trace acquisition levels	*Cosminexus-installation-directory*`\PRF\bin`	`cprflevel.exe`	Deny execution permission to any user other than the System administrator.
Commands used by Management Server	*Cosminexus-installation-directory*`\manager\bin`	`mngsvrutil.exe`	Deny execution permission to any user other than the System administrator.
	Cosminexus-installation-directory`\manager\bin`	`mstrexport.exe`	Deny execution permission to any user other than the System administrator.
	Cosminexus-installation-directory`\manager\bin`	`mstrimport.exe`	Deny execution permission to any user other than the System administrator.

Function name	Target directory	Target file	Action
	Cosminexus-installation-directory`\manager\bin`	`ssoexport.exe`	Deny execution permission to any user other than the System administrator.
	Cosminexus-installation-directory`\manager\bin`	`ssogenkey.exe`	Deny execution permission to any user other than the System administrator.
	Cosminexus-installation-directory`\manager\bin`	`ssoimport.exe`	Deny execution permission to any user other than the System administrator.
	Cosminexus-installation-directory`\manager\bin`	`uachpw.exe`	Deny execution permission to any user other than the System administrator.
	Cosminexus-installation-directory`\manager\bin`	`mngsvr_adapter_setup.exe`	Avoid use of the command.
	Cosminexus-installation-directory`\manager\bin`	`Adapter_HITACHI_COSMINEXUS_MANAGER.exe`	Deny execution permission to any user other than the System administrator.
	Cosminexus-installation-directory`\manager\externals\jp1\mngsvrmonitor`	`mngsvr_monitor_setup.exe`	Avoid use of the command.
Management portal	*Cosminexus-installation-directory*`\manager\containers\m\webapps\mngsvr`	`index.jsp`	Delete the file.
	Cosminexus-installation-directory`\manager\containers\m\webapps\mngsvr`	`login.jsp`	Delete the file.

Table 4-5: Functions that need to be disabled (for UNIX)

Function name	Target directory	Target file	Action
Cosminexus HTTP Server function for GUI server management	`/opt/hitachi/httpsd/sbin`	`adminctl`	Deny execution permission to any user other than the System administrator.
	`/opt/hitachi/httpsd/sbin`	`adm-httpsd`	Deny execution permission to any user other than the System administrator.
Cosminexus HTTP Server command for editing password files	`/opt/hitachi/httpsd/bin`	`htpasswd`	Deny execution permission to any user other than the System administrator.

Function name	Target directory	Target file	Action
CTM function for changing the number of concurrent schedule queues	`/opt/Cosminexus/CTM/bin`	`ctmchpara`	Deny execution permission to any user other than the System administrator.
CTM function for displaying and deleting CTM domain information	`/opt/Cosminexus/CTM/bin`	`ctmdminfo`	Deny execution permission to any user other than the System administrator.
CTM function for blocking schedule queues	`/opt/Cosminexus/CTM/bin`	`ctmholdque`	Deny execution permission to any user other than the System administrator.
CTM function for outputting version information on executable files and libraries	`/opt/Cosminexus/CTM/bin`	`ctmjver`	Deny execution permission to any user other than the System administrator.
CTM function for editing and outputting messages	`/opt/Cosminexus/CTM/bin`	`ctmlogcat`	Deny execution permission to any user other than the System administrator.
CTM function for outputting schedule queue information	`/opt/Cosminexus/CTM/bin`	`ctmlsque`	Deny execution permission to any user other than the System administrator.
CTM function for unblocking schedule queues	`/opt/Cosminexus/CTM/bin`	`ctmrlesque`	Deny execution permission to any user other than the System administrator.
CTM function for editing and outputting operating statistics	`/opt/Cosminexus/CTM/bin`	`ctmstsed`	Deny execution permission to any user other than the System administrator.
CTM function for forced output of buffer contents to a file	`/opt/Cosminexus/CTM/bin`	`ctmstsflush`	Deny execution permission to any user other than the System administrator.
CTM function for outputting version information on executable files and libraries	`/opt/Cosminexus/CTM/bin`	`ctmver`	Deny execution permission to any user other than the System administrator.
PRF function for editing and outputting performance analysis trace information	`/opt/Cosminexus/PRF/bin`	`cprfed`	Deny execution permission to any user other than the System administrator.

Function name	Target directory	Target file	Action
PRF function for forced output of buffer contents to a file	`/opt/Cosminexus/PRF/bin`	`cprfflush`	Deny execution permission to any user other than the System administrator.
PRF function for displaying and changing trace acquisition levels	`/opt/Cosminexus/PRF/bin`	`cprflevel`	Deny execution permission to any user other than the System administrator.
Commands used by Management Server	`/opt/Cosminexus/manager/bin`	`mngsvrutil`	Deny execution permission to any user other than the System administrator.
	`/opt/Cosminexus/manager/bin`	`mstrexport`	Deny execution permission to any user other than the System administrator.
	`/opt/Cosminexus/manager/bin`	`mstrimport`	Deny execution permission to any user other than the System administrator.
	`/opt/Cosminexus/manager/bin`	`ssoexport`	Deny execution permission to any user other than the System administrator.
	`/opt/Cosminexus/manager/bin`	`ssogenkey`	Deny execution permission to any user other than the System administrator.
	`/opt/Cosminexus/manager/bin`	`ssoimport`	Deny execution permission to any user other than the System administrator.
	`/opt/Cosminexus/manager/bin`	`uachpw`	Deny execution permission to any user other than the System administrator.
	`/opt/Cosminexus/manager/bin`	`mngsvr_adapter_setup`	Deny execution permission to any user other than the System administrator.
	`/opt/Cosminexus/manager/bin`	`Adapter_HITACHI_COSMINEXUS_MANAGER`	Deny execution permission to any user other than the System administrator.
Management portal	`/opt/Cosminexus/manager/containers/m/webapps/mngsvr`	`index.jsp`	Delete the file.
	`/opt/Cosminexus/manager/containers/m/webapps/mngsvr`	`login.jsp`	Delete the file.

(14) Registering the System operator

The System administrator should use OS functions and Smart Composer functionality commands on the administrator terminal of the management server to set the user ID and password for the System operators. The System administrator should notify the System operators of user IDs and passwords set for them. The procedure for registering System operators is as follows:

1. Use the appropriate OS function to set the OS user ID and password for the System operator.
2. Use the appropriate OS function to deny administrator privileges to the System operator.
3. Use the `cmx_admin_passwd` command to replace the management user ID and password of the System administrator on Management Server with those of the System operator.
4. Use a safe means to notify the System operator of the user ID and password set in steps 1 and 3.

4.9.3 Considering the system re-setup procedures

This subsection gives examples of the system re-setup procedures to be written in the *System Setup Procedure* document. When preparing a *System Setup Procedure* document, refer to these sample procedures.

To perform a re-setup of a secure system, Smart Composer functionality commands and server management commands are used. In addition, commands that create an audit log entry should be used for all operations. When adding an operation other than those described here to the work procedure document, use commands that create audit log entries. For details about commands that create audit log entries, see *6.6 List of commands and operations for outputting audit log entries*, in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

Note that all procedures described in this subsection are to be performed by the System administrator.

(1) Replacing a J2EE application

If maintenance of a J2EE application becomes necessary, the System administrator should replace the application. The procedure for replacing a J2EE application is as follows:

1. Use the `cmx_stop_target` command to set the service unit within the Web system to a ready state.
2. Use the `cjstopapp` command to stop the J2EE application to be replaced.
3. Use the `cjdeleteapp` command to delete the J2EE application to be replaced.

4. Use the `cjimportapp` command to import the replacement J2EE application.
5. Use the `cjgetappprop` command to obtain the integrated property file for the replacement J2EE application.
6. Edit the integrated property file obtained in step 5 and configure the settings necessary for the J2EE application. Customize the J2EE application if necessary.
7. Use the `cjsetappprop` command to incorporate the integrated property file edited in step 6 into the replaced J2EE application.
8. Use the `cjstartapp` command to start the replacement J2EE application.
9. Use the `cmx_start_target` command to set the service unit within the Web system to a running state.

In addition to this method, a J2EE application can also be replaced by using the `cjreplaceapp` command or the Redeploy function, or by using the `cjreloadapp` command or the Reload function.

(2) Tuning the system

The System administrator should tune the system if necessary. The procedure for tuning the system is as follows:

1. Edit the Easy Setup definition file.
2. Use the `cmx_stop_target` command to stop the service unit within the Web system.
3. Use the `cmx_build_system` command to reconfigure the Web system settings.
4. Use the `cmx_start_target` command to start the service unit within the Web system.

(3) Reconfiguring the system (by adding a service unit)

If necessary, the System administrator should reconfigure the system by adding a service unit. The procedure for reconfiguring the system by adding a service unit is as follows:

1. Create and edit a reconfiguration definition file.
2. Use the `cmx_change_model` command to change the information model of the Web system on Management Server.
3. Use the `cmx_build_system` command to apply the changed information model of the Web system.
4. Use the `cmx_start_target` command to set the added service unit within the Web system to a ready state.
5. Use the `cjstartrar` command to start the resource adapter.

6. Use the `cjstartapp` command to start the J2EE application.
7. Use the `cmx_start_target` command to set the added service unit within the Web system to a running state.

(4) Reconfiguring the system (by removing a service unit)

If necessary, the System administrator should reconfigure the system by deleting a service unit. The procedure for reconfiguring the system by removing a service unit is as follows:

1. Use the `cmx_stop_target` command to stop the service unit within the Web system to be removed.
2. Use the `cmx_delete_system` command to remove the service unit within the Web system specified in step 1.

4.9.4 Considering system operating procedures

This subsection gives examples of the system operating procedures to be written in the *System Operating Procedure* document. When preparing a *System Operating Procedure* document, refer to these sample procedures.

To operate a secure system, Smart Composer functionality commands and the `snapshotlog` command are used. When adding an operation other than those described here to the work procedure document, use commands that create audit log entries. For details about commands that create audit log entries, see *6.6 List of commands and operations for outputting audit log entries*, in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

Among the tasks described here, certain tasks involving starting the Web system and non-Web system maintenance work should be performed by a System operator. Web system maintenance should be performed by the System administrator when so requested by a System operator.

(1) Starting the Web system

The Web system on the administrator terminal of the management server should be started by a System operator. However, the resource adapters and J2EE applications should be started by the System administrator, using server management commands. The procedure for starting the Web system is as follows:

1. The System operator uses the `cmx_start_target` command to set the service unit within the Web system to a ready state.
2. The System operator requests the System administrator to start services.
3. The System administrator uses the `cjstartrar` command to start the resource adapters.
4. The System administrator uses the `cjstartapp` command to start the J2EE

application.

5. The System operator uses the `cmx_start_target` command to set the service unit within the Web system to a running state.

(2) Stopping the Web system

The System operator should use the appropriate Smart Composer functionality commands on the administrator terminal of the management server to stop the Web system. The procedure for stopping the Web system is as follows:

1. Use the `cmx_stop_target` command to set the service unit within the Web system to a stopped state.

(3) Managing end-users

The System operator should manage end-user access permissions and user IDs according to the *System Operating Procedure* document. The following tasks are performed by a System operator:

- Registering and deleting end-users
- Changing end-user permissions
- Changing end-user passwords

In the *System Operating Procedure* document, include detailed procedures for these tasks.

(4) Notifying end-users

The System operator should notify each end-user registered in *(3) Managing end-users* of his or her user ID and password. The procedure for notifying an end-user of his or her user ID and password is as follows:

1. Use the user ID and password registered for the end-user in accordance with the System Operating Procedures document to ensure the end-user can use the services.
2. Make sure that an audit log is output during service use.
3. After completing the checks in steps 1 and 2, use a safe means to notify the end-user of his or her user ID and password.

(5) Maintaining the Web system

The Web system should be maintained as necessary. Web system maintenance should be performed by the System administrator when so requested by the System operator. The procedure for maintaining the Web system is as follows:

1. The System operator uses the `cmx_stop_target` command to block or stop the service unit to be maintained.
2. To apply a fix patch to the application server, the System operator stops the

application server-related programs. If no fix patch is applied, proceed to step 3.

3. The System operator requests the System administrator to maintain the system.
4. The System administrator uses the System administrator user ID to log into the OS.
5. The System administrator applies service packs and security patches to the OS and applies fix patches to the application server. A System operator must be in attendance while the System administrator is performing this step.
6. If a fix patch is applied to the application server, the System operator restarts the application server-related programs.
7. The System operator uses the `cmx_start_target` command to restart the service unit.

(6) Troubleshooting the system

If an error occurs, it should be handled by the System operator. The procedure for troubleshooting a system error is as follows:

1. Use the `snapshotlog` command to collect application server logs.
2. If necessary, separately collect any logs that cannot be collected with the `snapshotlog` command.
3. Send the collected logs to maintenance personnel for examination.
4. Conduct system maintenance based on the examination results.

4.10 Checking how to audit the system

This section describes how to audit the system.

In system auditing, the Auditor should check operation records output to audit logs against the relevant respective work procedure documents to examine whether operations have been performed by authorized workers in accordance with the authorized procedures.

4.10.1 Obtaining audit logs

The procedure for obtaining audit logs is as follows:

1. The Auditor enters the server area physically isolated from the outside in accordance with the *Entry and Exit Procedure* document.
2. The Auditor logs into the audit log server and obtains audit logs from the machines running a variety of servers.

4.10.2 Examining audit logs

In examining the audit logs, ensure that the following points are observed:

1. A trusted System administrator has set up the system in the proper way.

   Make sure that none of the times, operators, events, or results recorded in the audit logs contradicts the *System Setup Procedure* document.
2. The system is set up in the proper way and is operated and used properly.

   Make sure that none of the times, operators, events, or results recorded in the audit logs contradicts the *System Operating Procedure* or *End-User Operating Procedure* documents.

For details about how to examine audit logs and messages output to audit logs, see the following manuals:

- How to examine audit logs

  See *6.3 What are audit log entries*, in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

4.11 Considering the security of systems that use external networks

This section describes the security threats that can be expected with respect to systems that use external networks, and their countermeasures.

4.11.1 Security threats that can be expected with respect to systems that use external networks

This subsection describes the security threats that can be expected with respect to systems that use external networks.

(1) Expected security threats

If appropriate security measures are not implemented for a system that uses a network, an application might be executed without authorization, or communications or data managed by the backend database might be leaked or altered. To prevent such issues, ascertain the security threats and implement countermeasures against them.

In this subsection, the following security threats are assumed:

- Unauthorized third-party intrusion into the system from the outside
- Leakage of data handled by applications to a third party
- Leakage of application communications to a third party
- Third-party alteration of application communications
- Operation or information acquisition by a system user beyond the scope of permission granted to that user

This subsection discusses countermeasures against these threats from outside the system. It does not discuss threats emanating from within the system.

(2) Possible countermeasures

The countermeasures as shown in the table below can be employed against expected security threats. For more concrete description of each countermeasure, see the indicated reference.

Table 4-6: Possible countermeasures against security threats

Threat	Countermeasure	Relevant information
Unauthorized third-party intrusion into the system from the outside	Deploy a firewall and intrusion detection system.	4.11.2
Leakage of data handled by applications to a third party		
Leakage of application communications to a third party	Encrypt communication.	4.11.3#

Threat	Countermeasure	Relevant information
Third-party alteration of application communications		
Operation or information acquisition by a system user beyond the scope of permission granted to that user	Authenticate users from within applications.	4.11.4

#: To encrypt communication, HTTPS is used. The relevant information describes how to use an SSL accelerator to handle encrypted communications in the case that HTTPS is used.

4.11.2 Deploying a firewall and intrusion detection system

This subsection describes how to properly deploy and configure a firewall and intrusion detection system in order to improve system security.

(1) Purposes of deploying a firewall and intrusion detection system

A firewall controls access between the external and internal networks. To prevent unauthorized access from an external network, it blocks communication other than that which is permitted for predetermined clients or communications, in accordance with predetermined rules. To use a firewall, it is therefore necessary to clarify and specify the ports or IP addresses for which communication is permitted.

An intrusion detection system (IDS) monitors the communication line and uses communication patterns to determine whether access is authorized.

Deploying a firewall and intrusion detection system at appropriate points and configuring them helps to protect the system from the following security threats:

- Unauthorized third-party intrusion into the system from the outside
- Leakage of data handled by applications to a third party

This subsection describes where to deploy a firewall and intrusion detection system for each system configuration listed in the following table and the points that need to be considered when configuring them.

Table 4-7: Considerations for deploying a firewall and intrusion detection system for different system configurations

System configuration	Description
Basic Web client configuration	This is a system configuration with a single application server. A Web browser is a client.
Basic EJB client configuration	This is a system configuration with a single application server. An EJB client application is a client.

System configuration	Description
Configuration separating each server layer by a firewall (application-centralized)	This is a system configuration with multiple application servers, each server layer separated by a firewall. All applications run on the same application server layer.
Configuration separating each server layer by a firewall (application-distributed)	This is a system configuration with multiple application servers, each server layer separated by a firewall. Applications run on different application server layers.

When connecting the system to the Internet, we recommend you consider a configuration that uses a reserved DMZ and a reverse proxy so that no application server on the internal network can be directly accessed from external networks.

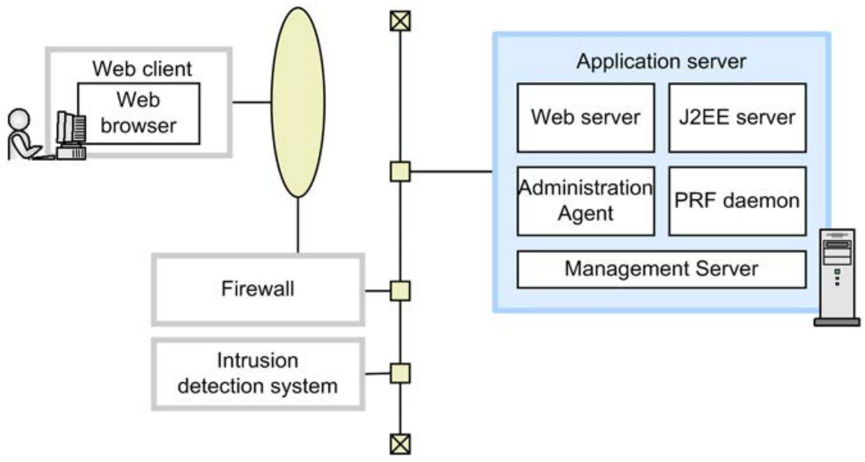
(2) Basic Web client configuration

This section describes where to deploy a firewall and intrusion detection system for a basic Web client configuration with a single application server.

When viewed from the network, the firewall should be deployed in front of the application server. In this configuration, a Web client on the network can only access the application server via the firewall.

The figure below shows an example of a firewall and intrusion detection system deployment for a basic Web client configuration.

Figure 4-4: Example firewall and intrusion detection system deployment for a basic Web client configuration



(a) Configuring the application server

For the application server, configure the following settings:

- Specifying the address of the communication port for J2EE server management

Specify the address at which the J2EE server management communication port can be accessed.

- Specifying the addresses of Management Server and Administration Agent

  Specify the addresses at which Management Server and Administration Agent can be accessed.

(b) Configuring the firewall

To control access between external networks and the Web server (Cosminexus HTTP Server) within the application server, configure the following settings:

- Permitting access from external networks to the Web server

  For communication between networks external to the firewall and the application server, only permit access to a public port such as HTTP/80 or HTTPS/443. Depending on the system configuration, permit access to a different communication port such as for DNS as necessary.

- Limiting access based on the IP addresses of Web clients (optional)

  System users can be identified by specifying the IP addresses of Web clients for which firewall function-based access is permitted. In this case, specify the IP addresses for which communication through the firewall is permitted.

- Specifying the communication ports of Management Server and Administration Agent

  Block communication to the communication ports for Management Server and Administration Agent so that they cannot be accessed from the outside the firewall. If these ports can be accessed, an external non-administrator user might perform an unauthorized operation on the application server.

(c) Configuring the intrusion detection system

To monitor communication between external networks and a public port on the Web server (Cosminexus HTTP Server) within the application server, configure the following settings:

- Communication monitoring

  Configure communication monitoring to issue an alert to an administrator or equivalent if communication contains a known or suspected attack pattern. The linkage function between the intrusion detection system and the firewall can be configured to automatically block suspect communications.

- Monitoring for attacks against established SSL connections

  Basically, HTTPS-based communication cannot be monitored because it is encrypted. In this case, monitor for attacks against an established SSL connection that follow known HTTPS attack patterns.

- Monitoring of communication to non-public ports

  If communication is sent from external networks to a non-public port on the application server, it might be that the firewall has been broken into, for example, due to a configuration error. We recommend you configure the system to issue an alert if such an event occurs.

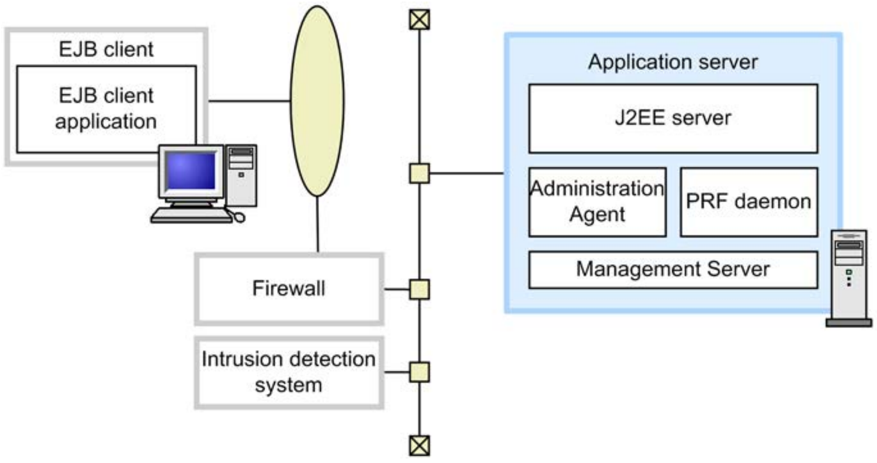
(3) Basic EJB client configuration

This section describes where to deploy a firewall and intrusion detection system for a basic EJB client configuration with a single application server.

When viewed from the network, the firewall should be deployed in front of the application server. In this configuration, an EJB client on the network can only access the application server via the firewall.

The figure below shows an example of a firewall and intrusion detection system deployment for a basic EJB client configuration.

Figure 4-5: Example firewall and intrusion detection system deployment for a basic EJB client configuration



(a) Configuring the application server

For the application server, configure the following settings:

- Specifying the address of the communication port for J2EE server management

  Specify the address at which the J2EE server management communication port can be accessed.

- Specifying the addresses of Management Server and Administration Agent

  Specify the addresses at which Management Server and Administration Agent can be accessed.

- Fixing the port numbers for access from the EJB client

  Configure the following port numbers so that the EJB client can communicate with the ports to use the application server:

  - CORBA naming service

    The port number is usually fixed. (The default port is 900.)

  - EJB container

    Because the port number used by EJB containers is not fixed, it is necessary to explicitly specify and fix the port number to be used by EJB containers. For details about port numbers that can be specified, see *3.16 TCP/UDP port numbers used by the application server's processes*, in the *uCosminexus Application Server System Design Guide*.

- Specifying the communication ports of Management Server and Administration Agent

  When specifying the communication ports for Management Server and Administration Agent, we recommend you do not use public ports so that they cannot be accessed from the outside of the firewall. If these ports can be accessed, an external non-administrator user might perform an unauthorized operation on the application server.

(b) Configuring the firewall

To control access between external networks and the application server, configure the following settings:

- Permitting access from external networks to the application server

  For communication between networks external to the firewall and the application server, only permit access to public ports such as those fixed for CORBA naming services or EJB containers. Depending on the system configuration, permit DNS or other communication as necessary.

- Limiting access based on the IP addresses of clients (optional)

  System users can be identified by specifying the IP addresses of clients for which firewall function-based access is permitted. In this case, specify the IP addresses for which communication through the firewall is permitted.

- Specifying the communication ports of Management Server and Administration Agent

  Block communication to the communication ports for Management Server and Administration Agent so that they cannot be accessed from the outside the firewall. If these ports can be accessed, an external non-administrator user might perform an unauthorized operation to the application server.

(c) Configuring the intrusion detection system

To monitor communication between external networks and a public port on the application server, configure the following settings:

- Communication monitoring

  Configure communication monitoring to issue an alert to an administrator or equivalent if communication contains a known or suspected attack pattern. The linkage function between the intrusion detection system and the firewall can be configured to automatically block suspect communications.

- Monitoring for attacks against established SSL connections

  Basically, HTTPS-based communication cannot be monitored because it is encrypted. In this case, monitor for attacks against an established SSL connection that follow known HTTPS attack patterns.

- Monitoring of communication to a non-public port

  If communication is sent from external networks to a non-public port on the application server, it might be that the firewall has been broken into, for example, due to a configuration error. We recommend you configure the system to issue an alert if such an event occurs.

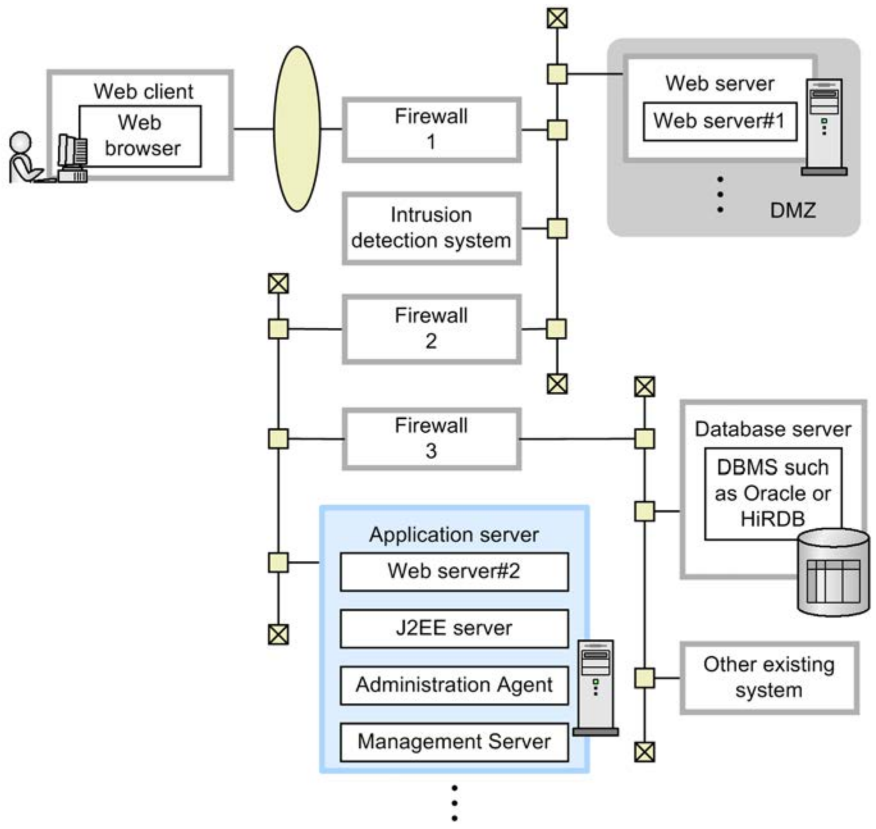
(4) Configuration separating each server layer by a firewall (application-centralized)

Depending on the scale of the system, a single system might consist of multiple application servers and other servers. In such a configuration, it is necessary to ensure security at each layer.

This section describes a configuration in which Web, application, and database servers are arranged into different layers, with all applications running on the same application server layer. This type of configuration is called an *application-centralized configuration*.

The figure below shows an example of a firewall and intrusion detection system deployment for an application-centralized configuration. In this configuration, a total of three firewalls are deployed, one for each server layer. In the DMZ, a Web server with a built-in reverse proxy module (reverse proxy server) is deployed.

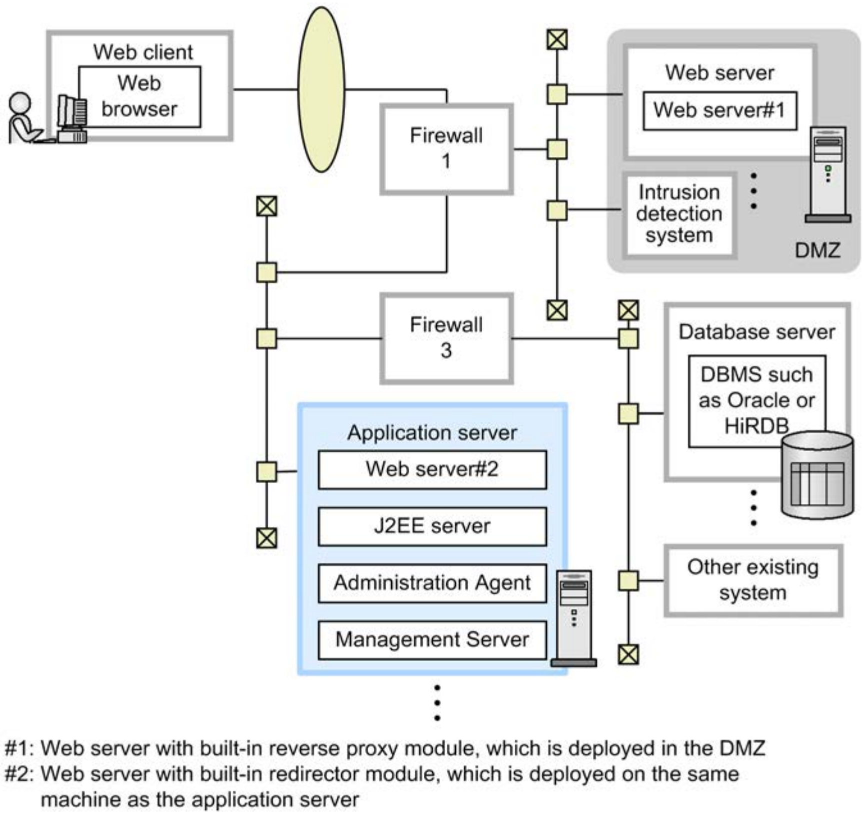
Figure 4-6: Firewall and intrusion detection system deployment in an application-centralized configuration



#1: Web server with built-in reverse proxy module, which is deployed in the DMZ
#2: Web server with built-in redirector module, which is deployed on the same machine as the application server

To reduce the number of firewalls, for example, to cut costs, a configuration as shown in the figure below is possible. In this example, firewall 2 is removed by consolidating the access controls to be performed by firewalls 1 and 2 into firewall 1.

Figure 4-7: Configuration with reduced firewalls


Web client
Web browser
Firewall 1
Web server
Web server#1
Intrusion detection system
DMZ
Firewall 3
Database server
DBMS such as Oracle or HiRDB
Application server
Web server#2
J2EE server
Administration Agent
Management Server
Other existing system
#1: Web server with built-in reverse proxy module, which is deployed in the DMZ
#2: Web server with built-in redirector module, which is deployed on the same machine as the application server

In this configuration, include the settings for firewall 2 into those for firewall 1.

(a) Configuring the application server

For the application server, configure the following settings:

- Specifying the address of the communication port for J2EE server management

  Specify the address at which the J2EE server management communication port can be accessed.

- Specifying the addresses of Management Server and Administration Agent

  Specify the addresses at which Management Server and Administration Agent can be accessed.

(b) Configuring each firewall

This configuration uses the following three firewalls:

- Firewall 1

This firewall controls access between external networks and the Web server (reverse proxy server) in the DMZ.

- Firewall 2

  This firewall controls access between the Web server (reverse proxy server) in the DMZ and the application server on the internal network.

- Firewall 3

  This firewall controls access between the application server and the database server.

Settings to be configured for each firewall are as follows.

■ Configuring firewall 1

Firewall 1 is used to control access between external networks and the Web server (reverse proxy server) in the DMZ. Configure the following settings:

- Permitting access from external networks to the Web server (reverse proxy server)

  For communication from networks external to firewall 1 to the Web server that is inside the application server, only permit access to public ports, such as HTTP/80 or HTTPS/443. Depending on the system configuration, permit DNS or other communication as necessary.

- Limiting access by the IP addresses of Web clients (optional)

  System users can be identified by specifying the IP addresses of Web clients for which firewall function-based access is permitted. In this case, specify the IP addresses for which communication to firewall 1 is permitted.

■ Configuring firewall 2

Firewall 2 is used to control access between the Web server and the application server. Configure the following settings:

- Permitting access from the Web server (reverse proxy server) in the DMZ to the Web server that is inside the application server on the internal network

  For communication from networks external to firewall 2 (DMZ) to the Web server that is inside the application server, only permit access to public ports, such as HTTP/80 or HTTPS/443. Depending on the system configuration, permit DNS or other communication as necessary.

- Limiting access based on the IP addresses of Web clients (optional)

  System users can be identified by specifying the IP addresses of Web clients for which firewall function-based access is permitted. In this case, specify the IP address of the reverse proxy server.

For other communication settings, permit access as appropriate according to the particular system configuration. It might be necessary to permit DNS communication, etc.

Reference note:

Reference

If a redirector module is built into a Web module that is within the DMZ, for example, and if a firewall is deployed between the Web server and the application server running the J2EE server, it is necessary to configure the following settings:

- Permitting access from the Web server to the application server

  For the Web server communication port for the J2EE server (or the port receiving requests from the redirector), permit communication to the default port number (8007).

■ Configuring firewall 3

Firewall 3 is used to control access between the application server and the database. This firewall serves as the last line of defense to protect the most important information in the system.

Configure the following settings:

- ■ Permitting access from the application server to the database server

  For communication from the application server to the database server, only permit access to a communication port for the database server. The communication port for the database server should be set up in accordance with the relevant settings for the database to be used. Note that it might be necessary to establish a connection from the database server to the application server.

For other communication settings, permit access as appropriate according to the particular system configuration. It might be necessary to permit DNS communication, etc.

(c) Configuring the intrusion detection system

To monitor communication between external networks and the public port for the Web server that is inside the application server, configure the following settings:

- ■ Communication monitoring

  Configure communication monitoring to issue an alert to an administrator or equivalent if communication contains a known or suspected attack pattern. The linkage function between the intrusion detection system and the firewall can be used to automatically block suspect communications.

- ■ Monitoring for attacks against established SSL connections

Basically, HTTPS-based communication cannot be monitored because it is encrypted. In this case, monitor for attacks against an established SSL connection that follow known HTTPS attack patterns.

- Monitoring of communication to non-public ports

  If communication is sent from external networks to a non-public port on the application server, it might be that the firewall has been broken into, for example, due to a configuration error. We recommend you configure the system to issue an alert if such an event occurs.

(5) Configuration separating each server layer by a firewall (application-distributed)

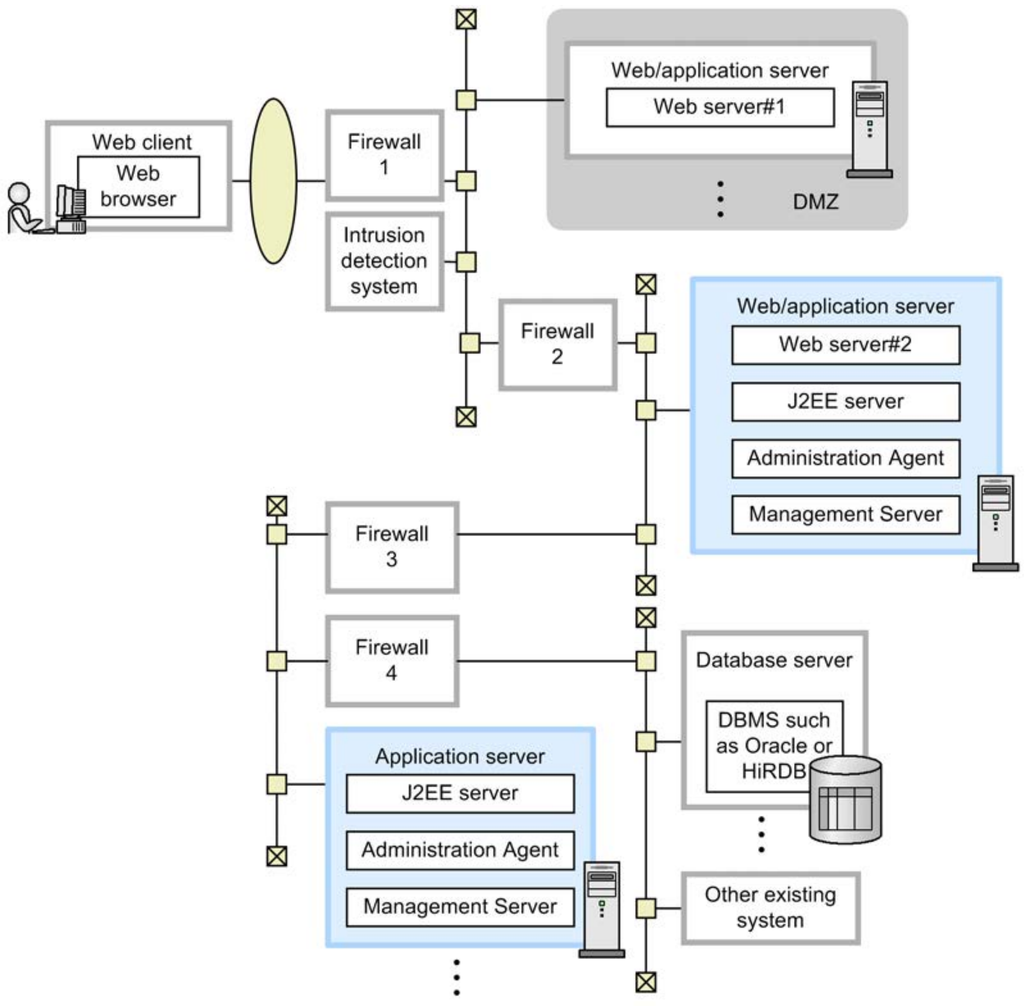
This section describes a configuration in which Web, application, and database servers are arranged into different layers and all applications are run on different application server layers. This type of configuration is called an *application-distributed configuration*.

The figure below shows an example of a firewall and intrusion detection system deployment for an application-distributed configuration. In this example, the Web applications run on the same layer as the Web server because the machine serving as the Web server also serves as an application server. Enterprise Bean runs on the application server that is set up on a separate machine from the Web server.

Administration is performed by instances of Management Server deployed on each host. Therefore, a management host is deployed to each layer.

In this configuration, a total of four firewalls are deployed: one in front of the DMZ and one for each server layer. In the DMZ, a Web server with a built-in reverse proxy module (reverse proxy server) is deployed.

Figure 4-8: Firewall and intrusion detection system deployment in an application-distributed configuration



#1: Web server with built-in reverse proxy module, which is deployed in the DMZ
#2: Web server with built-in redirector module, which is deployed on the same machine as the application server

(a) Configuring the Web/application server

For the application server machine that also serves as a Web server (Web/application server), configure the settings as shown below. Note that this application server machine also runs Web applications.

- Specifying the address of the communication port for J2EE server management

Specify the address at which the J2EE server management communication port can be accessed.

- Specifying the addresses of Management Server and Administration Agent

  Specify the addresses at which Management Server and Administration Agent can be accessed.

(b) Configuring the application server

For the application server running Enterprise Bean, configure the following settings:

- Specifying the address of the communication port for J2EE server management

  Specify the address at which the J2EE server management communication port can be accessed.

- Specifying the addresses of Management Server and Administration Agent

  Specify the addresses at which Management Server and Administration Agent can be accessed.

- Fixing the port number to be used by EJB containers

  It is necessary to explicitly specify and fix the port number to be used by EJB containers.

  For details about port numbers that can be specified, see *3.16 TCP/UDP port numbers used by the application server's processes*, in the *uCosminexus Application Server System Design Guide*.

(c) Configuring each firewall

This configuration uses a total of four firewalls:

- Firewall 1

  This firewall controls access between external networks and the Web server (reverse proxy server) in the DMZ.

- Firewall 2

  This firewall controls access between the Web server (reverse proxy server) in the DMZ and the Web/application server on the internal network.

- Firewall 3

  This firewall controls access between the Web/application server and the application server.

- Firewall 4

  This firewall controls access between the application server and the database server.

Settings to be configured for each firewall are as follows.

■ Configuring firewall 1

Firewall 1 is used to control access between external networks and the Web server (reverse proxy server) in the DMZ. Configure the following settings:

- Permitting access from external networks to the Web server (reverse proxy server)

  For communication from networks external to firewall 1 to the Web server that is inside the application server, only permit access to public ports, such as HTTP/80 or HTTPS/443. Depending on the system configuration, permit DNS or other communication as necessary.

- Limiting access based on the IP addresses of Web clients (optional)

  System users can be identified by specifying the IP addresses of Web clients for which firewall function-based access is permitted. In this case, specify the IP addresses for which communication to firewall 1 is permitted.

■ Configuring firewall 2

Firewall 2 is used to control access between external networks and the Web/application server on the internal network. Configure the following settings:

- Permitting access from the Web server (reverse proxy server) in the DMZ to the Web server that is inside the application server

  For communication from networks external to firewall 1 to the Web server that is inside the application server, only permit access to public ports, such as HTTP/80 or HTTPS/443. Depending on the system configuration, permit DNS or other communication as necessary.

- Limiting access based on the IP addresses of Web clients (optional)

  System users can be identified by specifying the IP addresses of Web clients for which firewall function-based access is permitted. In this case, specify the IP address of the reverse proxy server.

■ Configuring firewall 3

Firewall 3 is used to control access between the Web/application server and the application server. Configure the following settings:

- Permitting access from the Web/application server to the application server

  So that the Web/application server can use the J2EE server in the application server, permit communication to the following port numbers:

  - CORBA naming service

    The port number is usually fixed. (The default port is 900.)

  - EJB container

Because the port number used by EJB containers is not fixed, it is necessary to explicitly specify and fix the port number to be used by EJB containers.

For details about port numbers that can be specified, see *3.16 TCP/UDP port numbers used by the application server's processes*, in the *uCosminexus Application Server System Design Guide*.

- Permitting bidirectional access for transaction-related communication (if a global transaction is using transaction-context propagation)

  If a global transaction is using transaction-context propagation between the Web/application server and the application server, configure the following ports for bidirectional communication for both of the application servers:

  - Communication port for J2EE server transaction recovery (The default port is 20302.)
  - Smart agent communication port (The default port is 14000.)

- Other settings (optional)

  Depending on the system configuration, permit DNS and other communication as necessary.

Configuring firewall 4

Firewall 4 is used to control access between the application server and the database. This firewall serves as the last line of defense to protect the most important information in the system.

Configure the following settings:

- Permitting access from the application server to the database server

  For communication from the application server to the database server, only permit access to a communication port for the database server. The communication port for the database server should be set up in accordance with the relevant settings for the database to be used. Note that it might be necessary to establish a connection from the database server to the application server. Depending on the system configuration, permit DNS or other communication as necessary.

(d) Configuring the intrusion detection system

To monitor communication between external networks and the public port for the Web server within the application server, configure the following settings:

- Communication monitoring

  Configure communication monitoring to issue an alert to an administrator or equivalent if communication contains a known or suspected attack pattern. The linkage function between the intrusion detection system and the firewall can be used to automatically block suspect communications.

- Monitoring for attacks against established SSL connections

  Basically, HTTPS-based communication cannot be monitored because it is encrypted. In this case, monitor for attacks against an established SSL connection that follow known HTTPS attack patterns.

- Monitoring of communication to non-public ports

  If communication is sent from external networks to a non-public port on the Web/application server, it might be that the firewall has been broken into, for example, due to a configuration error. We recommend you configure the system to issue an alert if such an event occurs.

4.11.3 Using an SSL accelerator to process encrypted communication

This subsection describes how to use an SSL accelerator to process encrypted communication.

(1) The purpose of using an SSL accelerator

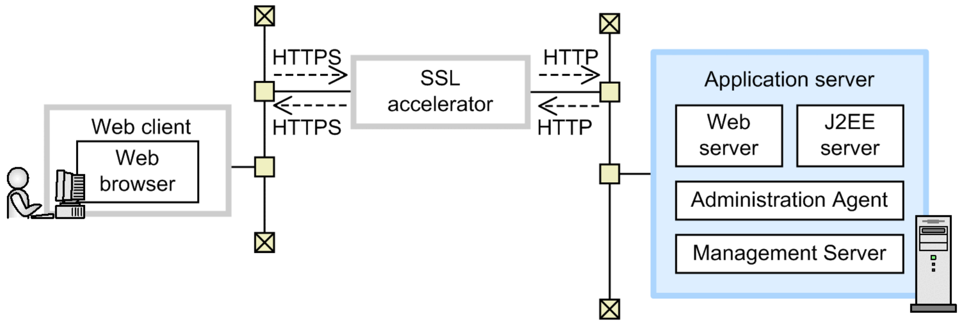
When considering security threats, one way to prevent leakage of application communication to third parties and alteration of such communication is the use of communication encryption. Using HTTPS for communication is one method of encryption. However, TLS/SSL-based communication, on which HTTPS is based, incurs a very high load.

An SSL accelerator is a piece of hardware dedicated to implementing HTTPS-encrypted communication processing without placing a load on the Web or application server. The correct deployment of an SSL accelerator will help accelerate encrypted communication without placing a load the Web or application server.

(2) Deploying an SSL accelerator

The following figure shows a configuration example using an SSL accelerator.

Figure 4-9: Configuration using an SSL accelerator



Communications sent by the Web client over HTTPS are decrypted by the SSL accelerator, and then passed on to the Web or application server over HTTP. Communications sent by the Web or application server over HTTP are encrypted by the SSL accelerator, and then passed on to the Web client.

When deploying an SSL accelerator, consider the following points:

- An SSL accelerator can be used as a firewall. In such a case, treat the SSL accelerator as part of your Web or application server.
- For Web server integration, when using an SSL accelerator, it might be necessary to configure additional redirector settings. For example, if a redirector is configured, a request forwarding destination or other information can be specified (only for HTTP/1.0 requests from Web clients). When using Management Server to set up the system, the system can be configured on the management portal by using **Environment Settings for Logical Server**. Open the Redirector Settings window for the logical Web server, and under **Gateway Specification Function Settings**, select **Yes** for **Use SSL accelerator**.

  When setting up the system without using Management Server, configure the setting in the following file:

  For Cosminexus HTTP Server

  The `JkGatewayHttpsScheme` key in the `mod_jk.conf` file

  For Microsoft IIS

  The `gateway_https_scheme` key in the `isapi_redirect.conf` file

  When setting up the system without using Management Server, see *9. Files Used for Web Server Integration*, in the *uCosminexus Application Server Definition Reference Guide*.
- To use an in-process HTTP server, you might need to configure additional Web container settings when using an SSL accelerator. For example, when these settings are configured, a request forwarding destination or other information can be specified (only for HTTP/1.0 requests from Web clients). When using Management Server to set up the system, the system can be configured on the management portal by using **Environment Settings for Logical Server**. Open the Other Settings window for the logical J2EE server, and under **Gateway Specification Function Settings**, select **Yes** for **Use SSL accelerator**.

  When setting up the system without using Management Server, configure the `webserver.connector.inprocess_http.gateway.https_scheme` key in the `usrconf.properties` file.

  When setting up the system without using Management Server, see *2.4. usrconf.properties (User property file for J2EE server)*, in the *uCosminexus Application Server Definition Reference Guide*.

4.11.4 Authenticating users from within applications

This subsection describes the authentication functionality available for applications to ensure security in Web client configurations.

(1) The purpose of authenticating users from within applications

When considering security threats, authenticating users when they execute applications helps to prevent system users from carrying out operations or acquiring information beyond the scope of the permission granted to them.

Application servers allow the use of the following three different protocols to ensure security via user authentication:

- HTTPS (Client authentication)
- HTTP (Basic authentication)
- HTTP (Form authentication)

Ensure security by using one of these protocols according to your particular purpose.

(2) Comparison between application-based user authentication methods

The table below shows where user authentication is carried out for each communication protocol, as well as the authentication engines used.

Table 4-8: Location of user authentication for each communication protocol and authentication engines used

Protocol used	Location of authentication	Authentication engine
HTTPS (Client authentication)	Cosminexus HTTP Server or Microsoft IIS	SSL
	SSL accelerator	SSL
HTTP (Basic authentication)	Cosminexus HTTP Server	HWS password file
		LDAP repository
	J2EE server (Web container)	Password file
HTTP (Form authentication)	J2EE server (Web container)	Password file
	J2EE server (integrated user management)	Integrated user management password file
		Database
		LDAP repository

Each protocol and authentication engine has certain features. Consider these features to select a suitable authentication method for the purpose of your system.

(a) Features of the protocols

The table below shows the features of the protocols that can be used for authentication on an application server system.

Table 4-9: Features of the protocols

Protocol used	Authentication interface flexibility	Ease of management from client	Network safety
HTTPS (Client authentication)	Available functions are limited to those provided by a Web browser.	Client certificates are required.	Due to encryption, authentication information is safe even if eavesdropping occurs.
HTTP (Basic authentication)	Available functions are limited to those provided by a Web browser.	Common user name/ password-based authentication is possible.	Passwords will be leaked in plain text or equivalent format. Therefore, this authentication is usually used along with the HTTPS encryption function (for server authentication only).
HTTP (Form authentication)	Different functions can be designed for each application.	Common user name/ password-based authentication is possible.	Passwords will be leaked in plain text or equivalent format. Therefore, this authentication is usually used along with the HTTPS encryption function (for server authentication only).

(b) Features of the authentication engines

The table below shows the features of the authentication engines.

Table 4-10: Features of the authentication engines

Engine type	Versatility	Maintainability	Effect on system configuration	Effect on performance
Password file	The format varies with the function used.	Each server or host has its own user information.	No special process is required for authentication.	Faster because no process or host communication occurs for authentication.

Engine type	Versatility	Maintainability	Effect on system configuration	Effect on performance
Database	Depending on the format, an existing user information database can be used.	User information management can be centralized.	A database server is required to store user information.	It takes additional time to access the database for authentication.
LDAP repository	Depending on the format, an existing user information repository can be used.	User information management can be centralized, including distributed user information.	An LDAP-compliant directory server is required to store user information.	It takes additional time to access the LDAP directory server for authentication.

Chapter

5. Integrated User Management-based Authentication

This chapter describes authentication using the integrated user management framework for the integrated user management of the system on the application server.

The functions explained in this chapter can be used in any J2EE server operation mode.

5.1 Organization of this chapter

The integrated user management framework is a framework that enables application user management based on Java Authentication and Authorization Service (JAAS) and single sign-on to multiple applications. This chapter describes integrated user management using the integrated user management framework.

The table below shows how the chapter is organized.

Table 5-1: Organization of this chapter (Integrated user management)

Part	Title	Relevant information
Description	Overview of integrated user management	5.2
	User authentication mechanism based on Cosminexus standard login modules	5.3
	Sessions managed in integrated user management	5.4
	Use of single sign-on	5.5
	Use of custom login modules	5.6
	Management of user information	5.7
	API provided by the integrated user management framework	5.8
Implementation	Implementation of user authentication based on the integrated user management framework	5.9
	Implementation of API-based user authentication	5.10
	Implementation of tag library-based user authentication	5.11
	Implementation of user authentication when using the session failover functionality	5.12
	Implementation of custom login module-based user authentication	5.13
Setup	Procedures to set up the integrated user management function	5.14
	Determination of realm names	5.15
	LDAP directory server setup	5.16
	Registration of user information	5.17
	Creation of encryption key files (When using single sign-on)	5.18

Part	Title	Relevant information
	Registration of user information (When using single sign-on)	5.19
	Creating configuration files	5.20
	JavaVM property setup	5.21
	Deployment of files	5.22

Note: This chapter does not include information on operation or precautions.

5.2 Overview of integrated user management

Integrated user management is the function that enables the integrated management of users who try to log into the Cosminexus system.

To enable integrated user management, Cosminexus provides the following function.

- Integrated user management framework

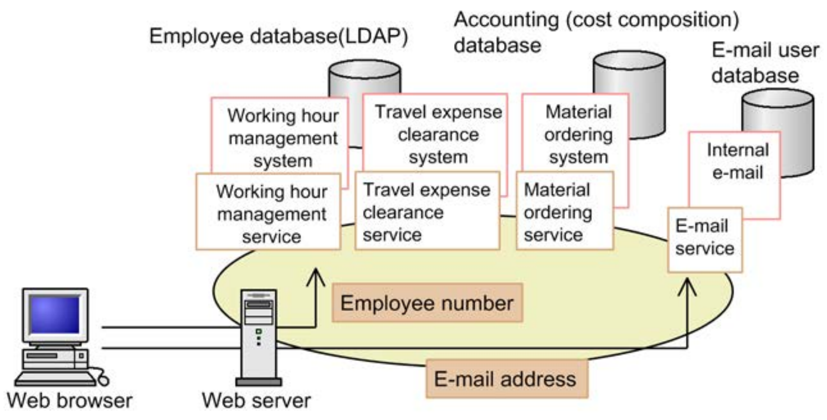
This is the framework that enables integrated user management-based authentication. It provides the application program interface based on the Java standard specification (JAAS).

5.2.1 Purpose of integrated user management

Traditionally, business systems deployed at a company manage their users in their own unique ways in order to meet the business requirements. For example, the working hour management system identifies the employees registered in the employee database as the system users, and the material ordering system identifies the cost management departments as the system users. Recently, the advancement of Intranet technology eliminates the need to install system-specific client programs, and enables users to use these business services through a Web browser. However, operation for user authentication in accordance with the system-specific user management procedures is still necessary. For example, email IDs, employee numbers, and department codes must be used to log into the internal email system, for working hour management, and for material ordering services, respectively. It is difficult to provide a new service that integrates these system services and simplifies access to them.

The integrated user management framework is a user management framework that uses JavaEE technologies to integrate these services. The following figure shows an example of the integrated user management framework.

Figure 5-1: Example of the integrated user management framework


Employee database(LDAP)
Accounting (cost composition) database
E-mail user database
Working hour management system
Travel expense clearance system
Material ordering system
Internal e-mail
Working hour management service
Travel expense clearance service
Material ordering service
E-mail service
Employee number
E-mail address
Web browser
Web server

5.2.2 User management and user mapping using realms

This section describes the concepts used for integrated user management: Realms and user mapping.

(1) Realms

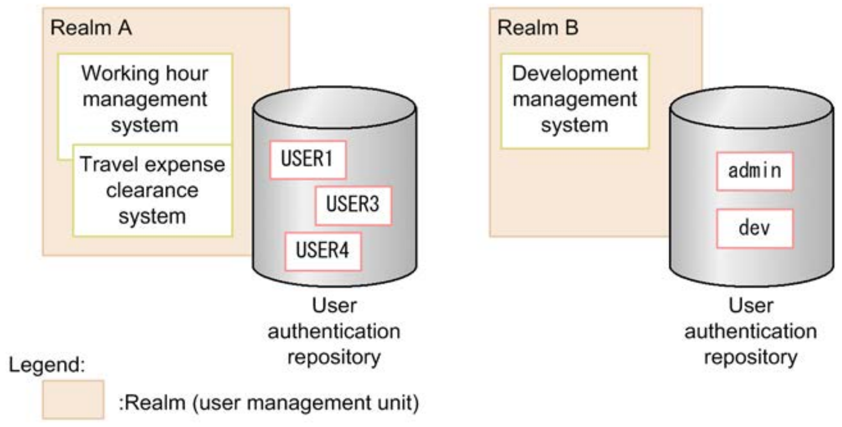
A realm is the extent to which the same authentication policies are applied. The business service application identifies the service users based on the service requirements. The process to identify users is generally called the *authentication process*. The authentication process is categorized by the authentication mechanism to identify service users and the user authentication database for the purpose intended (user authentication repository). System administrators are responsible to determine the authentication policies, such as which authentication mechanism(s) should be used and which users should registered in the user authentication database.

Unlike establishing the system providing a single service, it is essential to examine what authentication policies should applied to which extent to operate the system when establishing a system providing a number of services. In the Web system, the extent to which the same authentication policies are applied is called a *realm*, and the name used to identify a realm is called the *realm name*. The user authenticated in a realm has an identifier (user ID) that uniquely identifies him/her in that realm.

The applications that have the same user management requirements can be administered in a single realm. Ideally, all the newly established services should be integrated into a single realm so that they can be easily controlled based on the security attributes of the authenticated users. In reality, however, such systems are rare. There are many realms in the enterprise system, such as email IDs for the internal email, employee numbers for the human resources system, and department codes for the material ordering system, and they are separately managed.

When using Cosminexus to integrate these services, the administrators must consider the consolidation of realms by analyzing why they are necessary. Reducing the number of realms to as few as possible can simplify management. The following figure shows an example of realm management.

Figure 5-2: Example of realm management


Realm A
Working hour management system
Travel expense clearance system
USER1
USER3
USER4
User authentication repository
Realm B
Development management system
admin
dev
User authentication repository
Legend:
:Realm (user management unit)

(2) User mapping between realms

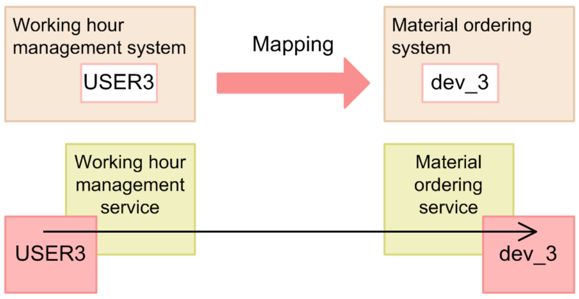
The business service application asks the end user to enter the user ID and password that are used for authentication. The authentication is kept until the user logs out. The user is then asked to enter the user ID and password each time he or she tries to log into the application, which uses different user IDs and passwords for authentication. In short, users are required to be authenticated to access the application that is managed in a different realm from the application that has already authenticated them.

If all J2EE applications know how the user who has logged into a realm is handled by other realms, the user does not need to repeatedly enter the user ID and password. To address this matter, the integrated user management framework uses user mapping, which maps users who have logged into a realm to other realm users.

User mapping automatically authenticates the user who has been authenticated in a realm when the successful authentication status is shared with other realms. To use user mapping, the Cosminexus system administrators should map users to realms and store the mapping information in the system in advance.

In the following example of user mapping, the user who has been authenticated as **USER3** in the working hour management service is pre-mapped to the **dev_3** user of the material ordering service. As a result, the user who has been authenticated as **USER3** in the working hour management service is automatically authenticated as **dev_3** in the material ordering service without login operation.

Figure 5-3: Example of user mapping


Working hour management system
USER3
Mapping
Material ordering system
dev_3
Working hour management service
USER3
Material ordering service
dev_3

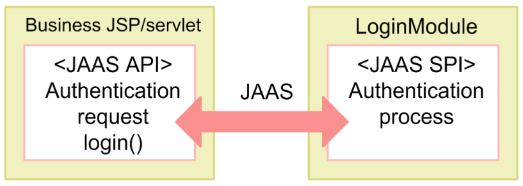
5.2.3 Overview of Java Authentication and Authorization Service (JAAS)-based user authentication

Traditionally, unique interfaces have been designed to invoke the user authentication mechanism of business systems through the Web. The Web business services built from scratch by using JavaEE technologies have also used unique authentication mechanisms provided by application servers. These interface differences have been a major hurdle in integrating services. To address the hurdle, JAAS (Java Authentication and Authorization Service) 1.0 was finalized as the Java standard user authentication specifications. It is now used as the user authentication standards for the Web business services developed using JavaEE technologies.

(1) Association between the integrated user management framework and individual user management

JAAS specifies the interface regarding which application sends authentication requests (API: Application Programming Interface) and the interface that receives and processes authentication requests (SPI: Service Provider Interface). The module that processes authentication is called the login module. The following figure shows the association between API and SPI.

Figure 5-4: Association between API and SPI

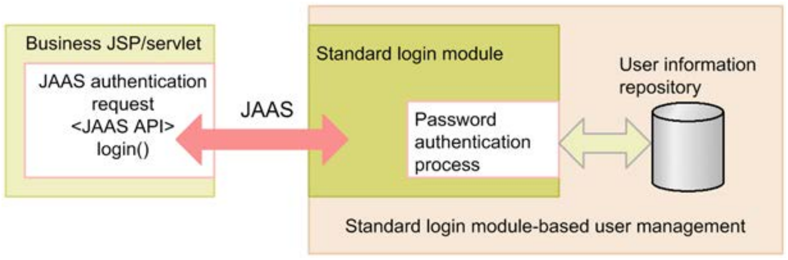


Cosminexus provides Cosminexus standard login modules, which perform password authentication by default. The Cosminexus standard login modules are used to manage users of the J2EE application created with the JSP/servlet, which uses the JAAS API.

Application developers no longer need to develop authentication modules by themselves if they use the Cosminexus standard login modules. As modules are stackable, it is easy to add an enhanced authentication module that works together with the Cosminexus standard login modules. If the application requires a completely unique authentication function, the Cosminexus standard login module can be easily replaced by a custom authentication module. As a result, application programs that use JAAS for user authentication can be effortlessly integrated by using Cosminexus.

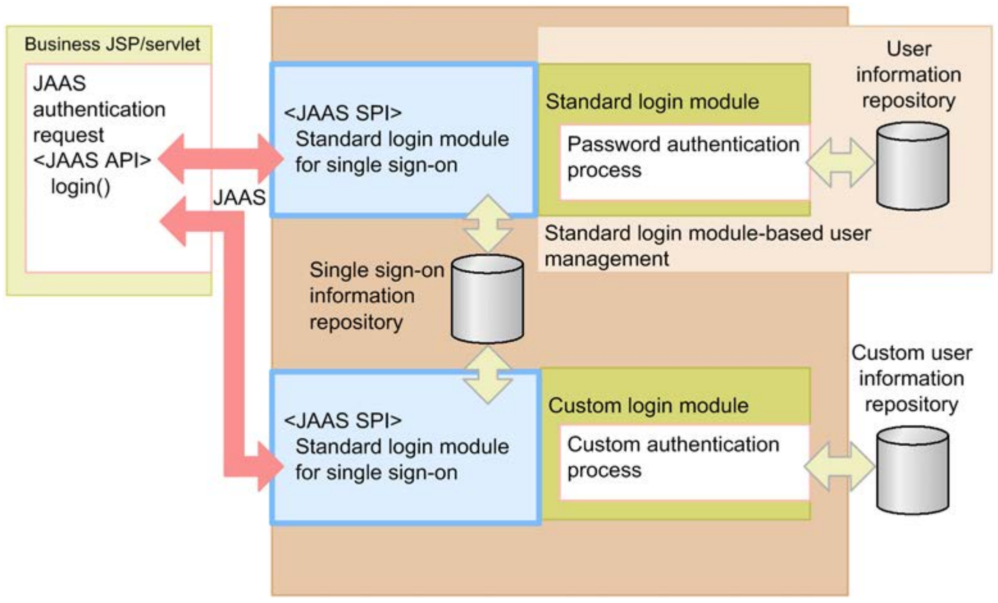
This is called JAAS-based user management. The following figure shows an overview of JAAS-based user management. Note that the user information repository in the figure is the repository that stores the user information needed for the authentication process.

Figure 5-5: Overview of JAAS-based user management


Business JSP/servlet
JAAS authentication request <JAAS API> login()
JAAS
Standard login module
Password authentication process
User information repository
Standard login module-based user management

When the JAAS-based user management is used in accordance with the integrated user management framework, the user mapping function can be used, which maps the user authenticated by one business service to the user of another business service and requests authentication. When a unique user information repository is used for a business service, creating a custom login module can implement single sign-on including that business service. The following figure shows an overview of single sign-on with user mapping. Note that the single sign-on repository in the figure is the repository that stores the mapping information needed for single sign-on.

Figure 5-6: Overview of the user mapping function


Business JSP/servlet
JAAS authentication request
<JAAS API>
login()
JAAS
<JAAS SPI>
Standard login module for single sign-on
Standard login module
Password authentication process
User information repository
Standard login module-based user management
Single sign-on information repository
<JAAS SPI>
Standard login module for single sign-on
Custom login module
Custom authentication process
Custom user information repository

(2) Overview of Cosminexus standard login modules

The Cosminexus standard login module is the LoginModule interface implementation-class contained in the `javax.security.auth.spi package`. It can be used differently depending on the implemented authentication method.

The Cosminexus standard login module has the following features.

User authentication can be done by using the existing user information (LDAP information or database information).

The Cosminexus standard login module allows the LDAP directory server or database (RDB) to be used as the repository that stores the user information used for user authentication.

When the LDAP directory server is used, Cosminexus specifies the DIT (Directory Information Tree) structure of the standard user management repository. If LDAP has already been deployed, the information can be available through simple customization. DIT is the LDAP mechanism used to manage the user and organization information in the tree structure. For details, see *5.2.4 Management method of user information used for integrated user management*.

Certificates or passwords can be used for authentication depending on the type of the Cosminexus standard login module.

The user information can be referenced.

When the user authentication is successful, the information about the login user can be referenced.

JAAS specifies that Credentials must be assigned to the Subject when the user is successfully authenticated. It also specifies the general methods used by the requesting application to reference the authenticated user information through the `java.util.set` interface (`getPublicCredentials` and `getPrivateCredentials` methods).

The Cosminexus standard login module provides the interface used to reference the user information. The user information is specified in the form of a Credential as the object that the UserAttributes interface, provided by Cosminexus, handles in accordance with the configuration. The application fetches this object by using the standard interface, and obtains the user information by specifying the attribute name in the `getAttribute` method of the object. Otherwise, specifying the alias can obtain the information.

Single sign-ons can be implemented.

The Cosminexus standard login module supports single sign-on.

To implement single sign-on, the LDAP directory server is needed regardless of the type of repository that manages the user information.

5.2.4 Management method of user information used for integrated user management

This section describes the management method of user information used for integrated user management.

Integrated user management uses LDAP or database as the repository that stores user information. In the LDAP directory server, DIT is used to manage the user and organization information. The users and realms are managed as the DIT entries in the LDAP directory server used in the integrated user management framework. The entry is the information that constitutes DIT and is the node of DIT. Each entry is identified by a DN (Distinguished Name).

Cosminexus specifies the DIT structure of the standard user management repository stored in the LDAP directory server used in the integrated user management framework. There are two types of repositories used in the integrated user management framework:

- User information repository

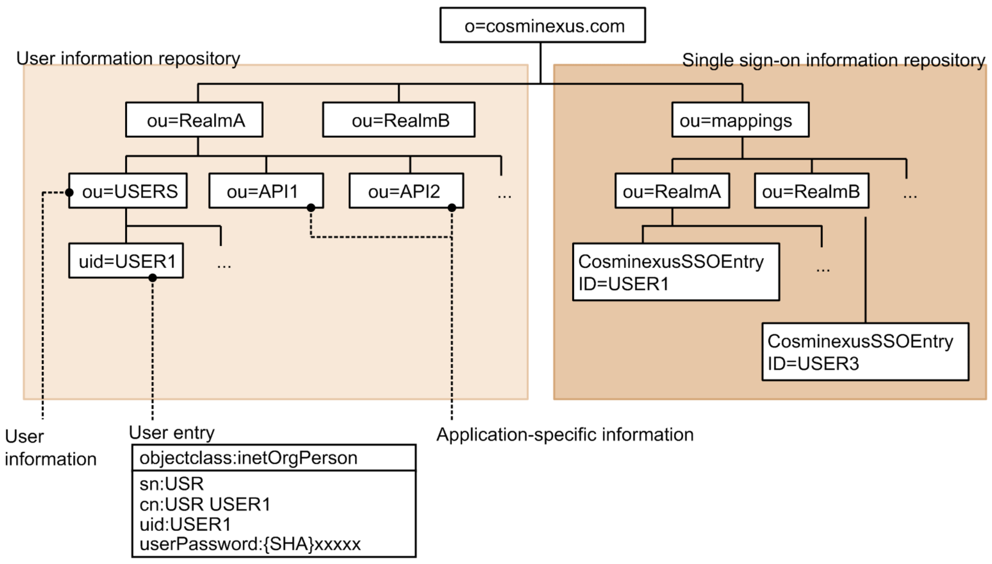
  This stores the user information.

- Single sign-on information repository

  This stores the authentication and mapping information used by the systems to authenticate single sign-on users by performing user mapping in the integrated user management framework.

These repositories have the directory structures as shown in the following figure.

Figure 5-7: Repository DIT structure in the integrated user management framework

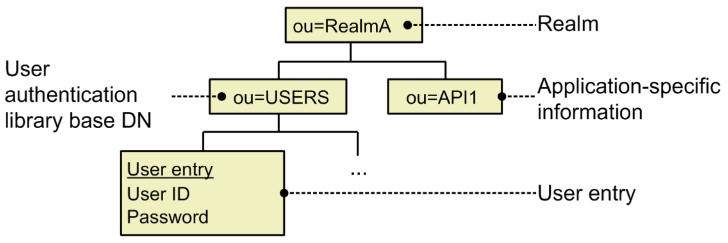


A description of these repositories follows:

(1) User information repository (in the LDAP directory server)

The user information used for user authentication is stored in the user information repository. The integrated user management framework authenticates the user based on the user information stored in the user information repository of the LDAP directory server, and then passes the authenticated user information to the application. The user authentication library is used to reference the user information in the user information repository. The following figure shows the DIT structure of the user information repository.

Figure 5-8: DIT structure of the user information repository


ou=RealmA
Realm
User authentication library base DN
ou=USERS
ou=API1
Application-specific information
User entry
User ID
Password
...
User entry

Create a user information repository for each managed realm.

(a) Realms

Specify a JAAS-based user management realm name. The realm name must conform to the guidelines specified in the following table:

Table 5-2: Realm name guidelines

Type of information	Meaning	Grammar
Realm name	The identifier that indicates the scope of user management	A string of alphanumeric characters Not case sensitive Specify the name that can be used in the DN.

Note: A string of alphanumeric characters means a sequence of alphabetical characters (A to Z and a to z) and numbers (0 to 9). Use ASCII characters in realm names. (The program does not check the grammar.)

(b) Application-specific Information

Use this repository to store the information that is specific to the application using the realm, when necessary. This does not contain information necessary for the integrated user management framework.

(c) User authentication library base DN

This is an upper entry of the user entry belonging to the realm. Each user entry belonging to the realm must be below this level. If the user entry is not immediately below this entry, the `com.cosminexus.admin.auth.ldap.search.scope` option in `ua.conf` (the integrated user management configuration file) must be changed. The information specified in this entry must also be specified in `jaas.conf` (the JAAS configuration file). For details about the configuration files, see *14.2 jaas.conf (JAAS configuration file).*

(d) User entry

This defines the user information. In the user authentication library, the attributes listed in the following table must be contained in the user information.

Table 5-3: Necessary attributes in user information

Attribute name	Description	Necessity
User ID	Stores the user ID; the attribute must be a character string (such as `cis`). By default, the `uid` attribute name is used.	Required

Attribute name	Description	Necessity
Password	Stores the password; the attribute is binary. The values are either stored in plain text or encrypted. If no values are specified for this attribute, the account will be invalid. By default, the `userPassword` attribute name is used.	Optional
Other attributes	Defined by each application	Follow the application specifications.

The user ID and password attribute names can be changed in `jaas.conf` (the JAAS configuration file).

(e) Notes

The directory structure of the user information repository conforms to the DIT structure recommended in the JAAS-based user management. When a different structure is used for management, the user entry that meets the following conditions must be created under the "user authentication library base DN".

- The user ID and password must be in the same object class.
- The password must be binary. In addition, it is recommended that the password values be encrypted. When the values are stored in plain text, export them to an `.ldif` file format, encrypt the file with the `convpw` command, and then register the encrypted values as the password. For details about the `convpw` command, see *convpw (Password encryption)*.
- Although the user ID and password can have the same attribute name in the multi-value format, the Cosminexus system allows only one attribute name in one object class. If there is more than one attribute name, the one detected first is used.
- The user ID attribute used as the user ID must be unique in the realm (at all the levels below the user information repository base DN).

The user information repository base DN and the attribute names of the user ID and password are specified in `ua.conf` (the integrated user management configuration file). To learn more about `ua.conf`, see *14.3 ua.conf (integrated user management configuration file).*

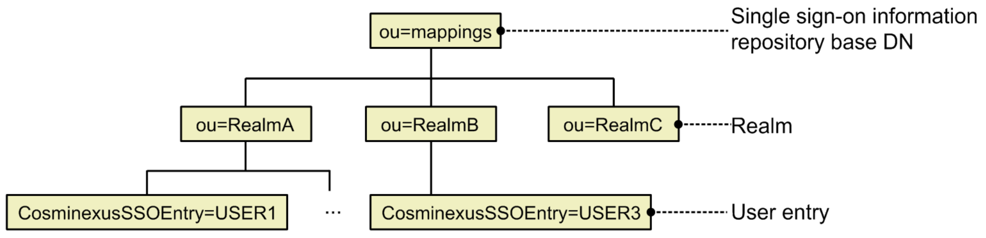
(2) User information repository (in the database)

The integrated user management framework authenticates users based on the user information stored in the database. In the database, ensure that the passwords can be retrieved based on the user IDs.

(3) Single sign-on information repository

The single sign-on information repository stores the system authentication and mapping information used to authenticate single sign-on users. The integrated user management framework implements single sign-on by mapping users based on the user information stored in the single sign-on information repository of the LDAP directory server. The user information in the single sign-on information repository can be referenced by using the single sign-on library. The following figure shows the DIT structure of the single sign-on information repository.

Figure 5-9: DIT structure of the single sign-on information repository



(a) Single sign-on information repository base DN

This is the uppermost entry of the DIT, which manages the necessary information for single sign-on. This entry is specified in `ua.conf` (the integrated user management configuration file). To learn more about `ua.conf`, see *14.3 ua.conf (integrated user management configuration file)*. The file is not case sensitive. The specified values are set to the `ou` attribute of the standard object class, `organizationalUnit`.

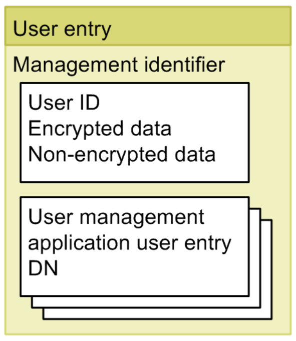
(b) Realms

The user information is managed per realm. The realm name in the single sign-on information repository is not case sensitive. The specified values are set to the `ou` attribute of the standard object class, `organizationalUnit`.

(c) User entry

This is the entry used to store the user authentication information and destination used for user management and the applications that can be accessed via single sign-on. The following figure shows the user entry structure.

Figure 5-10: User entry structure



Administration identifier

This is the identifier that is automatically set when a user entry is registered in the single sign-on library.

User ID

A unique user ID is specified for each realm by using a character string. The user ID is case sensitive.

Encrypted data

This stores the data that needs to be encrypted at the time of registration. For example, the password is encrypted when stored in this attribute.

Non-encrypted data

This stores the necessary information for authentication other than the user ID and the encrypted data that does not need to be encrypted. For example, the user group ID is stored here.

DN of the user entry of the application with user management

This stores the destination (DN) of the user authentication information of the application with user management, which the user can access via single sign-on. It can have more than one value.

5.2.5 Validity period of user authentication and the inheritance of authentication states

The validity period of JAAS user authentication is from when the `login` method succeeds to when the `logout` method is invoked.

The J2EE Web application uses the HttpSession object to control virtual sessions with the users. To handle a number of HTTP protocol communications as a series of sessions, it is necessary to associate the HttpSession object with the requesting user by modifying the cookie or the URL in the Web application.

In the integrated user management framework, the successful user authentication status is stored in the HttpSession object. If a request uses the same HttpSession object and is made in the same realm, the end user can skip entering the user authentication information (user ID and password), as the authentication information entered at the first login and the authentication state are automatically inherited.

However, this will not work when the login modules are used in the order listed in the following table. For the functional details of each login module, see *5.3 User authentication mechanism based on Cosminexus standard login modules*.

Table 5-4: Order of use that does not allow end users to skip entering authentication information

Order of use	Used login module
First login	• DelegationLoginModule (when a custom login module or WebCertificateLoginModule is invoked) • WebCertificateLoginModule • WebSSOLoginModule (when a custom login module or WebCertificateLoginModule is invoked)
Second and subsequent logins	• WebPasswordLDAPLoginModule

When the login modules are used in the order listed in the following table, the password must be stored in the integrated user management session by the login module used at the first login so that the end user can skip entering the authentication information at the second and subsequent logins using WebPasswordLDAPLoginModule. In addition, the password for the second and subsequent logins using WebPasswordLDAPLoginModule must be the same as the first login. The administrators can use `jaas.conf` (the JAAS configuration file) and `ua.conf` (the integrated user management configuration file) to specify whether to store passwords and whether to encrypt the stored passwords. For details about the specification method in `jaas.conf`, see *14.2 jaas.conf (JAAS configuration file)*. For details about the specification method in `ua.conf`, see *14.3 ua.conf (integrated user management configuration file)*.

Table 5-5: Order of use that allows end users to skip entering authentication information at the second and subsequent logins

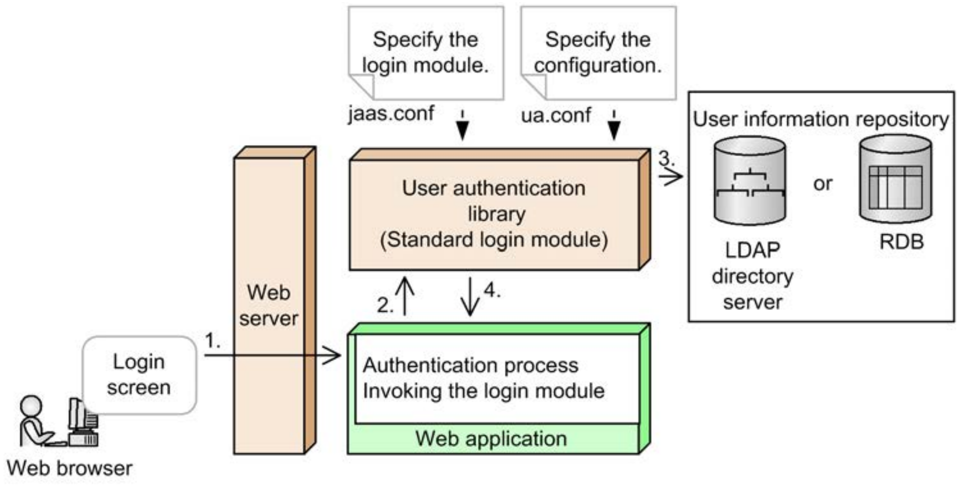
Order of use	Used login module
First login	• WebPasswordLoginModule • WebPasswordJDBCLoginModule • WebPasswordLDAPLoginModule
Second and subsequent logins	• WebPasswordLDAPLoginModule

When the stored passwords are not encrypted, they may be leaked if the memory contents used by the J2EE server are referenced by illegal means. If the session failover functionality is active, the session information including passwords flows throughout the network, so passwords can be intercepted. While encryption reduces the risk of password leakage, it affects the performance. Determine whether to encrypt passwords by taking into account the security and performance requirements. Note that the Triple DES encryption algorithm is used to encrypt passwords.

5.2.6 Integrated user management process flow

The following figure illustrates the process flow when using integrated user management.

Figure 5-11: Process flow when using integrated user management


Specify the login module.
Specify the configuration.
jaas.conf
ua.conf
User information repository
3.
User authentication library (Standard login module)
or
LDAP directory server
RDB
Web server
2.
4.
1.
Login screen
Authentication process Invoking the login module
Web application
Web browser

A description of this process is as follows:

1. A login request is sent through the Web browser.
2. The login module is invoked by the Web application to perform authentication process.
3. The Cosminexus standard login module is used to authenticate the user. The login module used for authentication and its configuration are defined in `jaas.conf` (the JAAS configuration file) or `ua.conf` (the integrated user management configuration file). The necessary information for authentication is retrieved from the user information repository in the LDAP directory server or the database.
4. The authentication result is returned to the Web application.

Implementing the authentication process requires the users who set up the system and the application developers to configure the system and develop the application.

What the users who set up the system must do

- Define the used login modules, repositories, and their configuration information in `jaas.conf` (the JAAS configuration file) and `ua.conf` (the integrated user management configuration file). To implement single sign-on, define the single sign-on parameters in `ua.conf` as well.
- Create the encryption key file when implementing single sign-on.
- Register the user information in the user information repository.
- If the Component Container administrators and the users in the integrated user management group use `jaas.conf` (the JAAS configuration file) and `ua.conf` (the integrated user management configuration file) stored under *Cosminexus installation directory*`/manager/config` in the UNIX environment, set the appropriate access permissions in these configuration files in advance.

Use a text editor, etc., to edit `jaas.conf` and `ua.conf`.

To learn more about the integrated user management configurations, see *5.20 Creating configuration files*.

What the application developers must do

- Use the JAAS API and the integrated user management API and JSP tag library provided by Cosminexus to create the authentication process program that invokes the login modules.
- Create a custom login module to authenticate users in a specific way to the application.
- When necessary, create an implementation class to enhance passwords at the time of password authentication.

For details about implementing user authentication based on the integrated user management framework, see *5.9 Implementation of user authentication based on the integrated user management framework*.

5.3 User authentication mechanism based on Cosminexus standard login modules

The integrated user management framework provides the JAAS-based Cosminexus standard login modules. The Cosminexus standard login modules allow for the integrated user management of the Cosminexus system without creating custom authentication modules.

5.3.1 Types and functions of Cosminexus standard login modules

The Cosminexus standard login modules provided by the integrated user management framework are grouped into the following two types:

- ■ Login modules used to authenticate users

  The following four login modules belong to this type.

  - `WebPasswordLoginModule`

    This login module uses passwords to authenticate users.

  - `WebCertificateLoginModule`

    This login module uses client certificates to authenticate users.

  - `WebPasswordLDAPLoginModule`

    This login module uses the LDAP directory server's authentication function to authenticate users.

  - `WebPasswordJDBCLoginModule`

    This login module is used when the database is already used as the user information repository.

- ■ Login modules used to invoke custom application login modules

  The following two login modules belong to this type.

  - `DelegationLoginModule`

    This login module is used to invoke custom login modules. It does not support single sign-on.

  - `WebSSOLoginModule`

    This login module is used for single sign-on. It invokes other login modules such as Cosminexus standard login modules and custom login modules.

  DelegationLoginModul is used to invoke custom login modules when single sign-on is not used. WebSSOLoginModule is used to invoke other Cosminexus standard login modules or custom login modules when single sign-on is used. For example, to provide single sign-on access to the application that requires password authentication, invoke WebSSOLoginModule and then WebPasswordLoginModule from that module.

The following table lists the function of each login module.

Table 5-6: Login module function list

Function				Type					
				P	C	L	J	D	S
Used repository	LDAP directory server			A	A	A	--	--	A
	Database (JDBC)			--	--	--	A	--	--
Authentication method	X509 certificate			--	A	--	--	--	--
	Password authentication			A	--	A[#1]	A	--	--
		Type that can store passwords	Binary (byte [])	A	--	--	A[#2]	--	--
			Character string	--	--	--	A[#3]	--	--
		Encryption algorithm used to compare/store passwords	Plain text	A	--	--	A	--	--
			SHA-1	A	--	--	A	--	--
			MD5	A	--	--	A	--	--
			Encryption enhancement	A	--	--	A	--	--
			Triple DES	--	--	--	--	--	A
Miscellaneous	Setting Principal objects			A	A	A	A	--	--
	Obtaining user attributes			A	A	A	--	--	--
	Registering the user ID and realm name of the user logging in the integrated user management session at the time of login (which are removed at the time of logout)			A	A	A	A	A #4	A[#4]
	Invoking custom login modules			--	--	--	--	A	A

Legend:

P: WebPasswordLoginModule

C: WebCertificateLoginModule

L: WebPasswordLDAPLoginModule

J: WebPasswordJDBCLoginModule

D: DelegationLoginModule

S: WebSSOLoginModule

A: Available

-: Not available

#1: The type that can store passwords and the encryption algorithm depend on the LDAP directory server.

#2: The mappable SQL data type can be specified in the byte [] type (VARBINARY/LONGVARBINARY).

#3: The mappable SQL data type can be specified in the String [] type (CHAR/VARCHAR/LONGVARCHAR).

#4: Registration is performed when the conditions are met.

5.3.2 WebPasswordLoginModule

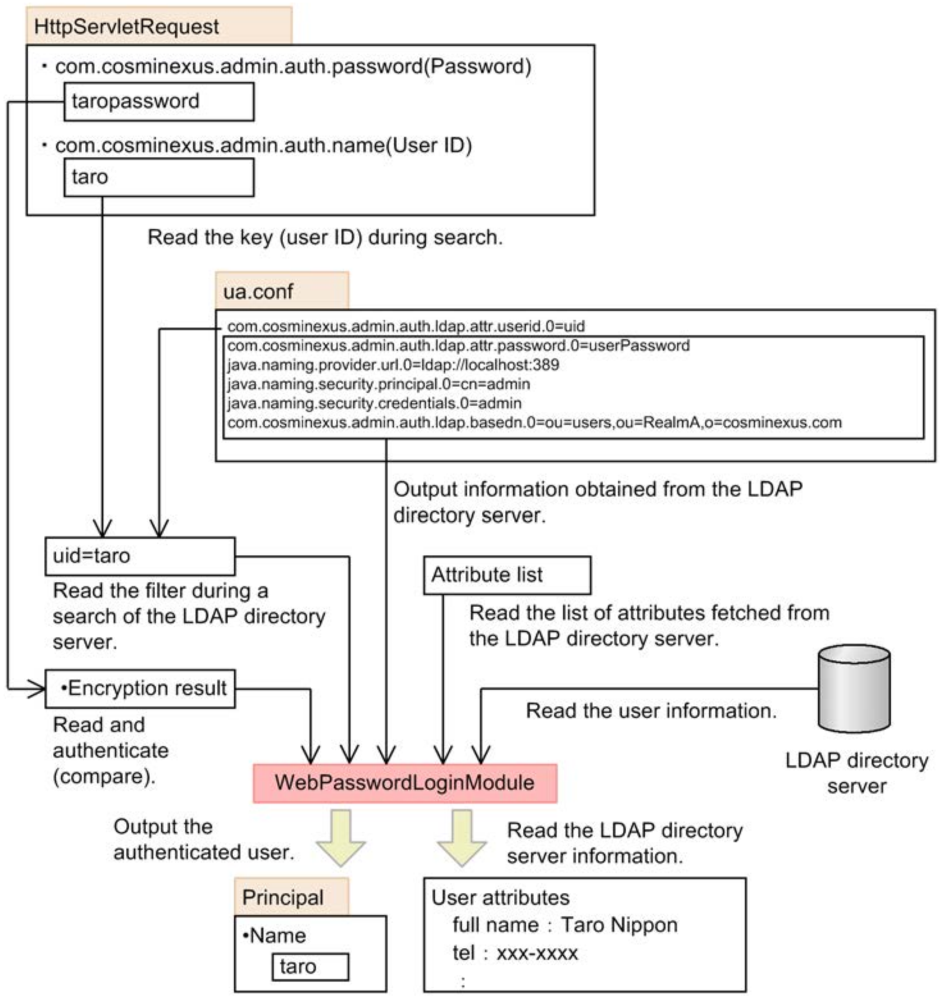
WebPasswordLoginModule is the login module that performs password authentication when there is a user information repository in the LDAP directory server.

It uses the entered user ID and password to retrieve the password from the user information stored in the LDAP directory server.

To use this module, pre-specify the definition to connect to the LDAP directory server and the attribute names used to retrieve entries (`uid` and `userPassword`) in `ua.conf` (the integrated user management configuration file).

WebPasswordLoginModule reads this file, obtains the user ID from HttpServletRequest to search for the password in the LDAP directory server, and then uses the password to perform password authentication. When authentication is successful, it returns the user attributes. The following figure shows an overview of WebPasswordLoginModule.

Figure 5-12: Overview of WebPasswordLoginModule


HttpServletRequest
• com.cosminexus.admin.auth.password(Password)
taropassword
• com.cosminexus.admin.auth.name(User ID)
taro
Read the key (user ID) during search.
ua.conf
com.cosminexus.admin.auth.ldap.attr.userid.0=uid
com.cosminexus.admin.auth.ldap.attr.password.0=userPassword
java.naming.provider.url.0=ldap://localhost:389
java.naming.security.principal.0=cn=admin
java.naming.security.credentials.0=admin
com.cosminexus.admin.auth.ldap.basedn.0=ou=users,ou=RealmA,o=cosminexus.com
Output information obtained from the LDAP directory server.
uid=taro
Read the filter during a search of the LDAP directory server.
Attribute list
Read the list of attributes fetched from the LDAP directory server.
•Encryption result
Read and authenticate (compare).
Read the user information.
LDAP directory server
WebPasswordLoginModule
Output the authenticated user.
Read the LDAP directory server information.
Principal
•Name
taro
User attributes
full name : Taro Nippon
tel : xxx-xxxx
:

5.3.3 WebCertificateLoginModule

WebCertificateLoginModule is the login module that uses the client certifications authenticated by the Web server to authenticate users.

Note:

A Web server with SSL functionality is required to use `WebCertificateLoginModule` in the integrated user management framework.

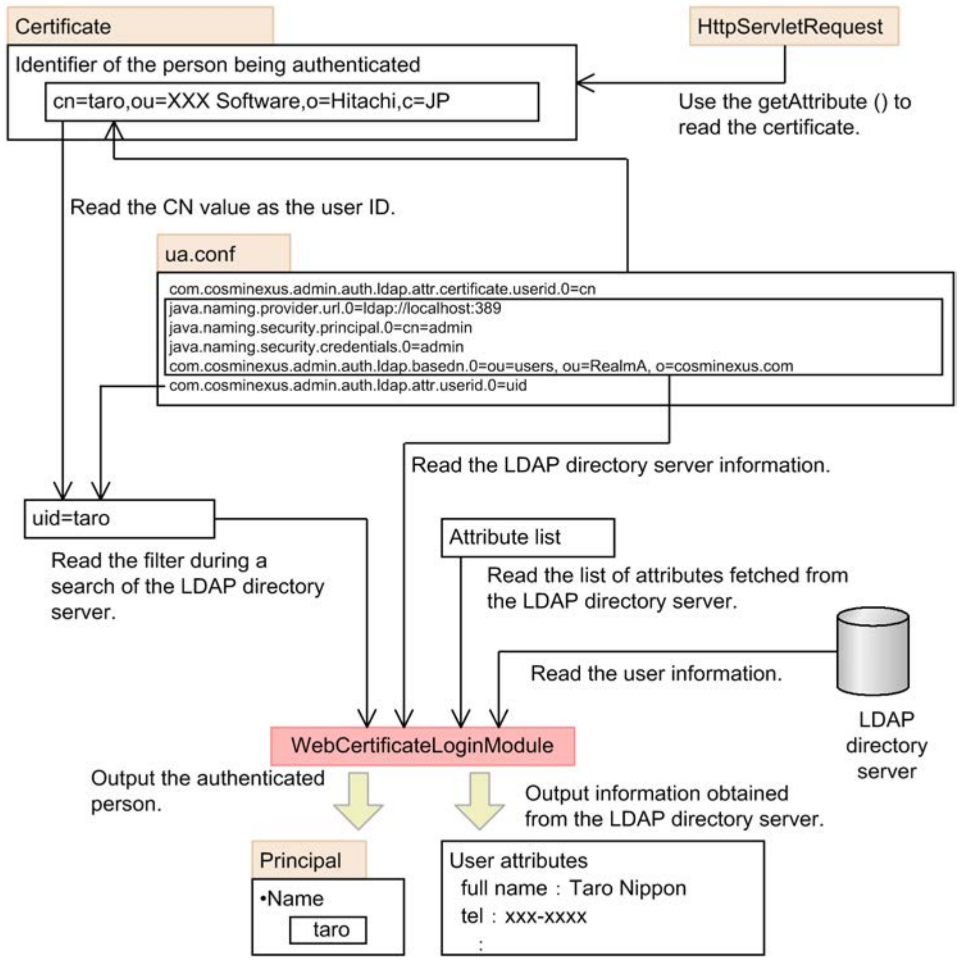
It maps the distinguished name of the user requesting authentication in the client certificate that the Web server requests from the browser during SSL authentication to the user information repository.

To use this module, pre-specify the attribute names that are the user IDs in the requesting users' distinguished names in the client certificates (cn) and the attribute names used to search the LDAP directory server (uid) in `ua.conf` (the integrated user management configuration file).

WebCertificateLoginModule reads this file and uses the client certificate to perform the authentication process. It then obtains the user ID from the client certificate and accesses the LDAP directory server. If authentication is successful, it returns the user attributes when they are found. If no user ID in the certificate is found, FailedLoginException is returned.

The following figure shows an overview of WebCertificateLoginModule.

Figure 5-13: Overview of WebCertificateLoginModule


Certificate
Identifier of the person being authenticated
cn=taro,ou=XXX Software,o=Hitachi,c=JP
HttpServletRequest
Use the getAttribute () to read the certificate.
Read the CN value as the user ID.
ua.conf
Read the LDAP directory server information.
uid=taro
Read the filter during a search of the LDAP directory server.
Attribute list
Read the list of attributes fetched from the LDAP directory server.
Read the user information.
LDAP directory server
WebCertificateLoginModule
Output the authenticated person.
Output information obtained from the LDAP directory server.
Principal
•Name
taro
User attributes
full name : Taro Nippon
tel : xxx-xxxx
:

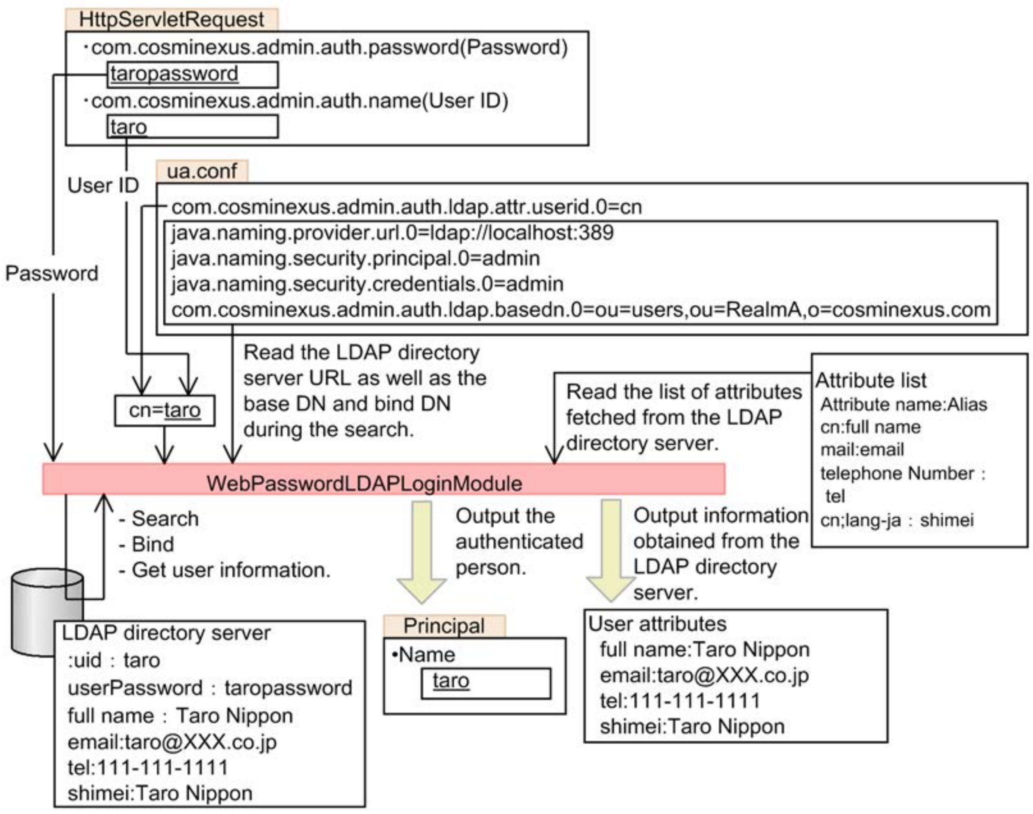
```
com.cosminexus.admin.auth.ldap.attr.certificate.userid.0=cn
java.naming.provider.url.0=ldap://localhost:389
java.naming.security.principal.0=cn=admin
java.naming.security.credentials.0=admin
com.cosminexus.admin.auth.ldap.basedn.0=ou=users, ou=RealmA, o=cosminexus.com
com.cosminexus.admin.auth.ldap.attr.userid.0=uid
```

5.3.4 WebPasswordLDAPLoginModule

WebPasswordLDAPLoginModule is the login module that uses the LDAP directory server's authentication function.

It tries to bind to the LDAP directory server by using the entered user ID and password. When the attempt succeeds, authentication is successful. The following figure shows an overview of WebPasswordLDAPLoginModule.

Figure 5-14: Overview of WebPasswordLDAPLoginModule


HttpServletRequest
・com.cosminexus.admin.auth.password(Password)
taropassword
・com.cosminexus.admin.auth.name(User ID)
taro
User ID
Password
ua.conf
com.cosminexus.admin.auth.ldap.attr.userid.0=cn
java.naming.provider.url.0=ldap://localhost:389
java.naming.security.principal.0=admin
java.naming.security.credentials.0=admin
com.cosminexus.admin.auth.ldap.basedn.0=ou=users,ou=RealmA,o=cosminexus.com
cn=taro
Read the LDAP directory server URL as well as the base DN and bind DN during the search.
Read the list of attributes fetched from the LDAP directory server.
Attribute list
Attribute name:Alias
cn:full name
mail:email
telephone Number : tel
cn;lang-ja : shimei
WebPasswordLDAPLoginModule
- Search
- Bind
- Get user information.
Output the authenticated person.
Output information obtained from the LDAP directory server.
LDAP directory server
:uid : taro
userPassword : taropassword
full name : Taro Nippon
email:taro@XXX.co.jp
tel:111-111-1111
shimei:Taro Nippon
Principal
•Name
taro
User attributes
full name:Taro Nippon
email:taro@XXX.co.jp
tel:111-111-1111
shimei:Taro Nippon

To use this module, specify the definition to connect to the LDAP directory server and the attribute names used to retrieve entries in `ua.conf` (the integrated user management configuration file).

WebPasswordLDAPLoginModule reads this file and obtains the user ID from HttpServletRequest to find the user entry DN. It then tries to bind to the LDAP directory server by using this DN and the password obtained from HttpServletRequest. When the attempt succeeds, it returns the user attributes.

User entry search and the user ID and password used to obtain user attributes

When searching for the user entry to authenticate the user, the module uses the bind DN and password specified in `ua.conf` (the integrated user management configuration file). To obtain user attributes, it uses the user entry DN and password as the bind DN and password. To learn more about user entry search, see *5.3.8 (1) User entry search*.

Notes on using the LDAP connection pool

The LDAP connection pool is used for the user entry search process only. It is not used to authenticate users or obtain user attributes. When not searching for user entries, disable the LDAP connection pool. To learn more about the LDAP connection pool, see *5.3.8 (2) Connection pool*.

5.3.5 WebPasswordJDBCLoginModule

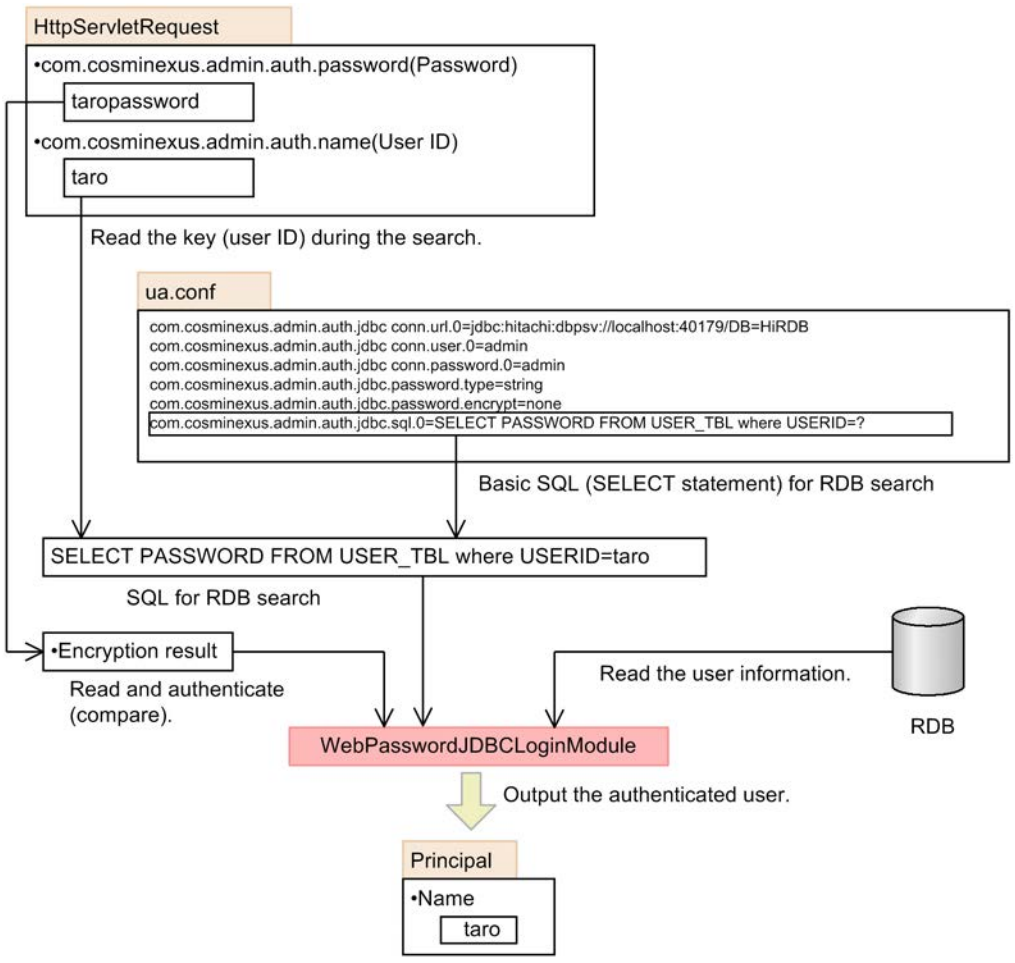
WebPasswordJDBCLoginModule is the login module used when the database is already used for user management.

It uses the entered user ID and password to retrieve the password from the user information stored in the database.

To use this module, specify the definition to connect to the database and the SQL used to retrieve entries (SELECT statement) in `ua.conf` (the integrated user management configuration file).

WebPasswordJDBCLoginModule reads this file, obtains the user ID from HttpServletRequest, uses JDBC to access to the database and search for the password, and then uses the password to perform password authentication. The following figure shows an overview of WebPasswordJDBCLoginModule.

Figure 5-15: Overview of WebPasswordJDBCLoginModule



```
com.cosminexus.admin.auth.jdbc conn.url.0=jdbc:hitachi:dbpsv://localhost:40179/DB=HiRDB
com.cosminexus.admin.auth.jdbc conn.user.0=admin
com.cosminexus.admin.auth.jdbc conn.password.0=admin
com.cosminexus.admin.auth.jdbc.password.type=string
com.cosminexus.admin.auth.jdbc.password.encrypt=none
com.cosminexus.admin.auth.jdbc.sql.0=SELECT PASSWORD FROM USER_TBL where USERID=?
```

```
SELECT PASSWORD FROM USER_TBL where USERID=taro
```

HttpServletRequest
•com.cosminexus.admin.auth.password(Password)
taropassword
•com.cosminexus.admin.auth.name(User ID)
taro
Read the key (user ID) during the search.
ua.conf
Basic SQL (SELECT statement) for RDB search
SQL for RDB search
•Encryption result
Read and authenticate (compare).
Read the user information.
RDB
WebPasswordJDBCLoginModule
Output the authenticated user.
Principal
•Name
taro

In addition, WebPasswordJDBCLoginModule references the JDBC driver classes in the login module. The available JDBC drivers and the procedures to set up the JDBC driver are as follows.

■ Available JDBC drivers

The following table lists the databases and JDBC drivers used by WebPasswordJDBCLoginModule.

Table 5-7: Databases and JDBC drivers used by WebPasswordJDBCLoginModule

Database		JDBC driver
HiRDB#		HiRDB Type4 JDBC Driver
Oracle	Oracle 11g	Oracle JDBC Thin Driver
SQL Server		SQL Server JDBC Driver

#: Includes HiRDB Run Time

■ Procedures to set up the JDBC driver

Set the JDBC driver class in `ua.conf` (the integrated user management configuration file). Store the JDBC driver in any directory, and then add that directory to the J2EE server class path. The setup procedures are as follows:

1. Enter the following items in `ua.conf` (the integrated user management configuration file).
   - The JDBC driver class name that corresponds to the used JDBC driver
   - The database and the URL to connect to that database
   - Delegate database users and their passwords

   The setup examples for databases are as follows: Replace the bold letters with the appropriate ones depending on the database environment.

   HiRDB:

   ```
   com.cosminexus.admin.auth.jdbc.driver.0=JP.co.Hitachi.s
   oft.HiRDB.JDBC.HiRDBDriver

   com.cosminexus.admin.auth.jdbc.conn.url.0=jdbc:hitachi:
   hirdb://DBID=22200,DBHOST=hostA

   com.cosminexus.admin.auth.jdbc.conn.user.0=system

   com.cosminexus.admin.auth.jdbc.conn.password.0=userpass
   ```

   Oracle:

   ```
   com.cosminexus.admin.auth.jdbc.driver.0=oracle.jdbc.Ora
   cleDriver

   com.cosminexus.admin.auth.jdbc.conn.url.0=jdbc:oracle:t
   hin:@localhost:1521:orcl

   com.cosminexus.admin.auth.jdbc.conn.user.0=system

   com.cosminexus.admin.auth.jdbc.conn.password.0=userpass
   ```

SQL Server:

```
com.cosminexus.admin.auth.jdbc.driver.0=com.microsoft.s
qlserver.jdbc.SQLServerDriver
```

```
com.cosminexus.admin.auth.jdbc.conn.url.0=jdbc:sqlserve
r://localhost:1433;DatabaseName=sqlserver
```

```
com.cosminexus.admin.auth.jdbc.conn.user.0=system
```

```
com.cosminexus.admin.auth.jdbc.conn.password.0=userpass
```

2. Store the JDBC driver JAR file in any directory of the machine running the J2EE server.
3. Enter the path of the JAR file stored in Step 2 in `usrconf.cfg` (the option definition file) of the J2EE server.

   The setup example is as follows:

   `add.class.path=`*directory stored in Step 2*`/`*JAR file name*

   Note that the JAR file name depends on the database to be connected.

■ **Notes**

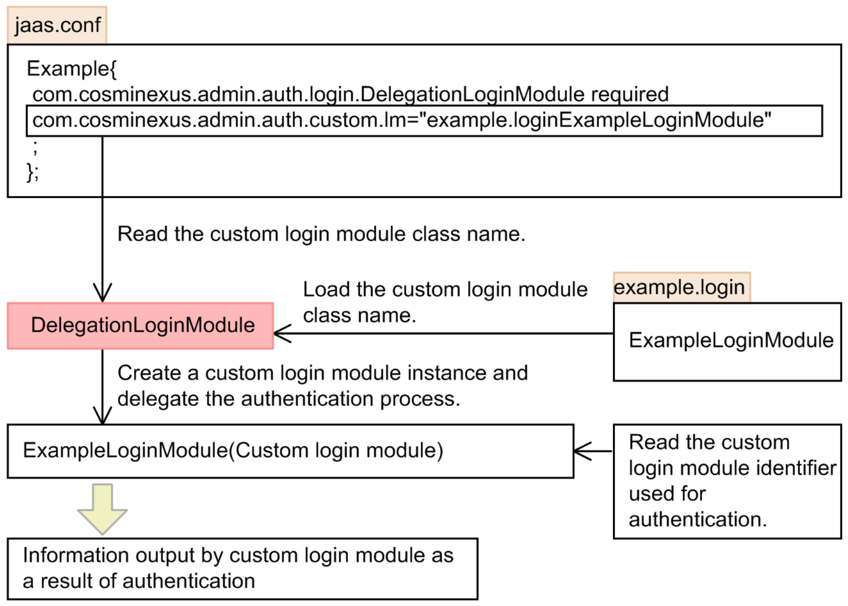
- When using the Oracle database, specify a variable-length character string such as VARCHAR2 in the column (USERID, etc.) of the user information table created in the database. If a fixed-length character string such as CHAR is specified, password authentication may fail.
- The Windows authentication is not supported as the SQL Server authentication mode.

5.3.6 DelegationLoginModule

This login module is used to invoke custom login modules.

It delegates the authentication process to a specified custom login module. The following figure shows an overview of DelegationLoginModule.

Figure 5-16: Overview of DelegationLoginModule


jaas.conf
Example{
com.cosminexus.admin.auth.login.DelegationLoginModule required
com.cosminexus.admin.auth.custom.lm="example.loginExampleLoginModule"
;
};
Read the custom login module class name.
DelegationLoginModule
Load the custom login module class name.
example.login
ExampleLoginModule
Create a custom login module instance and delegate the authentication process.
ExampleLoginModule(Custom login module)
Read the custom login module identifier used for authentication.
Information output by custom login module as a result of authentication

To use this module, specify the class name of the used custom login module in `jaas.conf` (the JAAS configuration file). DelegationLoginModule reads this file and instantiates the custom login module. The argument given to the `initialize` method of DelegationLoginModule is passed to the custom login module.

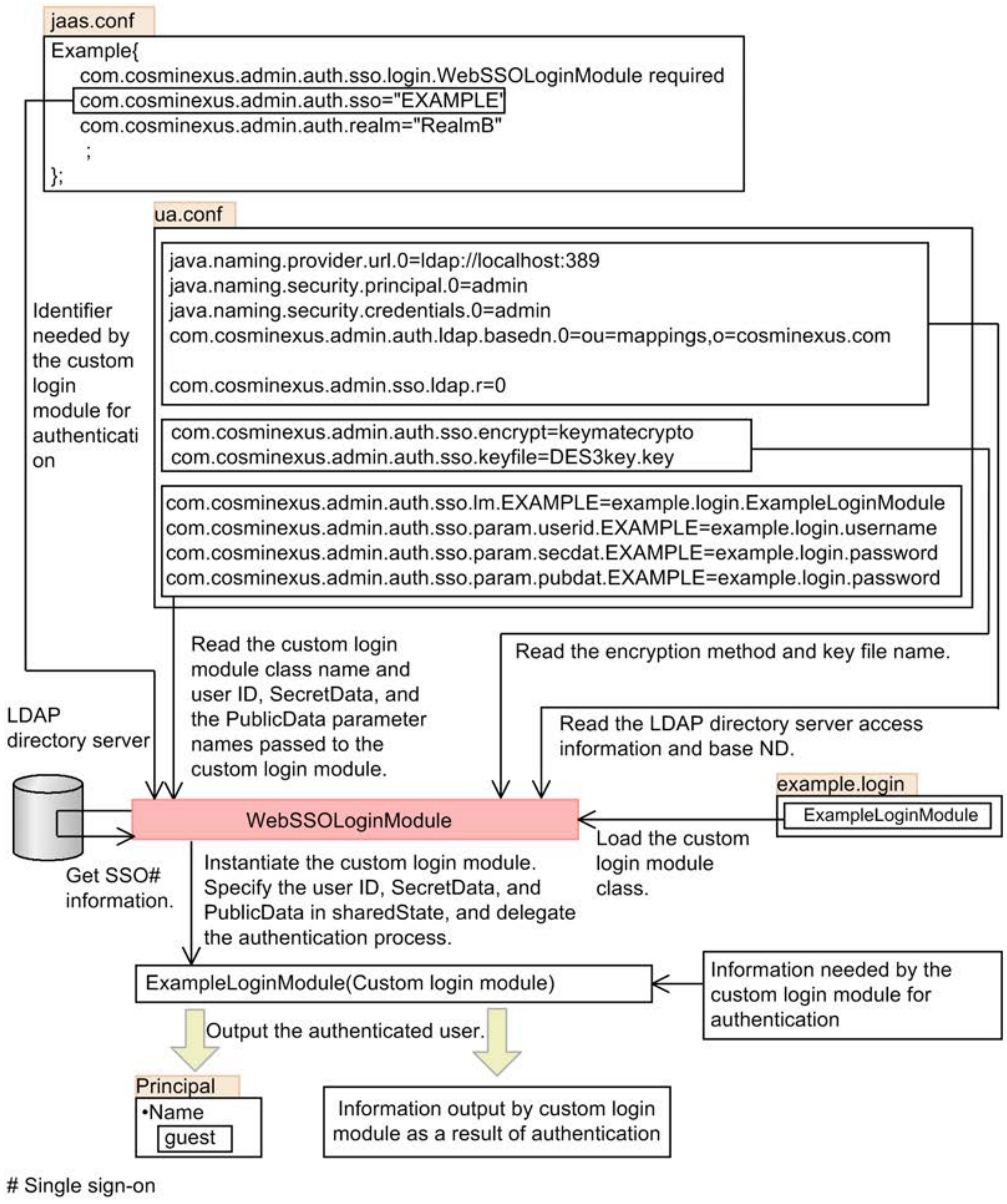
The authentication process is delegated to the custom login module.

5.3.7 WebSSOLoginModule

This is the login module used to implement single sign-on. It invokes Cosminexus standard login modules or custom login modules.

When a user has logged in one session, the information used for authentication in other realms (user ID, SecretData, and PublicData) is given to the custom login module. The following figure shows an overview of WebSSOLoginModule.

Figure 5-17: Overview of WebSSOLoginModule


jaas.conf
Example{
com.cosminexus.admin.auth.sso.login.WebSSOLoginModule required
com.cosminexus.admin.auth.sso="EXAMPLE"
com.cosminexus.admin.auth.realm="RealmB"
;
};
ua.conf
java.naming.provider.url.0=ldap://localhost:389
java.naming.security.principal.0=admin
java.naming.security.credentials.0=admin
com.cosminexus.admin.auth.ldap.basedn.0=ou=mappings,o=cosminexus.com
com.cosminexus.admin.sso.ldap.r=0
com.cosminexus.admin.auth.sso.encrypt=keymatecrypto
com.cosminexus.admin.auth.sso.keyfile=DES3key.key
com.cosminexus.admin.auth.sso.lm.EXAMPLE=example.login.ExampleLoginModule
com.cosminexus.admin.auth.sso.param.userid.EXAMPLE=example.login.username
com.cosminexus.admin.auth.sso.param.secdat.EXAMPLE=example.login.password
com.cosminexus.admin.auth.sso.param.pubdat.EXAMPLE=example.login.password
Identifier needed by the custom login module for authentication
Read the custom login module class name and user ID, SecretData, and the PublicData parameter names passed to the custom login module.
Read the encryption method and key file name.
Read the LDAP directory server access information and base ND.
LDAP directory server
WebSSOLoginModule
example.login
ExampleLoginModule
Load the custom login module class.
Get SSO# information.
Instantiate the custom login module. Specify the user ID, SecretData, and PublicData in sharedState, and delegate the authentication process.
ExampleLoginModule(Custom login module)
Information needed by the custom login module for authentication
Output the authenticated user.
Principal
•Name
guest
Information output by custom login module as a result of authentication
# Single sign-on

WebSSOLoginModule reads `ua.conf` (the integrated user management configuration file) to obtain the custom login module class name that corresponds to the login module identifier specified in `jaas.conf` (the JAAS configuration file), and it then instantiates the custom login module. The argument given to the `initialize` method of WebSSOLoginModule is passed to the custom login module.

When a user has logged into the session, the custom login module obtains the single sign-on information of the user who logged in from the LDAP directory server specified in `ua.conf`. If the single sign-on information contains the user mapping information of the destination realm, it obtains the single sign-on information of the destination user. SecretData in the single sign-on information is decrypted by the method specified in `ua.conf`. WebSSOLoginModule then enters the destination user ID, decrypted SecretData, and PublicData in sharedState (the Map object that is passed by the `initialize` method to the custom login module). The parameter name used for setup is specified in `ua.conf`.

When no user has logged into the session, WebSSOLoginModule does not change sharedState.

The authentication process is delegated to the custom login module.

5.3.8 Repository access by Cosminexus standard login modules

This section describes how the Cosminexus standard login modules access the user information repository.

(1) User entry search

The following login modules use the LDAP directory server as the user information repository and can search for user entries during authentication.

- WebPasswordLoginModule
- WebCertificateLoginModule
- WebPasswordLDAPLoginModule

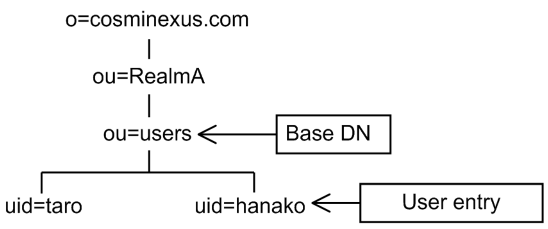
`ua.conf` (the integrated user management configuration file) is used to specify whether to search for user entries and the search scope. The necessity of the search depends on the DIT structure of the LDAP directory server.

Cases in which a search is not needed

To obtain the user attributes and authenticate the user, it is necessary to locate the user entry on the LDAP directory server based on the user ID entered by the user.

If the user entry is immediately below the base DN and if the user ID is included in the user entry RDN (Relative Distinguished Name) as shown in the following figure, the user entry DN can be composed of the base DN, the attribute name representing the user ID and the user ID. Thus, a search is not needed. When implementing integrated user management, it is recommended to construct a DIT structure that does not require search.

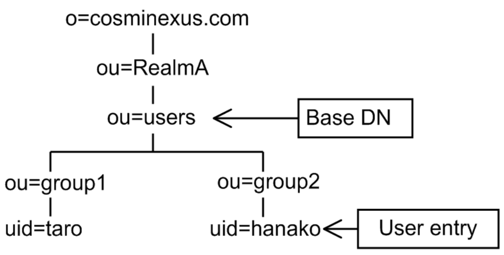
Figure 5-18: User entry immediately below the base DN



Cases in which a search is needed

If the user ID is not included in the user entry RDN or if the user entry is not immediately below the base DN, it is necessary to search for the user entry. When the user entry is at two or more levels below the base DN as shown in the figure, the search scope must include all the subtrees (all the levels below the base DN).

Figure 5-19: User entry at two or more levels below the base DN



(2) Connection pool

The Cosminexus standard login modules can use the connection pool to accelerate access to the user information repository.

The connection pool is specified in `ua.conf` (the integrated user management configuration file).

The following login modules can use the LDAP connection pool.

- WebPasswordLoginModule
- WebCertificateLoginModule
- WebPasswordLDAPLoginModule
- WebSSOLoginModule

The following login module can use the JDBC connection pool.

- WebPasswordJDBCLoginModule

5.3.9 Enhanced support of authentication password encryption

WebPasswordLoginModule and WebPasswordJDBCLoginModule allow password authentication even if the passwords stored in the repository are not encrypted in SHA-1 or MD5 or in plain text.

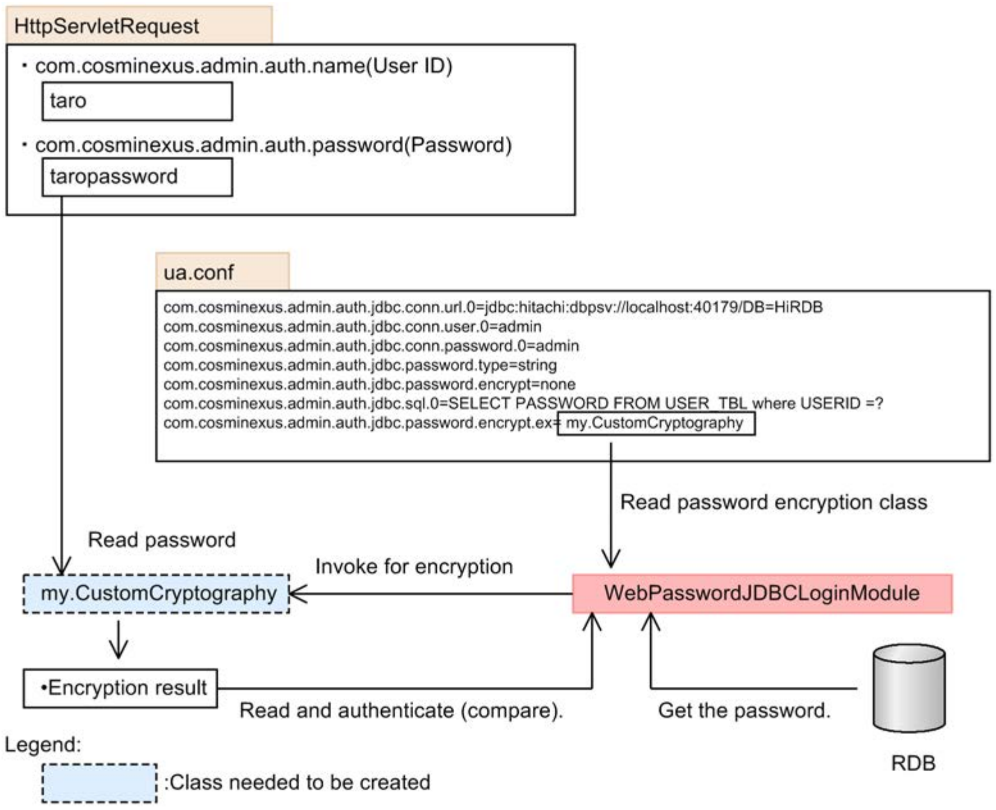
Thanks to the enhanced support of authentication password encryption, it is possible to perform password authentication even if the passwords stored in the repository are encrypted in any non-default format. To use the enhanced support, the application developer must create the implementation class in advance.

The login module converts the entered password in HttpServletRequest to compare it to the password obtained from the database.

When com.cosminexus.admin.auth.jdbc.password.encrypt.ex is set in `ua.conf` (the integrated user management configuration file), the module instantiates the class implementation to convert the entered password in HttpServletRequest.

When the byte characters of the converted password completely match the password in the database, authentication is successful. The following figure shows an overview of the enhanced support of authentication password encryption.

Figure 5-20: Overview of the enhanced support of authentication password encryption


HttpServletRequest
• com.cosminexus.admin.auth.name(User ID)
taro
• com.cosminexus.admin.auth.password(Password)
taropassword
ua.conf
Read password encryption class
Read password
Invoke for encryption
my.CustomCryptography
WebPasswordJDBCLoginModule
•Encryption result
Read and authenticate (compare).
Get the password.
RDB
Legend:
:Class needed to be created

```
com.cosminexus.admin.auth.jdbc.conn.url.0=jdbc:hitachi:dbpsv://localhost:40179/DB=HiRDB
com.cosminexus.admin.auth.jdbc.conn.user.0=admin
com.cosminexus.admin.auth.jdbc.conn.password.0=admin
com.cosminexus.admin.auth.jdbc.password.type=string
com.cosminexus.admin.auth.jdbc.password.encrypt=none
com.cosminexus.admin.auth.jdbc.sql.0=SELECT PASSWORD FROM USER_TBL where USERID =?
com.cosminexus.admin.auth.jdbc.password.encrypt.ex= my.CustomCryptography
```

To learn more about creating the implementation class, see *5.10 Implementation of API-based user authentication*.

5.3.10 Configuration file parameters used by login modules

The parameters that must be set in `ua.conf` (the integrated user management configuration file) depend on the used Cosminexus standard login modules.

(1) Login modules that use the LDAP directory server

The following table lists the parameters used by the login modules that use the LDAP directory server. To learn the meanings of the parameters, see *14.3 ua.conf (integrated user management configuration file)*.

Table 5-8: List of parameters used by login modules that use the LDAP directory server

Parameter	Module			
	P	C	S	L
java.naming.provider.url	A	A	A	A
java.naming.security.principal	A	A	A	A[#1]
java.naming.security.credentials	A	A	A	A[#1]
com.cosminexus.admin.auth.ldap.basedn	A	A	A	A
com.cosminexus.admin.auth.ldap.attr.userid	A	A	X	A
com.cosminexus.admin.auth.ldap.search.userrdn	A	A	X	A[#2]
com.cosminexus.admin.auth.ldap.search.scope	A	A	X	A[#3]
com.cosminexus.admin.auth.ldap.attr.password	A	X	X	A[#4]
com.cosminexus.admin.auth.ldap.pool.enable	A	A	A	A[#5]
com.cosminexus.admin.auth.ldap.pool.max	A	A	A	A
com.cosminexus.admin.auth.ldap.pool.max_spare	A	A	A	A
com.cosminexus.admin.auth.ldap.pool.min_spare	A	A	A	A
com.cosminexus.admin.auth.ldap.pool.gc_interval	A	A	A	A
com.cosminexus.admin.auth.ldap.conn.retry.count	A	A	A	A
com.cosminexus.admin.auth.ldap.conn.retry.wait	A	A	A	A
com.cosminexus.admin.auth.ldap.certificate.attr.userid	X	A	X	X
com.cosminexus.admin.auth.ldap.password.encrypt	A	X	X	X
com.cosminexus.admin.auth.ldap.password.encrypt.ex	A	X	X	X
com.cosminexus.admin.auth.ldap.directory.kind	X	X	X	A

Legend:

P: WebPasswordLoginModule

C: WebCertificateLoginModule

S: WebSSOLoginModule

L: WebPasswordLDAPLoginModule

A: Available; X: Not available

#1: This parameter is necessary only when user entries are searched for. User entry search uses the bind DN and password.

#2: Set this parameter to `true` when user entries are searched for (they are not immediately below the base DN).

#3: Specify the subtrees that must be included in the search scope when user entries are searched for.

#4: This parameter is necessary only when user passwords are changed. Specify unicodePwd if Active Directory is used as the LDAP directory server. Otherwise, specify userPassword.

#5: The LDAP connection pool is used only when user entries are searched for. Otherwise, set this parameter to `false`. Whether or not user entries are searched for, the LDAP connection pool is not used when the LDAP directory server is accessed for user authentication.

(2) Login modules that use a database

The following table lists the parameters used by the login module that uses the database. To learn the meanings of the parameters, see *14.3 ua.conf (integrated user management configuration file)*.

Table 5-9: List of parameters used by the login module that uses database

Parameter	Module
	J
com.cosminexus.admin.auth.jdbc.driver	A
com.cosminexus.admin.auth.jdbc.conn.url	A
com.cosminexus.admin.auth.jdbc.conn.user	A
com.cosminexus.admin.auth.jdbc.conn.password	A
com.cosminexus.admin.auth.jdbc.pool.enable	A
com.cosminexus.admin.auth.jdbc.pool.max	A
com.cosminexus.admin.auth.jdbc.pool.max_spare	A
com.cosminexus.admin.auth.jdbc.pool.min_spare	A
com.cosminexus.admin.auth.jdbc.pool.gc_interval	A
com.cosminexus.admin.auth.jdbc.conn.retry.count	A

Parameter	Module
	J
com.cosminexus.admin.auth.jdbc.conn.retry.wait	A
com.cosminexus.admin.auth.jdbc.sql	A
com.cosminexus.admin.auth.jdbc.password.type	A
com.cosminexus.admin.auth.jdbc.password.encrypt	A
com.cosminexus.admin.auth.jdbc.password.encrypt.ex	A

Legend:

J: WebPasswordJDBCLoginModule

A: Available

5.4 Sessions managed in integrated user management

This section explains the sessions managed in integrated user management.

5.4.1 Types of sessions

There are the following types of sessions that relate to the integrated user management function.

- Web container-managed session (`HttpSession`)

  Represents the `HttpSession` object.

- Integrated user management session

  This represents the period from login to logout using the Cosminexus standard login module.

  As multiple users can log into one session, the session will not become invalid unless all login users log out. As it synchronizes with HttpSession, which is the Web container-managed session, it will become invalid when HttpSession becomes invalid even if users are logging in.

  The following two functions are used to control the integrated user management session.

  - The Cosminexus standard login modules can automatically register or delete the login user IDs and realm names during or from the session.
  - Integrated user management supports the session failover functionality. When the session failover functionality is enabled, the session failover cluster can inherit the login state. The administrators can select whether to inherit the login state when the session failover functionality is enabled.

- Custom login module session

  This represents the login user session that each application has. The concept of this session depends on the custom login module specifications. (Generally, it represents the period from login to logout.)

  The custom login modules can automatically register or delete the login user IDs and realm names to or from the session.

5.4.2 Registration of login user IDs

This section explains the registering of the user IDs of users who log into the integrated user management session.

(1) Purpose of registering login user IDs

The purpose of registering the user IDs of users who log into the integrated user management session is as follows:

- It enables the login modules to determine if users log into the integrated user management session by checking the `<ua:notLogin>` tag in the JSP tag library. In addition, specifying a realm name enables them to determine if users log into that realm.

(2) Conditions in which to register login user IDs

The Cosminexus standard login modules that have the authentication mechanism automatically register the user IDs of users who log into the integrated user management session.

The custom login modules register the user IDs of users in the integrated user management session if they satisfy the following both conditions:

- If, in the custom login module implementation, the Principal object is associated with the Subject by the `commit` method.
- If WebSSOHandler provided by the integrated user management framework is set to the LoginContext class constructor argument.

When multiple login modules are invoked in one call, the user IDs are not registered until a user logs in by using the Cosminexus standard login module that has the authentication mechanism or the login module that satisfies the above conditions. When no users use such modules for log in, the user IDs are not registered in the integrated user management session.

(3) Contents registered in the integrated user management session

Registered in the integrated user management session are realm names, user IDs, and login times.

- ■ Realm name

  This is set to the value specified by `com.cosminexus.admin.auth.realm` in `jaas.conf` (the JAAS configuration file). A null character is assigned when this option is omitted. `com.cosminexus.admin.auth.realm` must be always specified unless DelegationLoginModule is used.

- ■ User ID

  This is set to the user ID in the Principal object initially requested by the Subject after each login module's `commit` method is invoked (the result returned by the `getName` method).

- ■ Login time

  This is set to the time when a user logs in and the user ID is registered in the integrated user management session. The login time is registered per user.

5.4.3 Deletion of user IDs registered in the integrated user management session

When the `logout` method of the LoginContext class that has the authenticated Subject is invoked, the user IDs and realm names are automatically deleted from the integrated user management session.

5.4.4 Examples of JAAS configuration file definition

The following are examples of `jaas.conf` (the JAAS configuration file) definition.

(1) Example of definition using Cosminexus standard login modules

When the following definition is made in the JAAS configuration file, the user ID used when the first-executed WebPasswordLoginModule authenticates the user is registered together with the realm name `RealmA` in the integrated user management session.

```
Example03 {
  com.cosminexus.admin.auth.login.WebPasswordLoginModule required

                         // This is to join the session
    com.cosminexus.admin.auth.realm="RealmA"
    com.cosminexus.admin.auth.ldap.r="0"
    com.cosminexus.admin.auth.ldap.w="1"
    ;
  com.cosminexus.admin.auth.login.DelegationLoginModule required
    com.cosminexus.admin.auth.custom.lm="my.login.MyLoginModule"
    my.login.useracctterm="acctTerm"
    ;
};
```

(2) Example of definition using custom login modules only

When the following definition is made in the JAAS configuration file, the user ID that is in the Principal object set by the first executed MyLoginModule1 `commit` method (the getName method value) is registered in the integrated user management session. (DelegationLoginModule registers the user ID.)

As `com.cosminexus.admin.auth.realm` is not specified, a null character (`" "`) is assigned as the realm name.

```
Example99 {
  com.cosminexus.admin.auth.login.DelegationLoginModule required

                        // This is to join the session
    com.cosminexus.admin.auth.custom.lm="my.login.MyLoginModule1"
    ;
  com.cosminexus.admin.auth.login.DelegationLoginModule required
    com.cosminexus.admin.auth.custom.lm="my.login.MyLoginModule2"
    ;
};
```

If MyLoginModule1 does not set the Principal object in the Subject, the user ID in the object set by MyLoginModule2 is registered in the integrated user management session. If MyLoginModule2 also does not set the Principal object, no user ID is registered in the integrated user management session.

For details about `jaas.conf`, see *14.2 jaas.conf (JAAS configuration file)*.

5.4.5 Inheritance of the login state using session failover functionality

Integrated user management uses the session failover functionality to pass the login state to another J2EE server.

The following table lists the information that is inherited by the session failover. In the table, *API* refers to the JAAS API. *Custom Tag* is the JSP custom tag provided in integrated user management.

Table 5-10: Information that is inherited by the session failover functionality

Information		Cosminexus standard login module		Custom login module	
		API	Custom tag	API	Custom tag
Integrated user management session information	Login state (whether users are logging in or not)	Auto	Auto	Auto	Auto
	Registered user ID/realm name	Auto	Auto	Auto	Auto
	Password	O	O	n/a	n/a

Information		Cosminexus standard login module		Custom login module	
		API	Custom tag	API	Custom tag
Authentication information	Subject	X	X	X	X
	Principal	X	X	X	X
	User attribute (UserAttributes)	Man	Man	n/a	n/a
	Credential (public/private) other than user attributes (UserAttributes)	X	X	X	X

Legend:

Auto: Can be inherited automatically

O: Can be selected depending on the option

Man: Can be inherited manually

X: Cannot be inherited

n/a: Not applicable

To fail over the user application created in versions earlier than 07-00 by using the session failover function, part of the user application process must be revised. For details about the method used to develop the integrated user management application that supports the session failover, see *5.12 Implementation of user authentication when using the session failover functionality*. To learn more about the session failover functionality, see *5. Inheritance of the Session Information Between J2EE Servers* in the *uCosminexus Application Server Expansion Guide*.

5.5 Use of single sign-on

This section describes the use of single sign-on. Single sign-on is the function that allows users to seamlessly use multiple systems having different user IDs once they log in.

5.5.1 Necessary procedures for single sign-on

To use single sign-on, WebSSOLoginModule is required to invoke all the custom login modules that use single sign-on and the Cosminexus standard login modules (login modules that authenticate users).

`jaas.conf` (the JAAS configuration file) is used to enable single sign-on.

In the following example, WebPasswordLoginModule is used to authentication users.

```
AP1 {
  com.cosminexus.admin.auth.login.WebPasswordLoginModule Requisite
    com.cosminexus.admin.auth.ldap.r="3"
    com.cosminexus.admin.auth.ldap.w="2"
    com.cosminexus.admin.auth.realm=XXXcompany;
};
```

To apply single sign-on, modify the statement as shown in bold below.

```
AP1 {
  com.cosminexus.admin.auth.sso.login.WebSSOLoginModule Requisite
    com.cosminexus.admin.auth.ldap.r="3"
    com.cosminexus.admin.auth.ldap.w="2"
    com.cosminexus.admin.auth.realm=XXXcompany;
};
```

The definition of all login modules for single sign-on is active after the above modification is made and after the J2EE server is started.

5.5.2 Application of single sign-on to existing application user management

The conditions listed in the following table must be met to apply integrated user management single sign-on when the existing application has already performed user management.

Table 5-11: Necessary conditions to apply integrated user management single sign-on

Presence of LoginModule	Condition 1	Condition 2	Applicability	Method
Yes	Can be modified.	n/a	O	Use sharedState to pass authentication information.
	Cannot be modified.	Authentication information can be passed by using sharedState.	O	Modify `jaas.conf` (the JAAS configuration file) for the single sign-on library.
		Authentication information cannot be passed by using sharedState.	X	n/a
No	The login API is available.	n/a	O	Create a login module.
	The login API is not available.	n/a	X	n/a

Legend:

O: Can be applied

X: Cannot be applied

n/a: Not applicable

Among the Cosminexus standard login modules, WebPasswordLoginModule, WebCertificateLoginModule, WebPasswordLDAPLoginModule, and WebPasswordJDBCLoginModule (login modules that authenticate users) support single sign-on.

5.6 Use of custom login modules

To perform application user authentication by using modules other than the Cosminexus standard login modules, create custom login modules and use them with the Cosminexus standard login modules.

The custom login modules should be implemented by application developers.

5.6.1 Overview of custom login modules

Custom login modules are a class that is created to perform application user authentication by using modules other than the Cosminexus standard login modules. This class is created by inheriting the LoginModule interface that is the JAAS SPI.

Custom login modules should be stored in the following directories.

- Windows:

  Cosminexus installation directory`\manager\modules`

- UNIX:

  `/opt/Cosminexus/manager/modules`

The custom login module directories can be changed by `com.cosminexus.admin.auth.custom.modules` in `ua.conf` (the integrated user management configuration file).

The following are notes to be aware of when storing custom login modules.

- The class in the directory that stores custom login modules is invoked by the custom login module class loader. Thus, it cannot be directly used by the application. To enable the application to directly use this class, specify the directory that stores custom login directories in the `add.class.path` key in `usrconf.cfg` (J2EE server option definition file).
- Always store custom login modules in the form of `.class` file. Do not store all custom login modules in a JAR file. If a class hierarchy is present, the custom login module directory must have the same class hierarchy structure.

  An example is as follows.

  Example: The custom login module class is `my.login.MyLoginModule`.

```
Cosminexus-installation-directory
    └manager
     └modules
      └my
       └login
        └MyLoginModule.class
```

- To use custom login modules in the integrated user management framework, you must set the custom login modules and the classes associated with the custom login modules in the execution environment beforehand. The JAR file format is not supported. Set the `bytecode` class files.

5.6.2 Invocation of custom login modules

Custom login modules can be invoked by either of the following Cosminexus standard login modules: DelegationLoginModule or WebSSOLoginModule.

- ■ `DelegationLoginModule`

  This is used when single sign-on is not used.

- ■ `WebSSOLoginModule`

  This is used when single sign-on is used.

By invoking the custom login module from the Cosminexus standard login module, the user can join the login session managed in integrated user management. This session is different from the Web container-managed session. To learn more about the integrated user management sessions, see *5.4 Sessions managed in integrated user management*.

Custom login modules are invoked by either DelegationLoginModule or WebSSOLoginModule. When the user is successfully authenticated by the custom login module invoked by either of these login modules, he or she automatically joins the integrated user management session. The user ID needed to register the integrated user management session is obtained from the Principal object associated to Subject.

To use single sign-on, WebSSOLoginModule is used when the custom login module is invoked for the second time, the user ID that is obtained during the first join to the integrated user management session is used to obtain the necessary authentication information from the single sign-on information repository, and the information is passed to the custom login module. When the information stored in the repository is encrypted, it is decrypted before being passed to the custom login module.

To learn more about implementing custom login modules, see *5.13 Implementation of custom login module-based user authentication*.

5.7 Management of user information

Integrated user management uses the LDAP directory server or database as the repository that manages user information. This section describes how user information is managed in the LDAP directory server.

5.7.1 Registration of user information to the LDAP directory server

When the LDAP directory server is used, user information can be registered in the following methods.

- Use the command provided by the LDAP directory server.

  Use the command provided by the LDAP directory server to register user information. Depending on the command, bulk user registration can be done based on the definition in the ldif file.

- Use the application developed by using the integrated user management API.

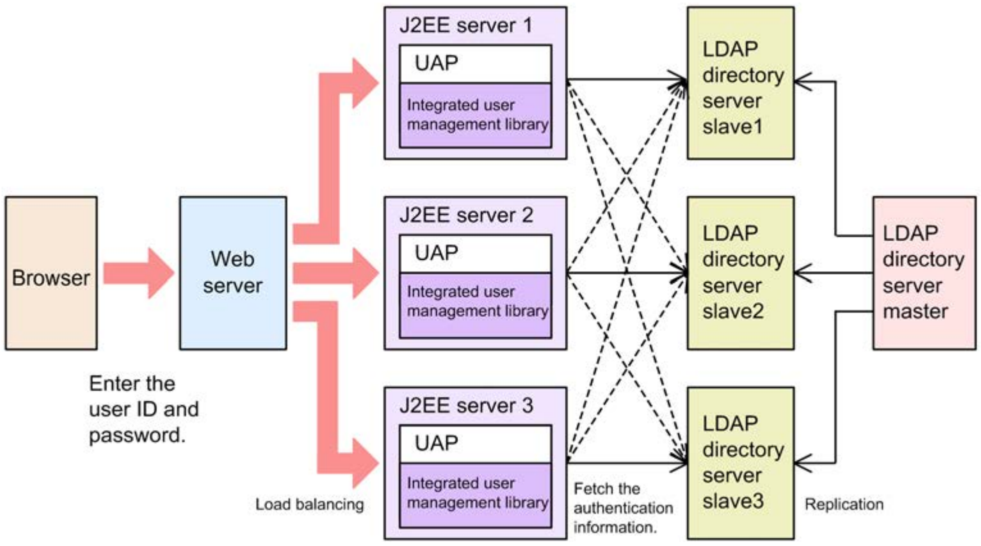
  Develop the application that uses user authentication and single sign-on libraries to perform user authentication. For details about implementing user authentication based on the integrated user management framework, see *5.9 Implementation of user authentication based on the integrated user management framework*.

5.7.2 Connection failover by multiplexing the LDAP directory server

Replicating and multiplexing the LDAP directory server enables the standard login modules provided in integrated user management to automatically switch from the failed LDAP directory server to the different pre-specified LDAP directory server, in order to reference the user and single sign-on information.

In the following example configuration, J2EE server 1 usually uses the LDAP directory server, slave1, to perform the authentication process. When slave1 goes down, J2EE server 1 automatically switches to slave2. (It switches to slave3 if slave2 goes down.)

Figure 5-21: Example configuration of LDAP directory server multiplexing


Browser
Web server
Enter the user ID and password.
Load balancing
J2EE server 1
UAP
Integrated user management library
J2EE server 2
UAP
Integrated user management library
J2EE server 3
UAP
Integrated user management library
Fetch the authentication information.
LDAP directory server slave1
LDAP directory server slave2
LDAP directory server slave3
LDAP directory server master
Replication

The J2EE server tries to access to the LDAP directory servers in the specified order. If all access attempts fail, authentication fails.

The procedures used to determine if the LDAP directory server goes down are as follows:

1. The `javax.naming.CommunicationException` exception occurs.

   This may be because the destination host denied access. For details, see the JDK documentation.

2. Execute the retry.

   The retry is repeated a preset number of times.

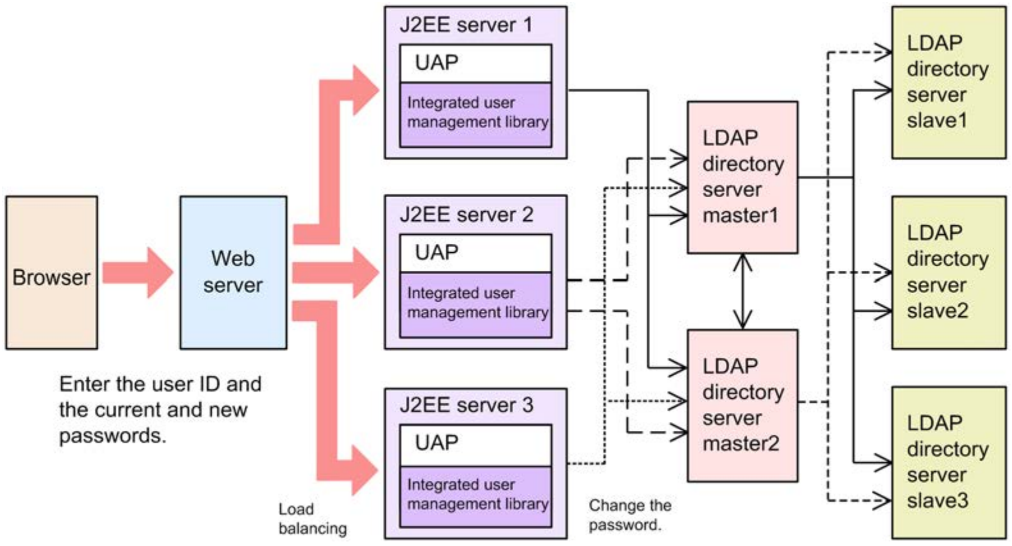
3. When the retry fails, the LDAP directory server is deemed to go down.

If all the LDAP directory servers go down, authentication fails and the LoginException exception occurs in the caller of the `login` method of the LoginContext class.

Store the LDAP directory server access settings in `ua.conf` (the integrated user management configuration file) and specify at least one LDAP setting for each JAAS application in `jaas.conf` (the JAAS configuration file). For details about `ua.conf` and the configuration file contents, see *14.3 ua.conf (integrated user management configuration file)*.

The connection failover also supports password change by using the PasswordUtil class. Password changes can be made to the master servers in a multi-master configuration as shown in the following figure.

Figure 5-22: Example configuration of LDAP directory server multiplexing (multi-master configuration)


Browser
Web server
Enter the user ID and the current and new passwords.
Load balancing
J2EE server 1
UAP
Integrated user management library
J2EE server 2
UAP
Integrated user management library
J2EE server 3
UAP
Integrated user management library
Change the password.
LDAP directory server master1
LDAP directory server master2
LDAP directory server slave1
LDAP directory server slave2
LDAP directory server slave3

To use the connection failover, ensure that all LDAP directory servers have the same entry tree structure and entry contents.

5.8 API provided by the integrated user management framework

The integrated user management framework provides JSP tag and integrated user management framework libraries. They can be used to develop the application that invokes the authentication process. Use them when needed.

For details about the API used in the integrated user management framework, see *15. APIs Used with the Integrated User Management Framework*.

5.8.1 JSP tag library

The integrated user management framework provides the JSP tag library that enables the JSP to easily use the functions of the integrated user management framework. Application developers (or Web designers) can develop the JSP without worrying about the Java program-based authentication process.

When the Cosminexus standard login modules are used as the integrated user management framework login modules, the JSP tag library is used to reference the user information.

5.8.2 Integrated user management framework libraries

The integrated user management framework is composed of the following two types of libraries:

- User authentication library

  This is the JAAS-supported user management library that is used to authenticate users based on the information in the user information repository built in the LDAP directory server, and it provides the authenticated user information to the application.

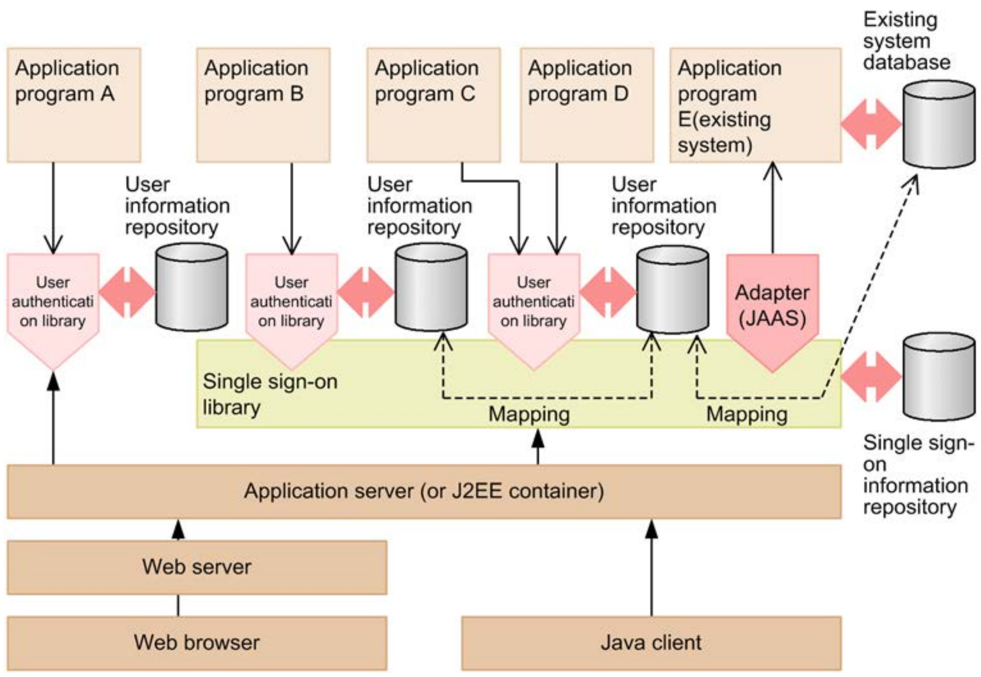
- Single sign-on library

  This is the library that is used to map users based on the user mapping information in the single sign-on information repository to implement single sign-on.

The following figure shows the positioning of the user authentication and single sign-on libraries.

The user authentication library can alone be used for application authentication. It also can work with the single sign-on library to authenticate mapped users.

Figure 5-23: Positioning of user authentication and single sign-on libraries


Application program A
Application program B
Application program C
Application program D
Application program E(existing system)
Existing system database
User information repository
User information repository
User information repository
User authentication library
User authentication library
User authentication library
Adapter (JAAS)
Single sign-on library
Mapping
Mapping
Single sign-on information repository
Application server (or J2EE container)
Web server
Web browser
Java client

5.9 Implementation of user authentication based on the integrated user management framework

The integrated user management framework is the framework that implements user authentication in integrated user management. It provides JAAS-based user authentication as the Cosminexus standard login modules. The integrated user management framework facilitates implementation of user authentication without specially developed authentication modules.

The integrated user management framework uses API to implement user authentication. It can also use the tag library that facilitates the implementation of user authentication. When API or the tag library is used to implement user authentication, a servlet or JSP is used to call the login module to perform user authentication. The invoked login module performs authentication and then logs out.

When authentication modules other than the Cosminexus standard login modules are used, they should be created as custom login modules. To learn more about the types of login modules and the login process flow, see *5.2.3 Overview of Java Authentication and Authorization Service (JAAS)-based user authentication* and *5.2.6 Integrated user management process flow*.

The following sections describe how API and the tag library are used to implement user authentication and the method used to create custom login modules.

Reference note:

Cosminexus provides the sample program used to check the operations of the integrated user management framework. Follow the operation procedures stored in the following file.

Cosminexus installation directory`\manager\examples\ua\index.html`

5.10 Implementation of API-based user authentication

This section describes how API is used to implement sessions from login to logout. The following description is also included here.

- Implementation for enhanced support of authentication password encryption

To learn more about the functionality and grammar of API provided by the integrated user management framework, see the *uCosminexus Application Server API Reference Guide*.

5.10.1 Implementation of the API-based login session

When the integrated user management framework is used to authenticate users, the process must be implemented that uses a servlet or JSP to invoke the login module at the time of login. The settings must be stored in the JAAS configuration file to use login modules. For details about the settings in the JAAS configuration file, see *14.2 jaas.conf (JAAS configuration file)*.

The following is an example of API-based login implementation.

```
<%@ page import="com.cosminexus.admin.auth.callback.WebPasswordHandler" %>
<%@ page import="javax.security.auth.login.LoginContext" %>
...
<%LoginContext lc = new LoginContext("Portal",
    new WebPasswordHandler(request, response, null, "login.html", true));
  try { lc.login(); } catch (LoginException e) { ... }
%>
...
```

The above example specifies to create an instance from the LoginContext class and to use the authentication module specified in the Portal entry of the JAAS configuration file as the argument. If com.cosminexus.admin.auth.name and com.cosminexus.admin.auth.password parameters are set in `request`, these parameters are used to authentication users. Otherwise, `login.html` is invoked to obtain authentication information (user ID and password) based on the user.

5.10.2 Implementation of the API-based session to obtain user IDs

After authentication is completed, the authenticated user ID is stored in the Subject as the Principal object (java.security.Principal). The following is an example of implementing the session to obtain the user ID.

```
<%@ page import="javax.security.auth.Subject" %>
<%@ page import="java.security.Principal" %>
...
<%
  ...
  Subject subject = lc.getSubject();
  Principal principal = (Principal)subject.getPrincipals().iterator().next();
  String userid = principal.getName();
%>
...
```

The above example fetches the `iterator`, which stores the Principal from the Subject, converts the first value in the `iterator` to the Principal object, and then uses the `getName` method of the Principal object to obtain the user ID.

5.10.3 Implementation of the API-based session to obtain user attributes

To obtain user attributes, it is required to specify the list of attributes that should be obtained at the time of login. The following is an example of implementing the login process that specifies the list of user attributes.

```
<%@ page import="com.cosminexus.admin.auth.callback.WebPasswordHandler" %>
<%@ page import="com.cosminexus.admin.auth.AttributeEntry" %>
<%@ page import="javax.security.auth.login.LoginContext" %>
...
<%
  AttributeEntry[] attributes = new AttributeEntry[2];
  attributes[0] = new AttributeEntry("cn", "full name", null);
  attributes[1] = new AttributeEntry("employeeNumber", "employee ID", null);
  LoginContext lc = new LoginContext("Portal",
    new WebPasswordHandler(request, response, attributes, "login.html", true));
  try { lc.login(); } catch (LoginException e) { ... }
%>
...
```

The above example obtains the specified attributes from the repository and assigns them to the UserAttributes object. These objects are managed as the `java.lang.Object` type. The following is an example of implementation in which the attributes obtained from the repository are assigned to the UserAttributes object.

```
LoginContext lc = new ...            // This is to instantiate the LoginContext class
...
Subject subject  = lc.getSubject();
Iterator it = subject.getPublicCredentials().iterator();
UserAttributes ua= (UserAttributes)it.next();             // This is to store the
...                                               // UserAttributes reference in ua.
```

As shown below, the `getAttribute` method is used to obtain the attribute value in String from the UserAttributes object.
`String role = (String)ua.getAttribute("Portal Role");`

The following is an example of implementing the session to obtain the user attribute by using the `getAttribute` method.

```
<%@ page import="com.cosminexus.admin.auth.UserAttributes" %>
<%@ page import="javax.security.auth.Subject" %>
...
<%
  ...
  Subject subject = lc.getSubject();
  UserAttributes attrs =
(UserAttributes)subject.getPublicCredentials().iterator().next();
  String fullname = (String)attrs.getAttribute("full name");
  String eid = (String)attrs.getAttribute("employee ID");
%>
...
```

5.10.4 Implementation of the session to register the successfully authenticated subject to HttpSession

The object that inherits the `java.io.Serializable` interface must be assigned to HttpSession. Store the Subject that inherits the `java.io.Serializable` interface in HttpSession instead of the LoginContext instance, which was created at the time of login. The stored Subject is necessary for logout implementation. The following is an example of implementation in which the Subject is stored in HttpSession (the line in bold letters).

```
<%
  LoginContext lc = new LoginContext("Portal",
    new WebPasswordHandler(request, response, null, "login.html", true));
  try {
    lc.login();
    session.setAttribute("ExampleSubject", lc.getSubject());
  } catch (LoginException e) { ... }%
>
...
```

To inherit the user attributes that have been associated with the Subject after login (UserAttributes) by using the session failover functionality, the Subject and the user attributes must be stored in HttpSession. The following is an example of implementation in which the Subject and the user attributes are stored in HttpSession (the line in bold letters).

```
<%
  LoginContext lc = new LoginContext("Portal",
    new WebPasswordHandler(request, response, null, "login.html", true));
  try {
    lc.login();
     session.setAttribute("ExampleSubject", lc.getSubject());
     session.setAttribute("ExampleCredential",
lc.getSubject().getPublicCredentials().iterator().next());
  } catch (LoginException e) { ... }%
>
...
```

5.10.5 Implementation of the API-based logout session

To perform the logout process, the logout session uses the Subject registered in HttpSession, as described in *5.10.4 Implementation of the session to register the successfully authenticated subject to HttpSession*, to re-create LoginContext. It then deletes the Subject registered in HttpSession. It also deletes the user attributes if they are registered in HttpSession. The following is an example of logout implementation if the user attributes are registered in HttpSession.

```
<%
  try {
    Subject subject = (Subject)session.getAttribute("ExampleSubject");
    LoginContext lc = new LoginContext("Example", subject);
    session.removeAttribute("ExampleCredential");
    session.removeAttribute("ExampleSubject");
    lc.logout();
  } catch (LoginException e) { ... }
%>
...
```

To complete logout when the session times out, assign the object that implements the HttpSessionBindingListener interface to the HttpSession object. The following is an example of logout implementation in which logout is completed when the session times out.

```
<%
  LoginContext lc = new LoginContext("Portal",
    new WebPasswordHandler(request, response, null, "login.html", true));
  try {
    lc.login();
     session.setAttribute("logoutObject",
        new MyListener("Portal", "ExampleSubject", "ExampleCredential"));
    session.setAttribute("ExampleSubject", lc.getSubject());
    session.setAttribute("ExampleCredential",
    lc.getSubject().getPublicCredentials().iterator().next());

  } catch (LoginException e) { ... }
%>
<%!
  class MyListener implements
          HttpSessionBindingListener, java.io.Serializable {
      String name;
      String subjectName;
      String attrsName;
      public MyListener(String name, String subjectName, String attrsName) {
          this.name = name;
          this.subjectName = subjectName;
          this.attrsName = attrsName;
      }
      public void valueBound(HttpSessionBindingEvent ev) {}
      public void valueUnbound(HttpSessionBindingEvent ev) {
          Subject subject =
           (Subject)ev.getSession().getAttribute(subjectName);
          ev.getSession().removeAttribute(attrsName);
          ev.getSession().removeAttribute(subjectName);
          try {
              LoginContext ctx = new LoginContext(name, subject);
              ctx.logout();
          } catch (LoginException e) {}
      }
  }
%>
```

5.10.6 Implementation of enhanced support of authentication password encryption

Password authentication is possible even if passwords are not encrypted in the default encryption methods (SHA-1 or MD5) or in plain text. To provide enhanced encryption support, implementation classes must be created in advance.

This section describes the login modules that provide enhanced encryption support and the method used to implement classes for enhanced encryption support. To get an overview of enhanced encryption support, see *5.3.9 Enhanced support of authentication password encryption*.

(1) Login modules that provide enhanced encryption support

WebPasswordLoginModule and WebPasswordJDBCLoginModule provide enhanced support of authentication password encryption.

(2) Method used to implement classes for enhanced encryption support

To achieve enhanced encryption support, the `com.cosminexus.admin.auth.security.PasswordCryptography` class must be inherited. The created class should be stored in the following directories as a class file.

- Windows:

  Cosminexus installation directory`\manager\modules`

- UNIX:

  `/opt/Cosminexus/manager/modules`

The directories can be changed by the `com.cosminexus.admin.auth.custom.modules` option in the integrated user management configuration file (`ua.conf`).

The following is an example of implementation in which the `byte` arrays are compared in the SHA-1 format.

```
package my;

import com.cosminexus.admin.auth.security.PasswordCryptography;
import java.security.*;

public class CustomCryptography implements PasswordCryptography
{
  public byte[] encrypt (byte[] plain) {
    byte[] encryptedPassword = null;
    try{
      MessageDigest md = MessageDigest.getInstance("SHA");
      md.update(plain);
      encryptedPassword = md.digest();
    } catch (NoSuchAlgorithmException e) {
      encryptedPassword = plain;
    }
    return encryptedPassword;
  }
}
```

5.10.7 Notes on API-based implementation

This section contains the notes on the implementation of API-based user authentication.

(1) Notes on implementing login and logout sessions

When logins and logouts are implemented without using the Subject and when the LoginContext instance created at the time of login is used at the time of logout, logout may fail depending on the login module settings.

Use the Subject when implementing login and logout. The following is an example of the implementation that should be avoided.

- Login and logout implementation that should be avoided

```
<%LoginContext lc = new LoginContext("Portal",
    new WebPasswordHandler(request, response, null, "login.html", true));
  try { lc.login(); } catch (LoginException e) { ... }
  session.setAttribute("loginContext", lc);
%>
...
<%
  LoginContext lc = (LoginContext)event.getSession().getAttribute("loginContext");
  try { lc.logout(); } catch (LoginException e) { ... }
%>
...
```

Note: The lines in bold letters must not be included in implementation.

- Login and session timeout logout implementation that should be avoided

```
...
<%LoginContext lc = new LoginContext("Portal",
    new WebPasswordHandler(request, response, null, "login.html", true));
  try { lc.login(); } catch (LoginException e) { ... }
  session.setAttribute("loginContext", lc);
%>
...

<%@ page import="javax.security.auth.login.LoginContext" %>
...
<%  session.setAttribute("logoutObject",
                 new HttpSessionBindingListener() {
                   public void valueBound(HttpSessionBindingEvent event) {}
                   public void valueUnbound(HttpSessionBindingEvent event) {
                     LoginContext lc =
(LoginContext)event.getSession().getAttribute("loginContext");
                     try { lc.logout(); } catch (LoginException e) { ... };
                   }
                 });
%>
...
```

Note: The lines in bold letters must not be included in implementation.

(2) Notes on implementing the sessions to reference and obtain user information

When implementing the sessions to reference and obtain user information, please note that:

- Changes in the UserAttributes object values are not applied to the repository. The obtained attributes are not modified in the user authentication library.
- The attributes registered in the UserAttributes object is in the String type only.
- If no attributes are specified in the attribute list, a null character is assigned.

5.11 Implementation of tag library-based user authentication

This section describes how the tag library is used to implement sessions from login to logout. The description on copying necessary files and defining DD is also included here. To learn more about the tag library and the tag attributes, see *16. Tag Library Used with the Integrated User Management Framework*.

5.11.1 Implementation of tag library-based login session

When the integrated user management framework is used to perform user authentication, the process must be implemented that uses a servlet or JSP to invoke the login module at the time of login. The settings must be stored in the JAAS configuration file to use login modules. For details about the JAAS configuration file settings, see *14.2 jaas.conf (JAAS configuration file)*.

To use the JSP `<ua:login/>` tag for login, the `com.cosminexus.admin.auth.name` and `com.cosminexus.admin.auth.password` parameters must be set in the HTTP request object. First, prepare the following login form so that the parameters can be set up.

```
<html>
<body>
<form action="auth.jsp" method="post">
<table>
<tr>
  <td>username</td>
  <td><input type="text" name="com.cosminexus.admin.auth.name" /></td>
</tr>
<tr>
  <td>password</td>
  <td><input type="password" name="com.cosminexus.admin.auth.password" />
</td>
</tr>
</table>
<br />
<input type="submit" value="Login" />
<input type="reset" value="Reset" />
</form>
</body>
</html>
```

Next use the `<ua:login/>` tag and the authentication module specified in the "Portal" entry of the JAAS configuration file to log in.

```
<%@ taglib uri="http://cosminexus.com/admin/auth/uatags" prefix="ua" %>
<%@ page errorPage="error.jsp" %>

<ua:login id="lc" entry="Portal" />
...
```

Due to the tag library specification, all exceptions that occurred during the tag process are regarded as JspException. To more minutely detect exceptions that occurred during the processing of the `<ua:login/>` tag, use the `<ua:exception>Body </ua:exception>` tag. In the following example, the exception is transferred to the exception detection JSP (`loginError.jsp`).

```
<%@ taglib uri="http://cosminexus.com/admin/auth/uatags" prefix="ua" %>

<ua:login id="lc" entry="Portal" excepId="ex" excepScope="session" />
<ua:exception name="ex" ><jsp:forward page="loginError.jsp" /></ua:exception>
...
```

Based on the exception, the exception detection JSP (`loginError.jsp`) selects the message to be returned.

```
<%@ page contentType="text/html; charset=Shift_JIS" %>
<%@ taglib uri="http://cosminexus.com/admin/auth/uatags" prefix="ua" %>

<html>
<body>
<ua:exception name="ex" type="javax.security.auth.login.FailedLoginException">

The user ID or password is incorrect.<br />
</ua:exception>
<ua:exception name="ex" type="javax.security.auth.login.AccountExpiredException">

The account has expired.<br />
</ua:exception>
<ua:exception name="ex" type="javax.security.auth.login.CredentialExpiredException">

The password has expired.<br />
</ua:exception>
<ua:exception name="ex" >

An exception occurred.<br />
<%= e.toString() %><br />
</ua:exception>
</body>
</html>
```

Hint:

How to check the login state

By adding the `<ua:notLogin>Body</ua:notLogin>` tag at the top of each JSP page, you can check the login status before processing the JSP page.

```
<%@ page contentType="text/html; charset=Shift_JIS" %>
<%@ taglib uri="http://cosminexus.com/admin/auth/uatags" prefix="ua" %>
...
<ua:notLogin>

<a href="login.html">Please log in.</a>
</ua:notLogin>
...
```

5.11.2 Implementation of the tag library-based session to obtain user ID

After authentication is completed, the authenticated user ID can be displayed or obtained by using the `<ua:getPrincipalName>` tag. The following is an example of displaying the user ID.

```
<%@ taglib uri="http://cosminexus.com/admin/auth/uatags" prefix="ua" %>
...
User ID: <ua:getPrincipalName name="lc" />
...
```

The above example specifies the LoginContext identifier (lc) that was specified at the time of login as the name attribute of the `<ua:getPrincipalName>` tag.

The following is an example of obtaining the user ID.

```
<%@ taglib uri="http://cosminexus.com/admin/auth/uatags" prefix="ua" %>
...
<ua:getPrincipalName name="lc" id="userid" />User ID: <%= userid %>
...
```

The above example specifies the `id` attribute in addition to the `name` attribute of the `<ua:getPrincipalName>` tag. The identifier that identifies the instance referencing the user ID is specified as the `id` attribute.

5.11.3 Implementation of the tag library-based session to obtain user attributes

To obtain user attributes, it is required to specify the list of attributes that should be obtained by using the `<ua:attributeEntries>` tag. The following is an example of implementing the session to specify the list of user attributes.

```
<%@ taglib uri="http://cosminexus.com/admin/auth/uatags" prefix="ua" %>

<ua:attributeEntries id="ae">
  <ua:attributeEntry attrName="cn" alias="full name" />
  ...
</ua:attributeEntries>
<ua:login id="lc" entry="Portal" attrEntName="ae" />
...
Full Name: <ua:getAttribute name="lc" attrName="full name" />
...
```

Then, the specified user attributes are obtained by using the `<ua:getAttribute>` tag.

```
<%@ taglib uri="http://cosminexus.com/admin/auth/uatags" prefix="ua" %>

<ua:login id="lc" entry="Portal" attrFile="MyAttrs.csv" />
...
<ua:getAttribute name="lc" attrName="full name" id="fullname" />
Full Name: <%= fullname %>
...
```

Notes on referencing or obtaining user information

- The UserAttributes values are for reference only. Changes in these values are not applied to the repository. The obtained attributes are not modified in the user authentication library.
- The registered attributes are in the `String` type only.
- If no attributes are specified in the attribute list, a null character is assigned.

5.11.4 Implementation of tag library-based logout session

The following is an example of logout implementation.

```
<%@ taglib uri="http://cosminexus.com/admin/auth/uatags" prefix="ua" %>

<ua:logout name="lc" />
...
```

When a `<ua:logout/>` tag that corresponds to the `<ua:login/>` tag is not explicitly specified, logout is implicitly performed when the session is disconnected.

5.11.5 Copying uatags.jar and uatags.tld and defining DD

The files must be copied and edited to use the JSP tag library.

Copy the JAR file (`uatags.jar`) and the tag library descriptor file (`uatags.tld`) for the JSP tag library, and edit the Web application DD (`web.xml`). The procedure is as follows:

1. Copy `uatags.jar` to either WEB-INF\lib (Windows) or WEB-INF/lib (UNIX)

of the Web application to be created.

2. Copy `uatags.tld` to WEB-INF of the Web application to be created.
3. Add the following description to the appropriate location in `web.xml`.

```
<taglib>
  <taglib-uri>http://cosminexus.com/admin/auth/uatags</
taglib-uri>
  <taglib-location>/WEB-INF/uatags.tld</taglib-location>
</taglib>
```

5.12 Implementation of user authentication when using the session failover functionality

Integrated user management uses the session failover functionality to pass the login state at the time of failure to another J2EE server.

This section describes the information that can be inherited by the session failover functionality, how to implement login and logout when using the session failover functionality, how to define DD, and the notes on implementation.

5.12.1 Session and authentication information that can be inherited by the session failover functionality

The following table lists the information that can be inherited by the session failover functionality. In the table, API refers to the JAAS API, and Custom Tag is the JSP custom tag provided in integrated user management.

Table 5-12: Information that can be inherited by the session failover functionality

Information		**Cosminexus Standard Login Module**		**Custom Login Module**	
		API	**Custom Tag**	**API**	**Custom Tag**
Integrated user management session information	Login state (whether users are logging in or not)	Auto	Auto	Auto	Auto
	Registered user ID and realm name	Auto	Auto	Auto	Auto
	Password	O	O	n/a	n/a
Authentication information	Subject	X	X	X	X
	Principal	X	X	X	X
	User attribute (UserAttributes)	Man	Man	n/a	n/a
	Credential (public/private) other than user attributes (UserAttributes)	X	X	X	X

Legend:

Auto: Can be inherited automatically

O: Can be selected depending on the option

Man: Can be inherited manually

X: Cannot be inherited

n/a: Not applicable

5.12.2 Implementation of login and logout when using the session failover functionality

This section describes how to implement login and logout when using the session failover functionality.

(1) Using API

The session failover functionality can inherit objects in the String type only. To inherit user attributes (UserAttributes), it is required to fetch necessary character information and store it in the global session information.

The following is an example of implementing user authentication when using the session failover functionality.

(a) Login (when using the session failover functionality)

When implementing login, register in the global session the user ID and user information necessary when login is successful.

```
<%
  LoginContext lc = new LoginContext("Example",
    new WebPasswordHandler(request, response, null, "login.html", true));
  try {
    lc.login();
    Subject subject = lc.getSubject();
    session.setAttribute("ExampleSubject", subject);   // This is to store the Subject
in HttpSession
    String uid = ((Principal)subject.getPrincipals().iterator().next()).getName();

    session.setAttribute("ExampleUserID", uid);   //This is to store the user ID in
the global session.
    UserAttributes attr =
(UserAttributes)lc.getSubject().getPublicCredentials().iterator().next();
    session.setAttribute("telephoneNumber", attr.getAttribute("tel"));

                                                   //This is to store the telephone
number in the global session.
  } catch (LoginException e) { ... }
%>
```

It is recommended that the user ID be registered as read-only in case of the SFO server failure.

(b) Logout (when using the session failover functionality)

Logout uses the Subject obtained at the time of login and the user ID and HttpSession registered in the global session. Note that the Subject object becomes null after failover. Create and specify a new Subject.

```
<%
  try {

   String uid = (String)session.getAttribute("ExampleUserID");//This is to obtain the
user ID from the global session.
    Subject subject = (Subject)session.getAttribute("ExampleSubject");
    LoginContext lc = new LoginContext("Example",

      (subject != null) ? subject : new Subject(), //This is to create a new Subject
when the Subject is null.
     new WebLogoutHandler(session, uid));       //This is to assign WebLogoutHandler
to LoginContext.
    session.removeAttribute("ExampleUserID");
    lc.logout();
  } catch (LoginException e) { ... }
%>
```

(c) Session timeout logout (when using the session failover functionality)

To complete logout when the session times out, assign the object that implements the HttpSessionBindingListener interface to the HttpSession object. Note that the registered object is cleared as a new HttpSession is generated after failover check if the object is present at the time of screen transition, etc., and re-register it when necessary.

```
<%
  LoginContext lc = new LoginContext("Portal",
    new WebPasswordHandler(request, response, null, "login.html", true));
  try {
    lc.login();
    Subject subject = lc.getSubject();
    String uid = ((Principal)subject.getPrincipals().iterator().next()).getName();

    session.setAttribute("PortalSubject", subject);   // This is to store the Subject 
in HttpSession
    session.setAttribute("PortalUserID", uid);   // This is to store the user ID in 
the global session.
    session.setAttribute("myLogoutObject",      // This is to assign the object which 
performs the logout process
      new MyListener("Portal", "PortalUserID", "PortalSubject")); //when the session 
times out to HttpSession.
  } catch (LoginException e) { ... }
%>
<%!
  class MyListener implements                            // The class which performs 
the logout process
      HttpSessionBindingListener, java.io.Serializable {   // when the session times 
out.
    String name;
    String userName;
    String subjectName;
    public MyListener(String name, String userName, String subjectName) {
      this.name = name;
      this.userName = userName;
      this. subjectName = subjectName;
    }
    public void valueBound(HttpSessionBindingEvent ev) {}
    public void valueUnbound(HttpSessionBindingEvent ev) {
      String uid = (String)ev.getSession().getAttribute(userName);
      Subject subject = (Subject)ev.getSession().getAttribute(subjectName);
      try {
        LoginContext ctx = new LoginContext(name,
          (subject != null) ? subject : new Subject(),
          new WebLogoutHandler(ev.getSession(), uid));
        ctx.logout();
      } catch (LoginException e) { ... }
    }
  }
%>
```

(d) Checking the login state (when using the session failover functionality)

The object that is registered for logout and implements the HttpSessionBindingListener interface is cleared after failover. Check if the object is present and re-register it when necessary.

```
<%!
  if (LoginUtil.check(request, response)) {
    HttpSession session = request.getSession();
    if (session.getAttribute("myLogoutObject") == null) {
      session.setAttribute("myLogoutObject",
          new MyListener("Portal", session.getAttribute("userid"), "PortalSubject");
    }
  } else {

    //This is the process when login is not done
  }
%>
```

(2) Using the tag library

The session failover functionality limits the use of the following tags, as user attributes (`UserAttributes`) cannot be inherited.

`<ua:getAttributes/>`

A null character is assigned to the script variable when the `id` attribute is specified. Otherwise, nothing is done.

`<ua:getAttribute/>`

A null character is assigned to the script variable when the `id` attribute is specified. Otherwise, nothing is done.

`<ua:getAttributeNames/>`

A null character is assigned to the script variable when the `id` attribute is specified. Otherwise, nothing is done.

To inherit user attributes (`UserAttributes`), it is required to fetch necessary character information and store it in the global session information.

(3) Notes on using single sign-on

Do not allow multiple logins in the same realm or by the same user. When multiple logins occur, single sign-on may not work properly.

5.12.3 Defining DD when using the session failover functionality

The session failover functionality in integrated user management can be enabled by adding the filter definition to DD (`web.xml`).

This section introduces GSInfoKey-RW (the read-only information used as the global session information) and GSInfosLengthMax (the maximum size of the global session information), which must be defined to use the session failover functionality.

(1) Defining GSInfoKey-RW (integrated user management and the session failover functionality)

The following package should be defined.
`com.cosminexus.admin.auth.user_data`

(2) Defining GSInfosLengthMax (integrated user management and the session failover functionality)

Calculate the value used in integrated user management by using any of the following formulas, and define the value derived by adding this to the value used by the application. The fixed values such as 1,500 and 1,200 in the formulas are necessary to save the integrated user management information.

- Using API

  When the password save functionality is not used

  {1,500 + (length of realm names + maximum length of user names + 1,200) x number of realms} x 2

  When the password save functionality is used

  {1,500 + (length of realm names + maximum length of user names + maximum length of passwords + 1,200) x number of realms} x 2

- Using the tag library

  When the password save functionality is not used

  {1,500 + (length of realm names + maximum length of user names + length of `login` tag `id` attribute values + length of `login` tag `entry` attribute values + 1,200) x number of realms} x 2

  When the password save functionality is used

  {1,500 + (length of realm names + maximum length of user names + length of `login` tag `id` attribute values + length of `login` tag `entry` attribute values + maximum length of passwords + 1,200) x number of realms} x 2

Example of calculating GSInfosLengthMax used in integrated user management

Conditions

- API is used.
- The password save functionality is not used.
- The length of realm names is 10 characters.
- The maximum length of user names is 20 characters.
- The number of single sign-on systems (realms) is 3.

Formula

{1,500 + (10 + 20 + 1,200) x 3} x 2=10,380

If value is not correctly defined, the session failover functionality will not work properly.

To disable the session failover functionality in integrated user management when using the session failover functionality, change the settings in the integrated user management configuration file (`ua.conf`). For details about the integrated user management configuration file, see *14.3 ua.conf (integrated user management configuration file)*.

(3) Example of defining DD (integrated user management and the session failover functionality)

The following is an example of defining DD when using the session failover functionality in integrated user management.

```
   :
<filter>
    <filter-name>com.hitachi.software.was.sfo.web.SFOFilter</filter-name>
    <filter-class>com.hitachi.software.was.sfo.web.SFOFilter</filter-class>
    <init-param>
        <param-name>GSInfoKey-RW</param-name>
        <param-value>com.cosminexus.admin.auth.user_data</param-value>
    </init-param>
    <init-param>
        <param-name>GSInfoKey-R</param-name>
        <param-value></param-value>
    </init-param>
    <init-param>
        <param-name>GSInfosLengthMax</param-name>
        <param-value>10380</param-value>
    </init-param>
    <init-param>
        <param-name>SFOServerName</param-name>
        <param-value>sfo_server1</param-value>
    </init-param>
    <init-param>
        <param-name>GIDCookieName</param-name>
        <param-value>GSESSIONID</param-value>
    </init-param>
    <init-param>
        <param-name>LockTimeout</param-name>
        <param-value>30</param-value>
    </init-param>
</filter>
   :
```

5.13 Implementation of custom login module-based user authentication

To authenticate users by using modules other than the Cosminexus standard login modules, create custom login modules and integrate them with the Cosminexus standard login modules.

5.13.1 Implementation for integration with Cosminexus standard login modules

JAAS allows you to invoke multiple authentication modules sequentially in a single authentication. These authentication modules use the Map object (sharedState) passed to the third parameter of the LoginModule interface initialize method to pass information. This section provides the information that is added by each of the Cosminexus standard login modules. Note that DelegationLoginModule and WebPasswordJDBCLoginModule do not add information.

(1) WebPasswordLoginModule, WebCertificateLoginModule, and WebPasswordLDAPLoginModule

WebPasswordLoginModule, WebCertificateLoginModule, and WebPasswordLDAPLoginModule add the following information to sharedState before invoking the next LoginModule.

Key: com.cosminexus.admin.auth.userattributes

Value type: UserAttributes

Description: References the UserAttributes object, which stores the user attributes associated with the Subject.

Setup timing: Just before the end of the `commit` method

(2) WebSSOLoginModule

If the user has been already authenticated in one session, WebSSOLoginModule obtains the information used for authentication in other realms from the user mapping and adds the following information to sharedState before invoking custom login modules. No information is added when the user has not been authenticated in the session or when there is no authentication information in the user mapping.

Key: com.cosminexus.admin.auth.sso.userid

Value type: String

Description: The value defined in USERID of the user mapping

Setup timing: Just before invoking the `login` method of the custom login module

Key: com.cosminexus.admin.auth.sso.secdat

Value type: String

Description: The value defined in SECRETDATA of the user mapping; the value is decrypted before being stored in sharedState.

Setup timing: Just before invoking the `login` method of the custom login module

Key: com.cosminexus.admin.auth.sso.pubdat

Value type: String

Description: The value defined in PUBLICDATA of the user mapping

Setup timing: Just before invoking the `login` method of the custom login module

The above keys can be changed in the configuration file of the integrated user management framework. If a custom login module exists and can obtain authentication information from sharedState, the keys can be tailored to the custom login module specifications. For details about the configuration file settings, see *14.2 jaas.conf (JAAS configuration file)*.

5.13.2 Points to remember when implementing custom login modules

When creating custom login modules, the LoginModule interface that is the JAAS SPI should be inherited to implement the necessary process. This section provides the points to remember when implementing custom login modules and the LoginModule interface, as well as when implementing the Principal object, which manages user IDs.

(1) Points to remember when implementing the LoginModule interface

- `login` method

  To support single sign-on, first determine if the user ID and password are not specified in sharedState. The name used to obtain the user ID and password from sharedState can be specified in the configuration file of the integrated user management framework.

- `commit` method

  Set the Principal object to Subject. When there are multiple Principal objects, WebSSOLoginModule and DelegationLoginModule use the first found Principal object to register the user ID in the integrated user management session. For single sign-on, it is used to recognize the first logged-in user ID.

- `logout` method

  The `logout` method deletes the Principal object and Credentials (such as user attributes) that are associated with the Subject by using the `commit` method. It also releases the resources secured by login.

  When the `logout` method is used, the following phenomena may occur.

- No Credentials are assigned to the Subject when the `logout` method is invoked.
- At the time of logout, the member attribute values that are set by the `commit` or `login` method of the custom login modules cannot be referenced.

The phenomenon that no Credentials are assigned may be caused by the fact that no Credentials are contained in the serializable Subject object.

On the other hand, the phenomenon such that the member attribute values cannot be referenced at the time of logout may be caused by the fact that the JAAS LoginContext (including LoginModule) is not a serialized object. As LoginContext stores the Subject object in HttpSession and generates a new login module instance from the Subject object to log out, the member attribute values set by the `commit` or `login` method cannot be referenced.

(2) Points to remember when implementing the Principal object

Implement the Principal object by inheriting the java.security.Principal and java.io.Serializable interfaces.

5.13.3 Examples of implementing custom login modules

The following are examples of implementing custom login modules and the Principal object.

The first example of implementing a custom login module does not use the session failover functionality. The example of using the session failover functionality includes the portion that is different from the example of not using the session failover functionality.

(1) Example of implementation not using the session failover functionality

The following is an example of implementing a custom login module without the session failover functionality.

```
/**

* The LoginModule implementation class is created by inheriting the LoginModule
interface.
*
*/
public class ExampleLoginModule implements LoginModule
{

    // The following are used to reference the parameter values passed to the
initialize() method.
    private Subject subject;
    private CallbackHandler callbackHandler;
    private Map sharedState;
    private Map options;
    // The following define the name used to obtain the user ID and password values
from sharedState.
    // "simple.login.username" and "simple.login.password" can be specified in
    // the integrated user management configuration file.
    private static final String USERNAME = "simple.login.username";
    private static final String PASSWORD = "simple.login.password";


    // The following stores the user ID used for authentication. The value is set by
login() and referenced by commit().
    private String username;

    // The following stores success or failure of login(). "True" means login()
succeeded while "false" means login() failed.
    // The value is set by login() and referenced by commit().
    private boolean succeeded;

    // The following stores success or failure of commit(). True means commit()
succeeded while false means commit() failed.
    // The value is set by commit() and referenced by abort().
    private boolean commitSucceeded;


    /**

    * The initialize() method stores the parameters passed to the arguments in the
member variables.
    * It also performs initialization when needed.
    * (It is called once when this class is instantiated.)
    *
    */
    public void initialize(Subject subject, CallbackHandler callbackHandler, Map
sharedState, Map options)
    {
        this.subject = subject;
        this.callbackHandler = callbackHandler;
        this.sharedState = sharedState;
        this.options = options;
    }
    /**

    * The login() method obtains the user ID needed for authentication and performs
authentication.
    * In this example, "succeeded" is set to true when authentication succeeds. The
authenticated user ID is
    * stored in "username".
    *
    */
    public boolean login()
        throws LoginException
    {

      // To support single sign-on, the user ID and password are obtained to be stored
in sharedState.
        this.username = (String)this.sharedState.get(USERNAME);
        String password = (String)this.sharedState.get(PASSWORD);



        // When no user ID is in sharedState, CallbackHandler is used to obtain
        // the user ID and password. (This example assumes that WebPasswordHandler
assign the values to
        // WebPasswordCallback.)

        if (this.username == null || this.username.length() == 0) {
            WebPasswordCallback webpc = new WebPasswordCallback();
            webpc.setOption(WebPasswordCallback.GETPW);
            Callback callbacks[] = new Callback[] { webpc };
            try {
                this.callbackHandler.handle(callbacks);
            }
            catch (Exception ex) {

                // Exception handling is performed.
            }

            // The user ID and password are obtained from Callback.
            this.username = webpc.getName();
            password = webpc.getPassword();
        }
        // The following checks if the user ID used for authentication is present. If
no user ID is present, exception is returned.
        if (this.username == null || this.username.length() == 0) {
            throw new FailedLoginException();
        }


        // The application authentication process is performed.
        // When authentication is successful, "succeeded" is set to "true".


        /* Enter the authentication process here. */

        if (!succeeded) {
            throw new FailedLoginException();
        }
        return succeeded;
    }


    /**

    * The commit () method associates the Principal object to the Subject to indicate
that authentication is completed.
    * (SimplePrincipal is a class which is created by inheriting the Principal and
Serializable interfaces.)
    *
    */
    public boolean commit()
        throws LoginException
    {

        // The following associates the Principal object to the Subject to allow join
the login session managed by integrated user management
        // and support of the single sign-on function.
        this.subject.getPrincipals().add( new SimplePrincipal(this.username) );


        /* Enter the process which associates the user attributes to the Subject. */

        return this.commitSucceeded = true;
    }
    /**

    * The abort() method is invoked when the login() or commit() method
    * failed.
    *
    */
    public boolean abort()
        throws LoginException
    {
        if (this.commitSucceeded) {

          // This releases the Principal and user attributes associated to the Subject.
            // In this example, the logout() method is invoked.
            logout();
        }
        return true;
    }


    /**

    * The logout () method is invoked to log out.
    * This method is used to release the Principal and user attributes associated to
the Subject.
    *
    *
    */
    public boolean logout()
        throws LoginException
    {

        // Enter the process which deletes the Principal and user attributes from the
Subject.
        // Add the process which releases the resources secured by the login() method.
        return true;
    }
}
```

(2) Example of Implementation using the session failover functionality

The `login` and `commitment` methods are the same as the example not using the session failover functionality. The following shows the difference in the `logout` method implementation.

```
/**

 * The LoginModule implementation class is created by inheriting the LoginModule
interface.
 */
public class ExampleLoginModule implements LoginModule
{
   :
  /**

   * The logout () method is invoked to log out.
   * This method is used to release the resources secured at the time of login.
   */
  public boolean logout() throws LoginException
  {
    if (callbackHandler != null) {
        WebLogoutCallback callback = new WebLogoutCallback();
        try {
            callbackHandler.handle(new Callback[]{callback});
        } catch (Exception e) { ... }
        String uid = callback.getUserID();
        HttpSession session = callback.getSession();

        // Add the process which releases the resources secured by the login() method.
        // to delete the information registered in the global session, etc.
    }
    return true;
  }
}
```

The `logout` method release the resources secured at the time of login. Note that the session failover functionality does not fail over the Subject and Principal.

(3) Example of implementing the Principal object

Create the Principal object by inheriting the java.security.Principal and java.io.Serializable interfaces. The following is an example of implementing the Principal object.

```
import java.security.Principal;
import java.io.Serializable;

/**

* The Principal implementation class is created by inheriting the Principal and
Serializable interfaces.
*
*/
public class SimplePrincipal implements Principal, Serializable
{
  private String name;

  public SimplePrincipal(String name) {
    if (name == null) throw new NullPointerException();
    this.name = name;
  }
  public String getName() { return name; }
  public String toString() { return getName(); }
  public boolean equals(Object o) {
    if (o == null) return false;
    if (this == o) return true;
    if (!(o instanceof SimplePrincipal)) return false;
    SimplePrincipal rhs = (SimplePrincipal)o;
    if (getName().equals(rhs.getName())) return true;
    return false;
  }
  public int hashCode() { return getName().hashCode(); }
}
```

5.14 Procedures to set up the integrated user management function

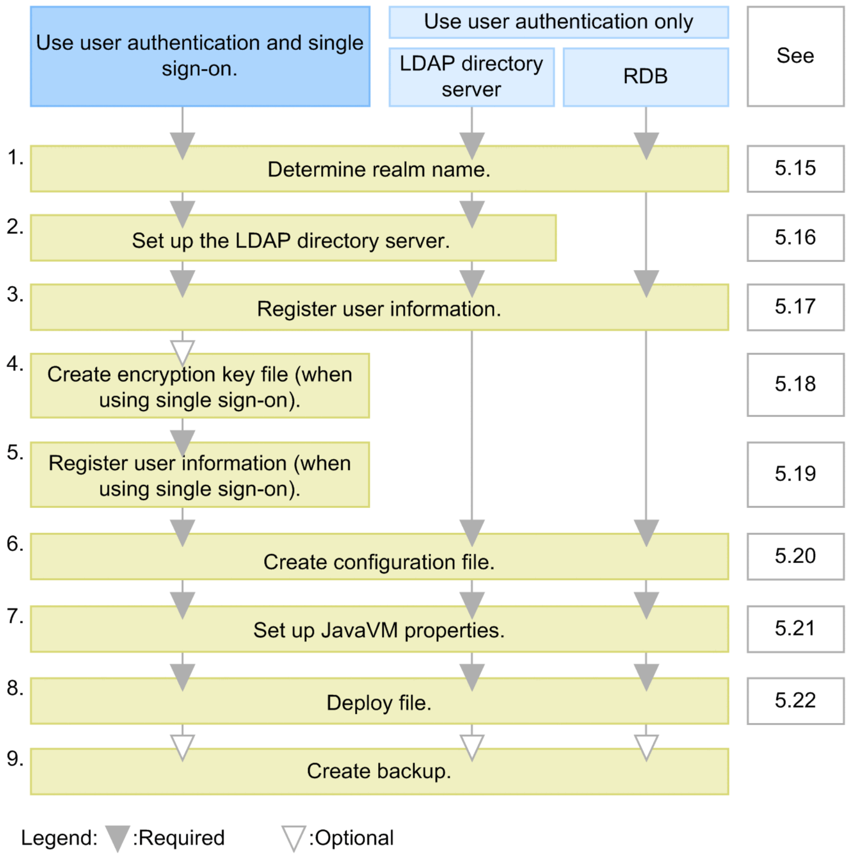
This section describes the procedures used to set up the integrated user management function.

Cosminexus can perform the integrated management of users who log into the Cosminexus-based systems. Integrated user management associates the user information managed by each of the J2EE applications so that the user who logs into one J2EE application can log into other J2EE applications. To use the integrated user management function, it is required to set up the LDAP directory server, which stores user authentication information, and the integrated user management configuration file.

It is also necessary to create the authentication process program that uses the JAAS API, the integrated user management API provided by Cosminexus, and the JSP tag library to invoke standard login modules. Custom login modules must be created to authenticate users in a specific way to the application. To learn more about creating custom login modules, see *5.13 Implementation of custom login module-based user authentication*.

The following figure shows the procedures used to set up the integrated user management function.

Figure 5-24: Procedures used to setup the integrated user management function


Use user authentication and single sign-on.
Use user authentication only
LDAP directory server
RDB
See
1. Determine realm name. 5.15
2. Set up the LDAP directory server. 5.16
3. Register user information. 5.17
4. Create encryption key file (when using single sign-on). 5.18
5. Register user information (when using single sign-on). 5.19
6. Create configuration file. 5.20
7. Set up JavaVM properties. 5.21
8. Deploy file. 5.22
9. Create backup.
Legend: :Required :Optional

The details of steps 1 to 9 in the figure are as follows.

1. Examine how to manage users and determine the range (realm) to which the same authentication is applied.

   Examine the unit used to manage users and determine the realm name. To learn more about determining the realm name, see *5.15 Determination of realm names*.

2. Set up the LDAP directory server.

   The LDAP directory server is needed to use single sign-on, as it is used to manage the single sign-on user information. To learn more about setting up the LDAP directory server, see *5.16 LDAP directory server setup*.

   Skip this step when only the default user authentication provided by RDB (HiRDB, Oracle, etc.) is used.

3. Register the user information used for user authentication in the LDAP directory

server or RDB.

To learn more about registering the user information to the LDAP directory server, see *5.17 Registration of user information*. Cosminexus specifies the standard DIT structure of the user management repository stored in the LDAP directory server. To learn more about the repository structure, see *5.2.4 Management method of user information used for integrated user management*.

For details about registering the user information to RDB, see the RDB documentation.

4. When single sign-on is used and the single sign-on user information should be encrypted, create the encryption key file used to encrypt and decrypt the user information.

   To learn more about creating the encryption key file, see *5.18 Creation of encryption key files (When using single sign-on)*.

   Skip this step when single sign-on is not used or the user information is not needed to be encrypted.

5. When single sign-on is used, register the single sign-on user information to the LDAP directory server.

   To learn more about registering the single sign-on user information to the LDAP directory server, see *5.19 Registration of user information (When using single sign-on)*. Cosminexus specifies the standard DIT structure of the single sign-on user management repository stored in the LDAP directory server. To learn more about the repository structure, see *5.2.4 Management method of user information used for integrated user management*.

   Skip this step when single sign-on is not used.

6. Create configuration files.

   The following two files should be created.

   - `jaas.conf` (the JAAS configuration file)
   - `ua.conf` (the integrated user management configuration file)

   For details about creating the configuration files, see *5.20 Creating configuration files*.

7. Set up the JavaVM properties.

   For details about setting up the JavaVM properties, see *5.21 JavaVM property setup*.

8. Deploy the EAR file used in integrated user management.

   To learn more about deploying the file, see *5.22 Deployment of files*.

9. Create a backup of the information used in integrated user management when needed.

   Use the commands provided by the LDAP directory service or the directory gateway to backup and restore the LDAP directory server repository. For details, see the LDAP directory server documentation.

   Make sure to backup `jaas.conf`, `ua.conf`, and the encryption key files.

Reference note:

Integrated user management uses the session failover functionality to pass the login state to another J2EE server. The session failover filter settings to use the session failover functionality depend on whether the integrated user management function is used or not.

5.15 Determination of realm names

Examine the unit used to manage users and determine the realm name. The name is used to authenticate users in the JAAS-based user management. Specify the name as a corresponding login module option in `jaas.conf` (the JAAS configuration file). Generally, it is recommended to use easy-to-understand names indicating a group of applications that share user management such as *Soumu System*.

5.16 LDAP directory server setup

This section describes how to set up the LDAP directory server.

The following are the procedures used to set up the LDAP directory server.

1. Install the LDAP directory server.

   For details, see *5.16.1 Installation of the LDAP directory server*.

2. Register users to the LDAP directory server and set access permissions.

   For details, see *5.16.2 User registration and access permission setup*.

3. When single sign-on is used, register the object class and user definition attributes that are specific to the single sign-on library to the LDAP directory server, in order to extend the object class and user attribute definitions.

   For details, see *5.16.3 Extension of object class and user definition attributes*.

5.16.1 Installation of the LDAP directory server

Cosminexus uses the LDAP directory server as the repository that manages user information. Install and initialize the LDAP directory server. To learn more about installation and initialization, see the LDAP directory server documentation.

The following are examples of usable LDAP directory servers. For details, see the Release Notes.

Examples of usable LDAP directory servers

- Sun Java System Directory Server
- IBM SecureWay Directory (or IBM Directory Server)
- Active Directory

If the LDAP directory server has already been used, check the schemas being used in the LDAP directory server to make sure that the schemas that are specific to single sign-on are not used. For details, see *5.16.3 Extension of object class and user definition attributes*.

5.16.2 User registration and access permission setup

To connect (bind) to the LDAP directory in integrated user management, register the management & reference users and set access permissions.

The management user DN is referred to as the management bind DN, while the reference user DN is referred to as the reference bind DN.

The management bind DN is the bind DN that has all access permissions (Read, Write, Add, Delete, Search, Compare, and Selfwrite) to all the entries below the base entry used in the integrated user management framework and all the attributes assigned to these entries. This is used to register, reference, modify, and delete the user management information in the LDAP directory server.

The reference bind DN is the bind DN that has the Read and Search access permissions to all the entries below the base entry used in the integrated user management framework and all the attributes assigned to these entries. This is used to obtain the user information from the LDAP directory server.

5.16.3 Extension of object class and user definition attributes

When single sign-on is used, register the object class and user definition attributes that are specific to the single sign-on library to the LDAP directory server, in order to extend the object class and user attribute definitions.

The extended object class and user definition attributes are the schemas that are specific to single sign-on library and cannot be shared with other systems. If the LDAP directory server has already been used, check the schemas being used in the LDAP directory server to make sure that the schemas that are specific to single sign-on are not used.

(1) Object Class to be extended in the single sign-on library

The following table shows the object class that is specific to the single sign-on library.

Table 5-13: Object class that is specific to the single sign-on library

Object class	OID	Required attribute	Optional attribute
CosminexusSSOEntry	1.2.392.200010.7.6.21	`objectClass`, `CosminexusSSOEntryID`, `CosminexusSSOUID`	`CosminexusSSOSecretdata`, `CosminexusSSOPublicdata`, `CosminexusSSOMapping`

(2) User definition attributes to be extended in the single sign-on library

The following table shows the attributes that are specific to the single sign-on library.

Table 5-14: Attributes that are specific to the single sign-on library

Attribute	OID	Syntax	Multi-value/single value
`CosminexusSSOEntryID`	`1.2.392.200010.7.4.71`	`cis`	`Single value`
`CosminexusSSOUID`	`1.2.392.200010.7.4.72`	`ces`	`Single value`
`CosminexusSSOSecretdata`	`1.2.392.200010.7.4.73`	`bin`	`Single value`
`CosminexusSSOPublicdata`	`1.2.392.200010.7.4.74`	`ces`	`Single value`

Attribute	OID	Syntax	Multi-value/single value
`CosminexusSSOMapping`	1.2.392.200010.7.4.75	dn	Multi-value

(3) Procedures used to add the object class and user definition attributes to be extended

This section explains the procedures used to add the object class and user definition attributes to be extended with respect to the types of LDAP directory servers.

Sun Java System Directory Server or Oracle Directory Server Enterprise Edition:

Make sure that the LDAP directory server is started and register `uaschema.slapd.ldif` to the LDAP directory server by using the following command:

`ldapmodify -h` *host name* `-p` *port number* `-D` *management bind DN* `-w` *password* `-c -f uaschema.slapd.ldif`

IBM SecureWay Directory or IBM Directory Server:

Make sure that the LDAP directory server is started and register `uaschema.ldif` to the LDAP directory server by using the following command:

`ldapmodify -h` *host name* `-p` *port number* `-D` *bind DN* `-w` *password* `-c -f uaschema.ldif`

Specify the bind DN that has administrative rights.

Active Directory:

1. Change the settings so that the schemas can be changed in Active Directory. Start Microsoft Management Consol (`mmc.exe`) and click **Add or Remove Snap-ins** to add Active Directory schemas. Right-click **Active Directory Schema**, select **Operations Master**, select the **The Schema may be modified on this Domain Controller** check box, and then click the **OK** button.
2. Use the following command to register `uaschema.ad.ldif` to Active Directory (when you want to connect to the domain controller to which you log on as the current logged on user).

   `ldifde -i -c "dc=domain" "`*ToDN*`" -f uaschema.ad.ldif`

Enter the appropriate DN in *ToDN*, which depends on the domain. For example, if the domain is `hitachi.co.jp`, *ToDN* will be `dc=hitachi,dc=co,dc=jp`.

5.17 Registration of user information

This section describes how to register user information in the LDAP directory server and the formatting used to register the user information. For details about the method used to register the user information to RDB, see the RDB documentation.

Cosminexus specifies the standard DIT structure of the user management repository stored in the LDAP directory server. To learn more about the repository structure, see *5.2.4 Management method of user information used for integrated user management*.

There are the following two methods used to register the user information in the user information repository:

- Registration by using commands
- Registration by using the integrated user management framework library

The following subsections explain these methods respectively as well as the formatting used to register the user information and the settings when Active Directory is used.

5.17.1 Registration by using commands

To use the commands provided by Application Server and the LDAP directory server to register user information:

1. Put the user information in the LDIF file.
2. Use the `convpw` command to encrypt the passwords in the LDIF file.

   The passwords specified in the LDIF will be encrypted. For details about the `convpw` command, see *convpw (Password encryption)*.
3. Use the `ldapmodify` command provided by the LDAP directory server to register the encrypted LDIF file in the user information repository.

The LDAP directory server may provide the GUI used to register the user information. For details, see the LDAP directory server documentation.

When you use IBM Tivoli Directory Server as the LDAP server for integrated user management, users cannot be registered just by registering suffix DNs. You must add coding as shown in the example below at the beginning of the LDIF file used for user registration, and then execute the `ldapmodify` command.

Example: Adding `o=apsm.com` to the suffix DN

```
dn: o=apsm.com
objectclass: top
objectclass: organization
o: apsm.com
```

5.17.2 Registration by using the integrated user management framework library

Use the API provided by the user authentication library to create an application and register the user information by using that application.

The application that uses the API generates the LdapUserDataManager object first. Assign the LDAP directory server definition identifier to this class constructor. This definition identifier is specified in `ua.conf` (the integrated user management configuration file). It is associated with the LDAP directory server URL, bind DN, bind DN password, and base DN. Generate one LdapUserDataManager object per definition.

To learn more about how to register users, update the user information and change the repository information passwords when Active Directory is used as the LDAP directory server, see *5.17.4 Settings when using Active Directory*.

To learn more about implementing user authentication, see *5.10 Implementation of API-based user authentication*. For details about the API used in the integrated user management framework, *15. APIs Used with the Integrated User Management Framework*.

5.17.3 Formatting used to register the user information

The user information must conform to the formatting specified in the following table.

Table 5-15: Formatting of user information

Type of information	Meaning	Formatting
User ID	User identifier	A string of alphanumeric characters; the length is 1 to 512 characters.
Password	User-specific password	A string of alphanumeric and special characters; the length is 0 to 512 characters.

Note 1: A string of alphanumeric characters means a sequence of alphabetical characters (A to Z and a to z) and numbers (0 to 9).

Note 2: Special characters indicate the following symbols.

(white space) ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ (underscore) ` { } | (vertical bar) ~

Note 3: Unless otherwise stated, characters are case-sensitive.

Note 4: Use ASCII characters. (The program does not check grammar.)

Note 5: When passwords are stored in the repository in plain text, do not use null characters (`" "`) for passwords. The users who use null character passwords may not be able to log in. If null characters are to be used as passwords, encrypt them by using

SHA-1, etc.

5.17.4 Settings when using Active Directory

This section describes the settings when Active Directory is used as the LDAP directory server.

When the integrated user management framework library is used to register users to the user information repository and update the user information (including user passwords), it is necessary to set the Active Directory environment and register the certificate that enables connections over SSL.

The following table shows the settings when Active Directory is used as the LDAP directory server. The settings depend on the user authentication method being used.

Table 5-16: List of settings when using Active Directory

Setting		Password authentication	Change password and add/change user	Client authentication (X509 certificate)
`jaas.conf`	Specify the necessary login module/	R	R	R
	Specify the login module option/	O	R	--
`ua.conf`	Specify the user identifying attribute/	R	R	R
	Specify the attribute name used as the user ID in the DN/	--	--	R
	Specify the password attribute/	R	R	--
	Specify the type of LDAP directory server/	R	R	--
	Convert the DN containing 2-byte characters such as Japanese/	O	O	O
	Change the URL protocol/	O	R	O
Connections over SSL		O	R	O

Legend:

R: Required

O: Optional

-: Not required (The settings are ignored.)

The methods used to set the Active Directory environment and register the certificate that enables connections over SSL are as follows.

The object class and attributes of the users managed in Active Directory are different from those managed in other LDAP directory servers. When the integrated user management framework library is used, use the `user` object class and specify the `cn`, `unicodePwd`, `sAMAccountName`, and `userAccountControl` attributes to create that `user`.

Assign the security account manager (SAM) account name to `sAMAccountName`. Generally, the account name is the same value as the user ID.

Assign the user account property flag to `userAccountControl`. To create the general user entry, assign `512`. Note that users cannot be created when the minimum password length is set to one character in the security policy of the server on which Active Directory is installed. Take either of the following measures so that user entries can be created.

Change the minimum password length to zero or more characters in the security policy and then assign `512` to `userAccountControl`.

Assign `544` to `userAccountControl` without changing the minimum password length in the security policy.

(1) Setting jaas.conf

The settings in `jaas.conf` are as follows.

(a) Designating the login module

Specify WebPasswordLDAPLoginModule when password authentication is used. To use client certificates, specify WebCertificateLoginModule.

(b) Specifying the login module option (when using password authentication)

Assign `ldap.w` to the `WebPasswordLDAPLoginModule` option. Or, assign `sso.ldap.w` when single sign-on is used.

(2) Setting ua.conf

The settings in `ua.conf` are as follows.

(a) Specifying the user identifying attribute

Assign `cn` (full user name) or `sAMAccountName` to the user identifying attribute.

The setup examples are as follows:

Example 1: `cn` is used as the user identifying attribute.

```
com.cosminexus.admin.auth.ldap.attr.userid.0=cn
```

Example 2: `sAMAccountName` is used as the user identifying attribute.

```
com.cosminexus.admin.auth.ldap.attr.userid.0=sAMAccountName
```

When the user identifying attribute is set to `sAMAccountName`, it is necessary to search repository user entries (RDN). The following is the setting example of properties needed for the search.

Example:

```
java.naming.security.principal.0=cn=Administrator,cn=Users,
dc=cosminexus,dc=com
java.naming.security.credentials.0=adminpassword
com.cosminexus.admin.auth.ldap.search.userrdn.0=true
com.cosminexus.admin.auth.ldap.search.scope.0=onelevel
```

When the user identifying attribute is set to `sAMAccountName`, it is not possible to use the `addUserData` (`String uid`, `UserData userData`) method of the LdapUserDataManager class. To add users, use the `addUserData` (`String uid`, `UserData userData`, `String name`, `String value`) method and assign `cn` to the attribute name argument (String name).

(b) Specifying the attribute name used as the user ID in the DN (when using client certificates)

When client certificates are used, disassemble the DNs stored in the client certificates (requesting user's distinguished names in the client certificates) and set the attribute names used as the user IDs. The setup example is as follows:

Example:

```
com.cosminexus.admin.auth.ldap.certificate.attr.userid.0=cn
```

The above attribute name is different from the user identifying attribute specified in *(a) Specifying the user identifying attribute*.

(c) Specifying the password attribute (when using password authentication)

Assign `unicodePwd` to the password attribute. The setup example is as follows:

Example:

```
com.cosminexus.admin.auth.ldap.attr.password.0=unicodePwd
```

(d) Specifying the type of LDAP directory server (when using password authentication)

Add a line that specifies `AD` as the type of destination LDAP directory server. The following is an example of when the LDAP number is `0`.

Example:

```
com.cosminexus.admin.auth.ldap.directory.kind.0=AD
```

(e) Converting the DN containing two-byte characters such as Japanese characters

When the DN containing two-byte characters such as Japanese characters is set to the following `ua.conf` parameters, use the `native2ascii` command to convert it to Unicode.

- java.naming.security.principal
- com.cosminexus.admin.auth.ldap.basedn

The following is the executable form of the `native2ascii` command.

Executable form

```
Cosminexus installation directory\jdk\bin\native2ascii  pre-converted ua.conf file  converted ua.conf file
```

Use the converted `ua.conf` file as the integrated user management configuration file.

An example of code conversion is as follows.

Example:

The following are the procedures used to convert `a0.conf`, the `ua.conf` file containing the DN setting.

```
java.naming.security.principal.0=cn=Administrator,cn=管理者,dc=cosminexus,dc=co
com.cosminexus.admin.auth.ldap.basedn.0=cn=ユーザ,dc=cosminexus,dc=com
```

Execute the following `native2ascii` command against `ua0.conf`.
`native2ascii ua0.conf ua.conf`

(f) Changing the URL protocol

Specify `ldaps` as the URL protocol of the destination LDAP directory server. The setup example is as follows. Note that the port number can be omitted.

Example:

Before: `java.naming.provider.url.0=ldap://localhost:389`

After: `java.naming.provider.url.0=ldaps://localhost:636`

(3) Setting connections over SSL

Register the certificate to establish SSL communication between the J2EE server and Active Directory. The following describes how to register the certificate.

1. Create and register the digital certificate to the server on which Active Directory

is installed (LDAP server).

For details about how to create and register digital certificates, see the Active Directory documentation.

2. Register the certificate authority (CA) certificate to the J2EE server.

   The CA certificate can be registered to the J2EE server by using keytool, which comes with Cosminexus Developer's Kit for Java. For details about keytool, see the Java 2 SDK Standard Edition documentation. The following is an example of executing keytool. Although the example is composed of multiple lines, the actual statement is in one line.

   Windows:
   ```
   keytool -import -alias cakey -file C:\temp\cacer.cer
   -trustcacerts -keystore
   "Cosminexus installation directory\jdk\jre\lib\security\cacerts"
   ```

   UNIX:
   ```
   /opt/Cosminexus/jdk/bin/keytool -import -alias cakey -file /
   tmp/cacer.cer
   -trustcacerts -keystore /opt/Cosminexus/jdk/jre/lib/
   security/cacerts
   ```

If the J2EE server is running when registering the certificate by using keytool, restart the J2EE server.

(4) Notes

Please note:

- When `unicodePwd` is specified as the attribute value representing the user password in `ua.conf`, the `password.encrypt` and `password.encrypt.ex` password format option settings are disabled.
- Active Directory does not support changes to the existing user entry object class. When `com.cosminexus.admin.auth.ldap.directory.kind.0=AD` is specified in `ua.conf`, the object class at the time of creating the user entry will be applied even if the existing user entry is updated after a new object is added in the LdapUserDataManager class constructor.

5.18 Creation of encryption key files (When using single sign-on)

This section describes creating and changing encryption key files used to encrypt or decrypt the single sign-on user information. When the user information is not encrypted, it is not necessary to create encryption key files.

Use the encryption key file to encrypt the single sign-on user information, and store the encrypted information in the LDAP directory server. The encryption key file is also used to decrypt and reference the user information. To encrypt the user information, create the encryption key file before registering it in the LDAP directory server.

5.18.1 Creating encryption key files

Use the `ssogenkey` command to create the encryption key file used to encrypt or decrypt the single sign-on user information. The system administrators are responsible for storing the encryption key files in a safe place.

In addition, specify `JCE` as the encryption engine to encrypt or decrypt the single sign-on user information in `ua.conf` (the integrated user management configuration file).

For details about the `ssogenkey` command, see *ssogenkey (Creating encryption key files)*. For details about `ua.conf`, see *14.3 ua.conf (integrated user management configuration file)*.

5.18.2 Changing encryption key files

When the single sign-on user information has already been registered, perform the following procedures to change the encryption key file.

1. Execute the `ssoexport` command to fetch all the contents in the single sign-on information repository.
2. Run `ssogenkey` to create the encryption key file.
3. Execute the `ssoimport` command to register the contents fetched in Step 1.

For details about the `ssoexport` command, see *ssoexport (Referencing the single sign-on information repository)*. For details about the `ssogenkey` command, see *ssogenkey (Creating encryption key files)*. For details about the `ssoimport` command, see *ssoimport (Registering the single sign-on information repository)*.

5.19 Registration of user information (When using single sign-on)

This section describes how to register the single sign-on user information in the LDAP directory server and the formatting used to register the user information.

Cosminexus specifies the standard DIT structure of the single sign-on user management repository stored in the LDAP directory server. To learn more about the repository structure, see *5.2.4 Management method of user information used for integrated user management*.

There are the following two methods used to register the single sign-on user information in the single sign-on user information repository:

- Registration by using commands
- Registration by using the integrated user management framework library

The following subsections explain these methods respectively as well as the formatting used to register the user information.

5.19.1 Registration by using commands

Create the single sign-on user information in a CSV format file, and use the `ssoimport` command provided by Cosminexus to register it in the single sign-on information repository.

For details about the `ssoimport` command, see *ssoimport (Registering the single sign-on information repository)*. For details about CSV format files, see *14.4 CSV files containing single sign-on authentication information*.

5.19.2 Registration by using the integrated user management framework library

Use the API provided by the single sign-on library to create an application, and register the user information by using that application.

The application that uses the API generates the LdapSSODataManager object first. Assign the realm name to this class constructor. Generate one LdapSSODataManager object per definition.

Implementing the single sign-on authentication information listener class that notifies when the API updates the single sign-on information repository contents can update the destination system repository in synchronization with the single sign-on information repository update.

To learn more about implementing user authentication, see *5.10 Implementation of API-based user authentication*. For details about the API used in the integrated user management framework, see *15. APIs Used with the Integrated User Management Framework*.

5.19.3 Formatting used to register the user information

The single sign-on user information must conform to the formatting specified in the following table:

Table 5-17: Formatting of single sign-on user information

Type of information	Meaning	Grammar
Realm name	The identifier that indicates the scope of user management	A string of alphanumeric characters; not case sensitive; specify the name that can be used in the DN.
User ID	The identifier that indicates the user of the application with the user management function	A string of alphanumeric characters; the length is 1 to 512 characters.
SecretData	The authentication information that needs to be encrypted such as the user ID-specific password of the application with the user management function; specify the value used to authenticate the user. The value specified here is saved in the encrypted form.	A string of alphanumeric and special characters; the length is 0 to 512 characters.
PublicData	The authentication information that the application with the user management function requires authenticating the user excluding the user ID and SecretData; the value specified here is not encrypted.	A string of alphanumeric and special characters; the length is 0 to 512 characters.

Note 1: A string of alphanumeric characters means a sequence of alphabetical characters (A to Z and a to z) and numbers (0 to 9).

Note 2: Special characters indicate the following symbols.

(white space) ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ (underscore) ` { } | (vertical bar) ~

Note 3: Unless otherwise stated, characters are case-sensitive.

Note 4: Use ASCII characters. The program does not check grammar.

5.20 Creating configuration files

This section describes creating the following two configuration files and provides examples of configuration file settings.

- `jaas.conf` (the JAAS configuration file)
- `ua.conf` (the integrated user management configuration file)

5.20.1 Creating jaas.conf

`jaas.conf` stores the login module names used by each application, the repository number specified in `ua.conf` (the LDAP directory server or RDB), and other information used by the user authentication or single sign-on library.

(1) Location

The `jaas.conf` location is as follows.

- Windows:

  Cosminexus installation directory`\manager\config\jaas.conf`
- UNIX:

  `/opt/Cosminexus/manager/config/jaas.conf`

Overwrite this `jaas.conf` file or copy it to a new location. Specify the `jaas.conf` location in the Java VM properties at the time of startup. For details about setting the JavaVM properties at the time of startup, see *5.21 JavaVM property setup*.

Change the `jaas.conf` access permission so that the Component Container administrators can reference the file. To learn more about setting the Component Container administrators, see *4.1.4 Important points in setting the Component Container administrators (in UNIX)* in the *uCosminexus Application Server System Setup and Operation Guide*.

(2) Specification

Specify the following information per application.

When the user authentication library is used to authenticate users:

Define WebPasswordLoginModule as the login module. Assign the LDAP number and realm name defined by the repository defined in `ua.conf` to the WebPasswordLoginModule options.

To integrate WebPasswordLoginModule with custom login modules, define DelegationLoginModule as the login module, and assign the custom login module name to invoke custom login modules to the DelegationLoginModule option.

When the single sign-on library is used to authenticate users:

Define WebSSOLoginModule as the login module. Assign the custom login module identifier and realm name defined in `ua.conf` to the WebSSOLoginModule options.

To learn more about the `jaas.conf` settings when using Active Directory as the LDAP directory server, see *5.17.4 Settings when using Active Directory*.

For details about `jaas.conf`, see *14.2 jaas.conf (JAAS configuration file)*.

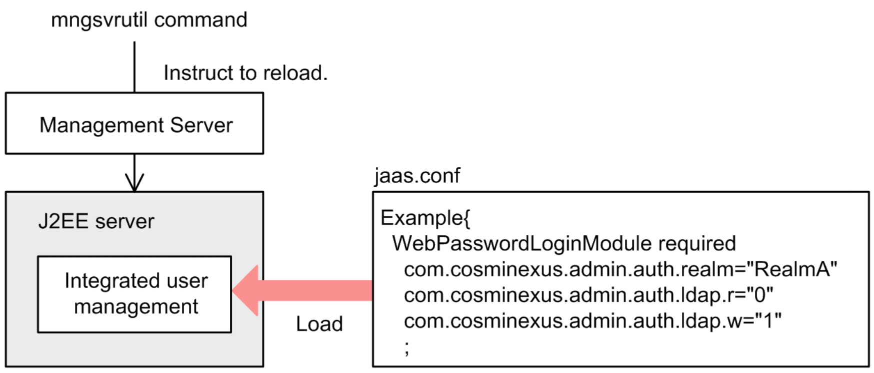
(3) Reloading jaas.conf

`jaas.conf` can be reloaded without restarting the J2EE server by using the `mngsvrutil` command. This Management Server management command can be used to change the LDAP number used by the login module without restarting the J2EE server.

To use the `mngsvrutil` command, it is required that Management Server has started and been configured appropriately.

The following figure shows the flow of reloading `jaas.conf`.

Figure 5-25: Flow of reloading jaas.conf



```
Example{
 WebPasswordLoginModule required
  com.cosminexus.admin.auth.realm="RealmA"
  com.cosminexus.admin.auth.ldap.r="0"
  com.cosminexus.admin.auth.ldap.w="1"
  ;
```

For details about the `mngsvrutil` command, see *mngsvrutil (Management Server Management Command)* in the *uCosminexus Application Server Command Reference Guide*.

5.20.2 Creating ua.conf

`ua.conf` stores the repository access information (the LDAP directory server or RDB), the path to the encryption key file used to encrypt or decrypt the single sign-on user information, and other information used by the user authentication or single sign-on library.

(1) Location

The `ua.conf` location is as follows.

- Windows:

  Cosminexus installation directory`\manager\config\ua.conf`
- UNIX:

  `/opt/Cosminexus/manager/config/ua.conf`

Overwrite this `ua.conf` file or copy it to a new location. Specify the `ua.conf` location in the Java VM properties at the time of startup. For details about setting the JavaVM properties at the time of startup, see *5.21 JavaVM property setup*.

Change the `ua.conf` access permission so that the Component Container administrators can reference the file. To learn more about setting the Component Container administrators, see *4.1.4 Important points in setting the Component Container administrators (in UNIX)* in the *uCosminexus Application Server System Setup and Operation Guide*.

(2) Specification

To enable user authentication by using the user authentication library and single sign-on by using the single sign-on library, configure the LDAP directory server URL, base DN, and access permissions.

When the single sign-on library function is used to implement single sign-on, select the encryption product and specify the encryption key file name. When the single sign-on function invokes a custom login module, specify the custom login module name and the directory that stores the class file associated with the custom login module.

To learn more about the `ua.conf` settings when using Active Directory as the LDAP directory server, see *5.17.4 Settings when using Active Directory*.

To learn more about `ua.conf`, see *14.3 ua.conf (integrated user management configuration file)*.

(3) Changing and scrambling passwords

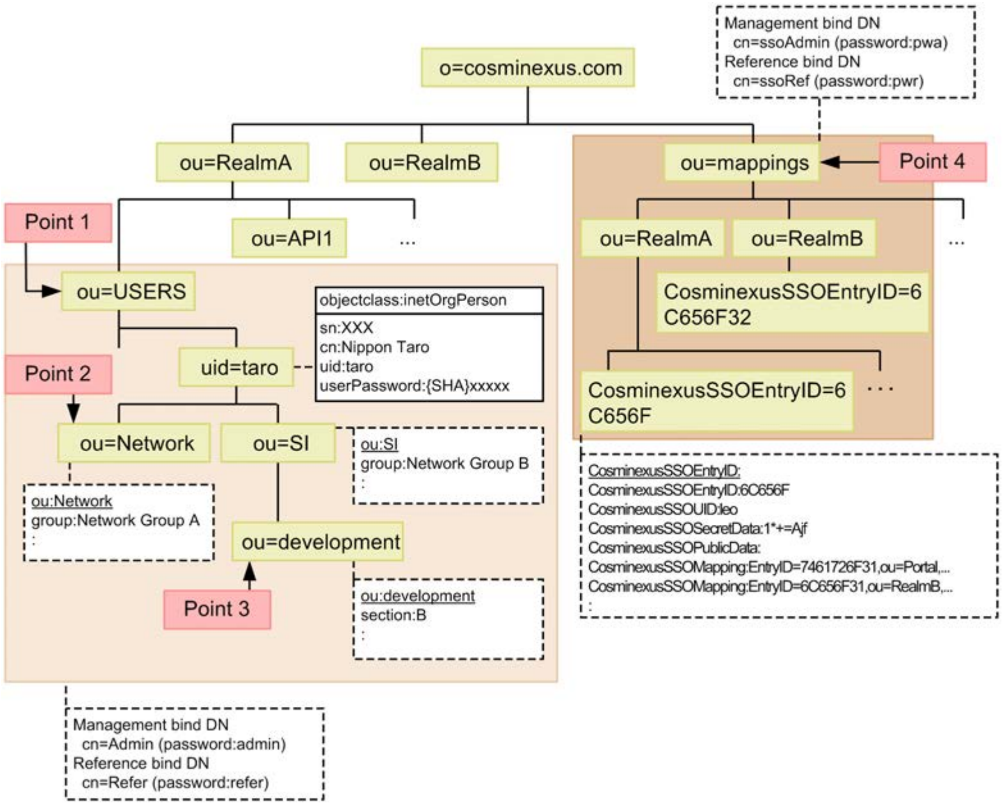
The `uachpw` command can be used not only to edit `ua.conf` but also to change the password specified in `ua.conf` used to access the LDAP directory server or RDB. Specifying the `-scramble` option when using the `uachpw` command to change the password used to access to the LDAP directory server or RDB can scramble the password.

When using the `uachpw` command to scramble the password, be sure to set the com.cosminexus.admin.auth.passwordScramble.enable key within the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file.

5.20.3 Example of setting the configuration file

This section gives an example of setting the user information that has the directory configuration as shown in the following figure.

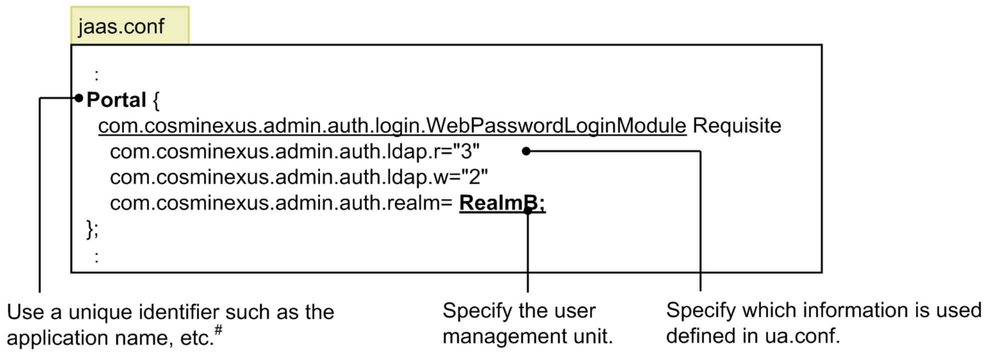
Figure 5-26: Example of user information directory configuration



(1) Example of the jaas.conf settings

`jaas.conf` stores the user authentication information. Specify the login module name and the repository number specified in `ua.conf` (the LDAP directory server or RDB) in `jaas.conf`. The following figure shows an example of the `jaas.conf` settings.

Figure 5-27: Example of the jaas.conf settings



```
jaas.conf
  :
Portal {
  com.cosminexus.admin.auth.login.WebPasswordLoginModule Requisite
    com.cosminexus.admin.auth.ldap.r="3"
    com.cosminexus.admin.auth.ldap.w="2"
    com.cosminexus.admin.auth.realm= RealmB;
};
  :
```

Use a unique identifier such as the application name, etc.#

Specify the user management unit.

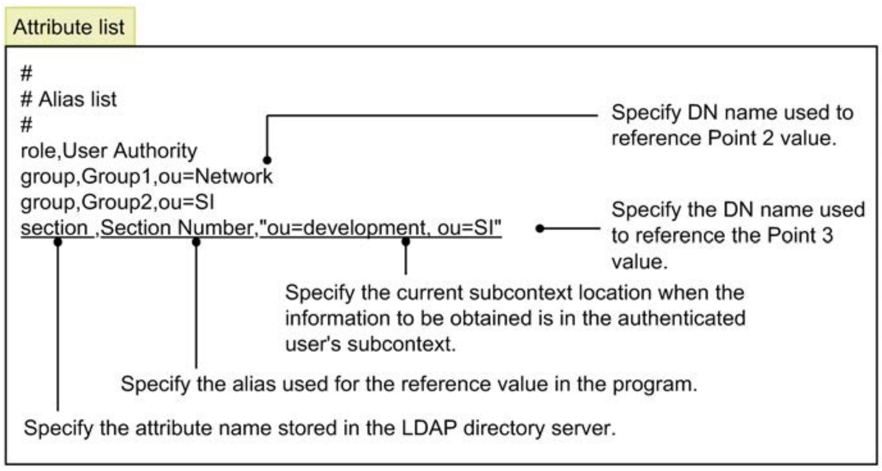
Specify which information is used defined in ua.conf.

Note#: The name specified here should be specified as the LoginContext constructor to be instantiated at the time of authentication.

(2) Example of the attribute list settings

The attribute list contains the user information that is obtained when the user is successfully authenticated (attribute name) and the alias is used to reference it. The following figure shows an example of the attribute list settings (c:\RealmA\config\AliasFile.csv (in Windows) or /tmp/RealmA/config/AliasFile.csv (in UNIX)).

Figure 5-28: Example of setting the attribute list settings



```
Attribute list
#
# Alias list
#
role,User Authority
group,Group1,ou=Network
group,Group2,ou=SI
section ,Section Number,"ou=development, ou=SI"
```

Specify DN name used to reference Point 2 value.

Specify the DN name used to reference the Point 3 value.

Specify the current subcontext location when the information to be obtained is in the authenticated user's subcontext.

Specify the alias used for the reference value in the program.

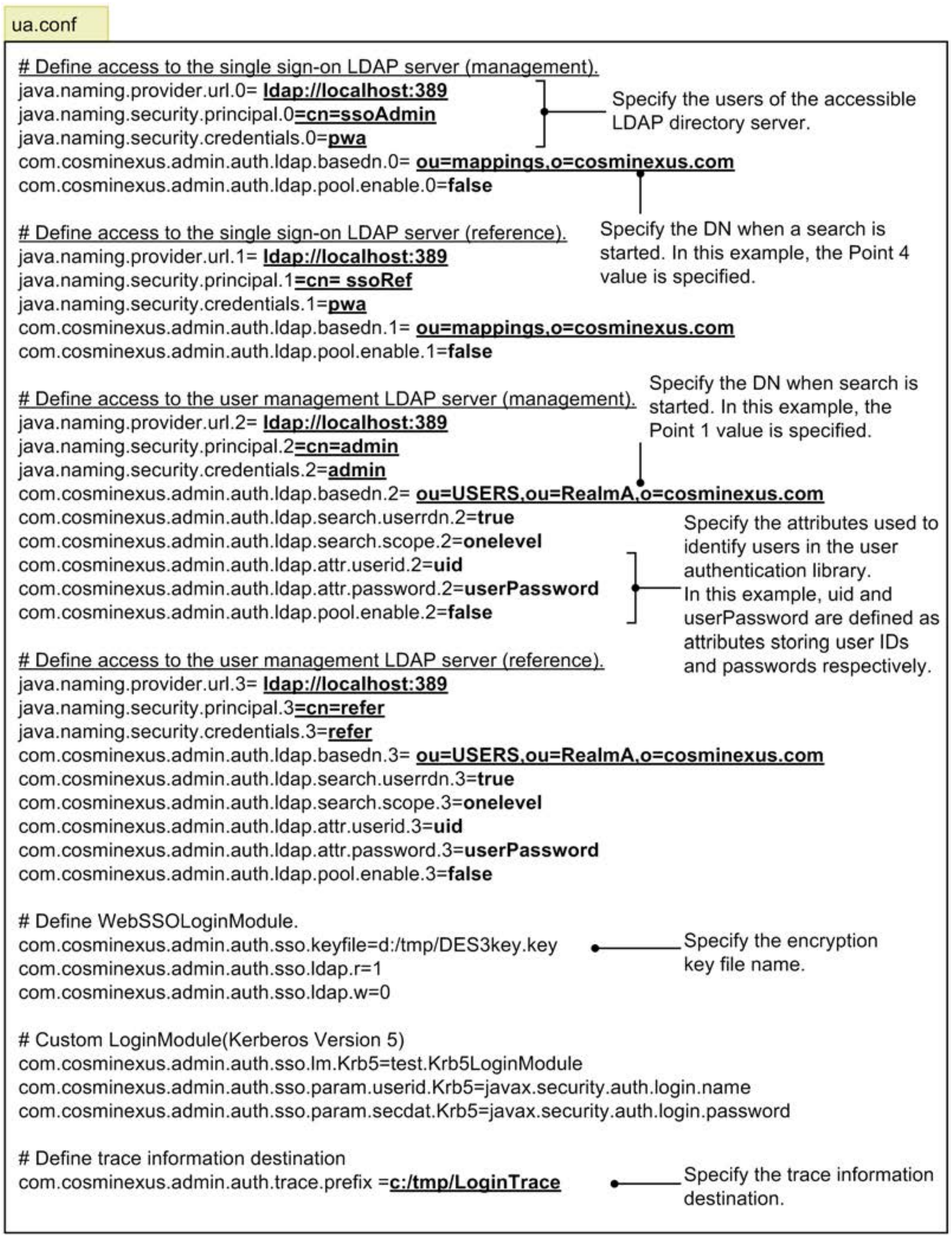
Specify the attribute name stored in the LDAP directory server.

Note that the attribute list can be directly specified within the program by using the AttributeEntry class without creating the file.

(3) Example of the ua.conf settings

`ua.conf` stores the repository access information (the LDAP directory server or RDB). The following figure shows an example of the `ua.conf` settings.

Figure 5-29: Example of the ua.conf settings (in Windows)



ua.conf

```
# Define access to the single sign-on LDAP server (management).
java.naming.provider.url.0= ldap://localhost:389
java.naming.security.principal.0=cn=ssoAdmin
java.naming.security.credentials.0=pwa
com.cosminexus.admin.auth.ldap.basedn.0= ou=mappings,o=cosminexus.com
com.cosminexus.admin.auth.ldap.pool.enable.0=false

# Define access to the single sign-on LDAP server (reference).
java.naming.provider.url.1= ldap://localhost:389
java.naming.security.principal.1=cn= ssoRef
java.naming.security.credentials.1=pwa
com.cosminexus.admin.auth.ldap.basedn.1= ou=mappings,o=cosminexus.com
com.cosminexus.admin.auth.ldap.pool.enable.1=false

# Define access to the user management LDAP server (management).
java.naming.provider.url.2= ldap://localhost:389
java.naming.security.principal.2=cn=admin
java.naming.security.credentials.2=admin
com.cosminexus.admin.auth.ldap.basedn.2= ou=USERS,ou=RealmA,o=cosminexus.com
com.cosminexus.admin.auth.ldap.search.userrdn.2=true
com.cosminexus.admin.auth.ldap.search.scope.2=onelevel
com.cosminexus.admin.auth.ldap.attr.userid.2=uid
com.cosminexus.admin.auth.ldap.attr.password.2=userPassword
com.cosminexus.admin.auth.ldap.pool.enable.2=false

# Define access to the user management LDAP server (reference).
java.naming.provider.url.3= ldap://localhost:389
java.naming.security.principal.3=cn=refer
java.naming.security.credentials.3=refer
com.cosminexus.admin.auth.ldap.basedn.3= ou=USERS,ou=RealmA,o=cosminexus.com
com.cosminexus.admin.auth.ldap.search.userrdn.3=true
com.cosminexus.admin.auth.ldap.search.scope.3=onelevel
com.cosminexus.admin.auth.ldap.attr.userid.3=uid
com.cosminexus.admin.auth.ldap.attr.password.3=userPassword
com.cosminexus.admin.auth.ldap.pool.enable.3=false

# Define WebSSOLoginModule.
com.cosminexus.admin.auth.sso.keyfile=d:/tmp/DES3key.key
com.cosminexus.admin.auth.sso.ldap.r=1
com.cosminexus.admin.auth.sso.ldap.w=0

# Custom LoginModule(Kerberos Version 5)
com.cosminexus.admin.auth.sso.lm.Krb5=test.Krb5LoginModule
com.cosminexus.admin.auth.sso.param.userid.Krb5=javax.security.auth.login.name
com.cosminexus.admin.auth.sso.param.secdat.Krb5=javax.security.auth.login.password

# Define trace information destination
com.cosminexus.admin.auth.trace.prefix =c:/tmp/LoginTrace
```

Specify the users of the accessible LDAP directory server.

Specify the DN when a search is started. In this example, the Point 4 value is specified.

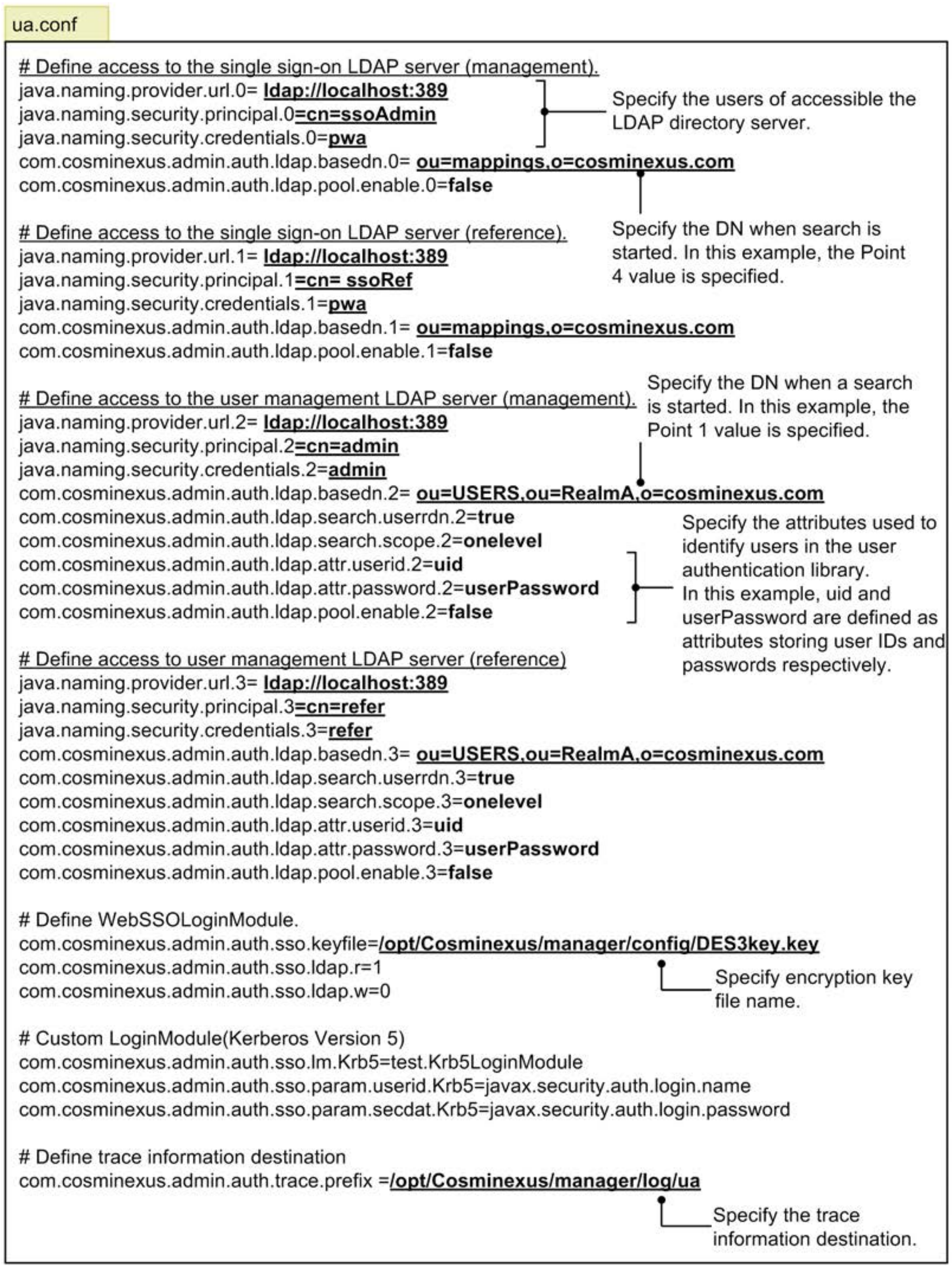
Specify the DN when search is started. In this example, the Point 1 value is specified.

Specify the attributes used to identify users in the user authentication library.
In this example, uid and userPassword are defined as attributes storing user IDs and passwords respectively.

Specify the encryption key file name.

Specify the trace information destination.

Figure 5-30: Example of the ua.conf settings (in UNIX)



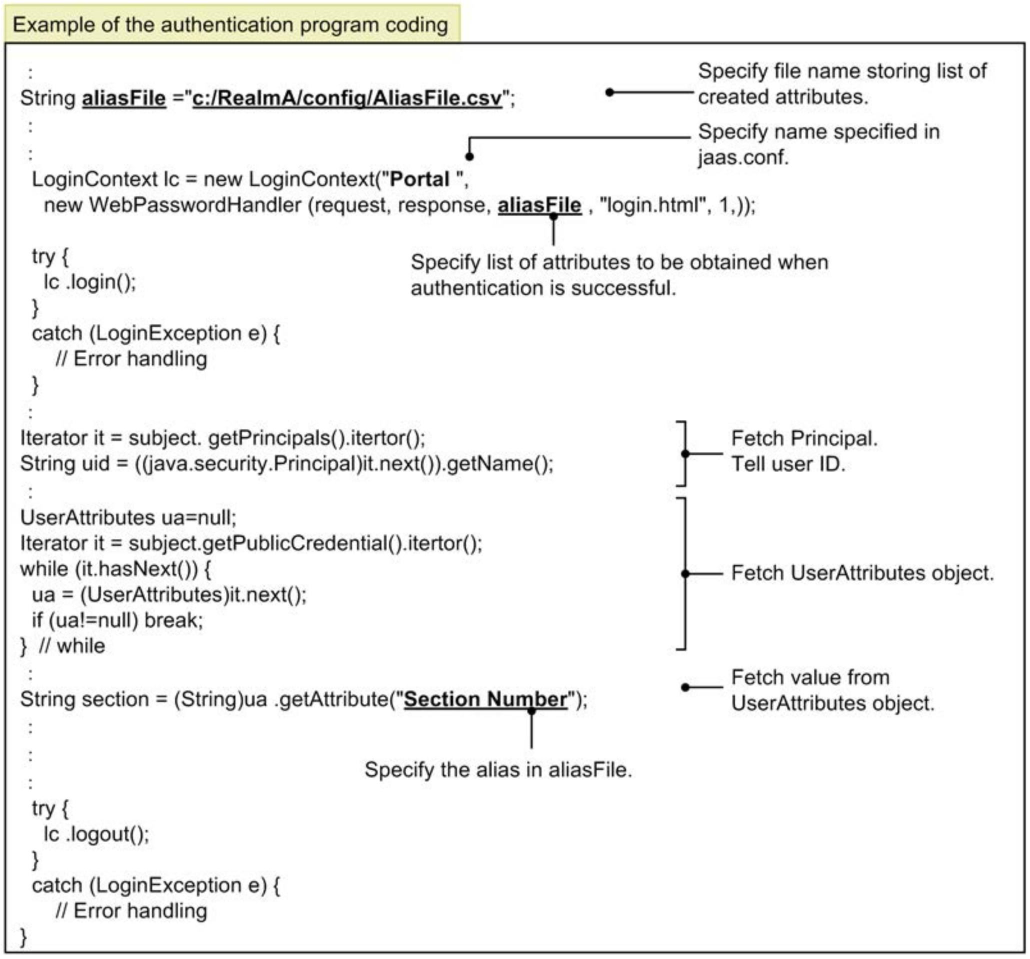
ua.conf

```
# Define access to the single sign-on LDAP server (management).
java.naming.provider.url.0= ldap://localhost:389
java.naming.security.principal.0=cn=ssoAdmin
java.naming.security.credentials.0=pwa
com.cosminexus.admin.auth.ldap.basedn.0= ou=mappings,o=cosminexus.com
com.cosminexus.admin.auth.ldap.pool.enable.0=false

# Define access to the single sign-on LDAP server (reference).
java.naming.provider.url.1= ldap://localhost:389
java.naming.security.principal.1=cn= ssoRef
java.naming.security.credentials.1=pwa
com.cosminexus.admin.auth.ldap.basedn.1= ou=mappings,o=cosminexus.com
com.cosminexus.admin.auth.ldap.pool.enable.1=false

# Define access to the user management LDAP server (management).
java.naming.provider.url.2= ldap://localhost:389
java.naming.security.principal.2=cn=admin
java.naming.security.credentials.2=admin
com.cosminexus.admin.auth.ldap.basedn.2= ou=USERS,ou=RealmA,o=cosminexus.com
com.cosminexus.admin.auth.ldap.search.userrdn.2=true
com.cosminexus.admin.auth.ldap.search.scope.2=onelevel
com.cosminexus.admin.auth.ldap.attr.userid.2=uid
com.cosminexus.admin.auth.ldap.attr.password.2=userPassword
com.cosminexus.admin.auth.ldap.pool.enable.2=false

# Define access to user management LDAP server (reference)
java.naming.provider.url.3= ldap://localhost:389
java.naming.security.principal.3=cn=refer
java.naming.security.credentials.3=refer
com.cosminexus.admin.auth.ldap.basedn.3= ou=USERS,ou=RealmA,o=cosminexus.com
com.cosminexus.admin.auth.ldap.search.userrdn.3=true
com.cosminexus.admin.auth.ldap.search.scope.3=onelevel
com.cosminexus.admin.auth.ldap.attr.userid.3=uid
com.cosminexus.admin.auth.ldap.attr.password.3=userPassword
com.cosminexus.admin.auth.ldap.pool.enable.3=false

# Define WebSSOLoginModule.
com.cosminexus.admin.auth.sso.keyfile=/opt/Cosminexus/manager/config/DES3key.key
com.cosminexus.admin.auth.sso.ldap.r=1
com.cosminexus.admin.auth.sso.ldap.w=0

# Custom LoginModule(Kerberos Version 5)
com.cosminexus.admin.auth.sso.lm.Krb5=test.Krb5LoginModule
com.cosminexus.admin.auth.sso.param.userid.Krb5=javax.security.auth.login.name
com.cosminexus.admin.auth.sso.param.secdat.Krb5=javax.security.auth.login.password

# Define trace information destination
com.cosminexus.admin.auth.trace.prefix =/opt/Cosminexus/manager/log/ua
```

Specify the users of accessible the LDAP directory server.
Specify the DN when search is started. In this example, the Point 4 value is specified.
Specify the DN when a search is started. In this example, the Point 1 value is specified.
Specify the attributes used to identify users in the user authentication library. In this example, uid and userPassword are defined as attributes storing user IDs and passwords respectively.
Specify encryption key file name.
Specify the trace information destination.

The following figure shows an example of the authentication program coding.

Figure 5-31: Example of the authentication program coding (in Windows)



Example of the authentication program coding

```
 :
String aliasFile ="c:/RealmA/config/AliasFile.csv";
 :
 :
  LoginContext lc = new LoginContext("Portal ",
    new WebPasswordHandler (request, response, aliasFile , "login.html", 1,));

  try {
    lc .login();
  }
  catch (LoginException e) {
      // Error handling
  }
 :
Iterator it = subject. getPrincipals().itertor();
String uid = ((java.security.Principal)it.next()).getName();
 :
UserAttributes ua=null;
Iterator it = subject.getPublicCredential().itertor();
while (it.hasNext()) {
  ua = (UserAttributes)it.next();
  if (ua!=null) break;
}  // while
 :
String section = (String)ua .getAttribute("Section Number");
 :
 :
 :
  try {
    lc .logout();
  }
  catch (LoginException e) {
      // Error handling
}
```

Specify file name storing list of created attributes.

Specify name specified in jaas.conf.

Specify list of attributes to be obtained when authentication is successful.

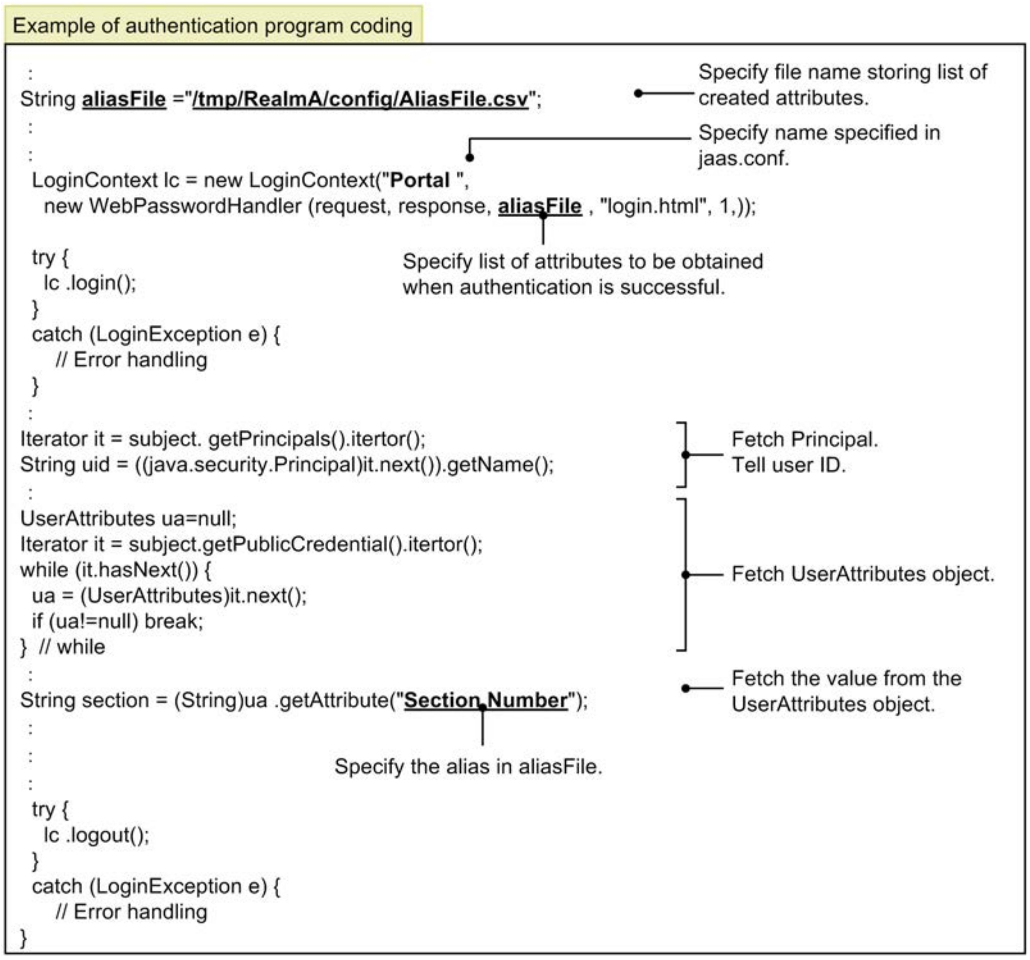
Fetch Principal.
Tell user ID.

Fetch UserAttributes object.

Fetch value from UserAttributes object.

Specify the alias in aliasFile.

Figure 5-32: Example of the authentication program coding (in UNIX)

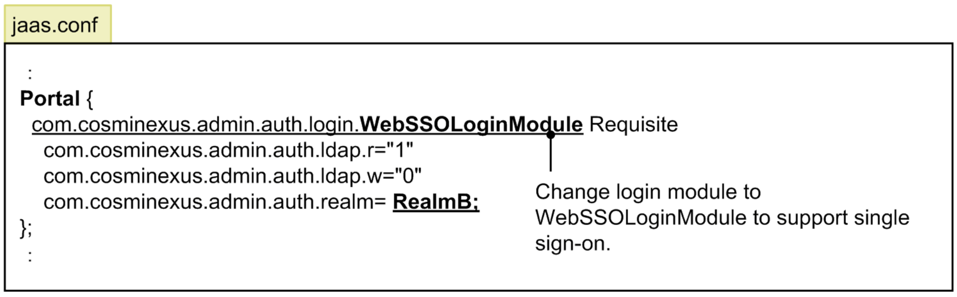

Example of authentication program coding
Specify file name storing list of created attributes.
Specify name specified in jaas.conf.
Specify list of attributes to be obtained when authentication is successful.
Fetch Principal. Tell user ID.
Fetch UserAttributes object.
Fetch the value from the UserAttributes object.
Specify the alias in aliasFile.

```
 :
String aliasFile ="/tmp/RealmA/config/AliasFile.csv";
 :
 :
 LoginContext lc = new LoginContext("Portal ",
   new WebPasswordHandler (request, response, aliasFile , "login.html", 1,));

 try {
   lc .login();
 }
 catch (LoginException e) {
    // Error handling
 }
 :
Iterator it = subject. getPrincipals().itertor();
String uid = ((java.security.Principal)it.next()).getName();
 :
UserAttributes ua=null;
Iterator it = subject.getPublicCredential().itertor();
while (it.hasNext()) {
  ua = (UserAttributes)it.next();
  if (ua!=null) break;
}  // while
 :
String section = (String)ua .getAttribute("Section Number");
 :
 :
 :
 try {
   lc .logout();
 }
 catch (LoginException e) {
    // Error handling
}
```

(4) Example of supporting single sign-on (when using the standard login module)

`jaas.conf` needs to be edited so that the user authentication library login module supports single sign-on. The following figure shows an example of supporting single sign-on (when using the standard login module).

Figure 5-33: Example of supporting single sign-on (when using the standard login module)



jaas.conf

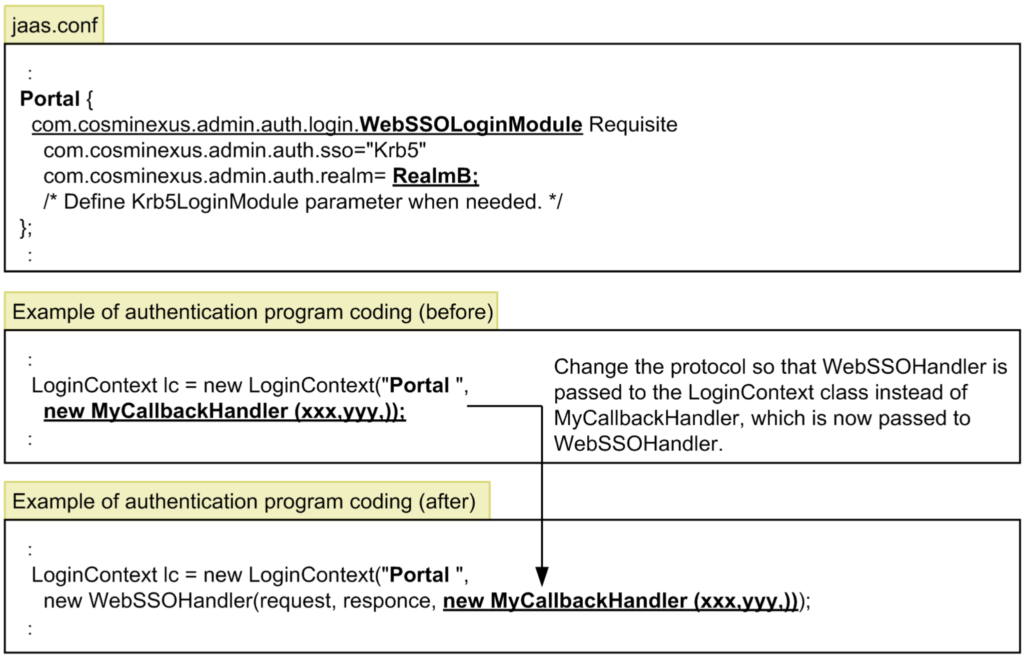
```
 :
Portal {
 com.cosminexus.admin.auth.login.WebSSOLoginModule Requisite
   com.cosminexus.admin.auth.ldap.r="1"
   com.cosminexus.admin.auth.ldap.w="0"
   com.cosminexus.admin.auth.realm= RealmB;
};
 :
```

Change login module to WebSSOLoginModule to support single sign-on.

(5) Example of supporting single sign-on (when using the custom login module)

`jaas.conf` and the authentication program coding must be edited so that the custom login module supports single sign-on. In addition, the custom login module definition item must be specified in `ua.conf`, which corresponds to the login module identifier ("Krb5" in the example). The following figure shows an example of supporting single sign-on (when using the custom login module).

Figure 5-34: Example of supporting single sign-on (when using the custom login module)



jaas.conf

```
 :
Portal {
 com.cosminexus.admin.auth.login.WebSSOLoginModule Requisite
   com.cosminexus.admin.auth.sso="Krb5"
   com.cosminexus.admin.auth.realm= RealmB;
   /* Define Krb5LoginModule parameter when needed. */
};
 :
```

Example of authentication program coding (before)

```
 :
LoginContext lc = new LoginContext("Portal ",
  new MyCallbackHandler (xxx,yyy,));
 :
```

Change the protocol so that WebSSOHandler is passed to the LoginContext class instead of MyCallbackHandler, which is now passed to WebSSOHandler.

Example of authentication program coding (after)

```
 :
LoginContext lc = new LoginContext("Portal ",
  new WebSSOHandler(request, responce, new MyCallbackHandler (xxx,yyy,)));
 :
```

5.21 JavaVM property setup

When the integrated user management is used, it is necessary to set the JavaVM properties when JavaVM starts. The Easy Setup definition file or the management portal should be used for this purpose. This section describes how to set the properties in the Easy Setup definition file when the SmartComposer function is used to establish the Web system.

Enter the JavaVM properties within the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file.

The following table lists the JavaVM properties.

Table 5-18: JavaVM properties in the easy setup definition file

Parameter	Setting
jaas.ua.enabled	Enables the JavaVM JAAS.
java.security.auth.login.config	Specifies the `jaas.conf` file path.
com.cosminexus.admin.auth.config	Specifies the `ua.conf` file path.
com.cosminexus.admin.auth.passwordScramble.enable	Enables or disables the function to encrypt passwords scrambled by the `uachpw` command; for details, see *5.20.2(3) Changing and scrambling passwords*.
jaas.config.load_exclusively	Specifies whether to ignore login configurations other than `jaas.conf` specified by the `java.security.auth.login.config` parameter.

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

The following is an example of JavaVM properties.

In the example, password scrambling is enabled and login configurations other than `jaas.conf` are ignored.

Example of the Easy Setup definition file

```
:
<configuration>
  <logical-server-type>j2ee-server</logical-server-type>
    <param>
      <param-name>jaas.ua.enabled</param-name>
      <param-value>true</param-value>
    </param>
    <param>
      <param-name>java.security.auth.login.config</
param-name>
      <param-value><Cosminexus installation directory>/
manager/config/jaas.conf</param-value>
    </param>
    <param>
      <param-name>com.cosminexus.admin.auth.config</
param-name>
      <param-value><Cosminexus installation directory>/
manager/config/ua.conf</param-value>
    </param>
    <param>

<param-name>com.cosminexus.admin.auth.passwordScramble.enab
le</param-name>
      <param-value>true</param-value>
    </param>
    <param>
      <param-name>jaas.config.load_exclusively</param-name>
      <param-value>true</param-value>
    </param>
:
</configuration>
```

Reference note:

To change the access permission so that the Web application can carry out operations on the LoginContext class, change the `server.policy` settings.

For details about `server.policy`, see *2.5 server.policy (security policy file for J2EE server)* in the *uCosminexus Application Server Definition Reference Guide*.

Reference note:

JavaVM property setup in the servlet engine mode

When the integrated user management is used, it is necessary to set the J2EE server `usrconf.properties` and `web.policy` when JavaVM starts. The following gives an overview of the file settings.

Settings in usrconf.properties

Use the java.security.auth.login.config key to specify the `jaas.conf` location.

Use the com.cosminexus.admin.auth.config key to specify the `ua.conf` location.

Use the com.cosminexus.admin.auth.passwordScramble.enable key to enable or disable the function to encrypt passwords scrambled by the `uachpw` command.

When passwords are scrambled by the `uachpw` command, be sure to include com.cosminexus.admin.auth.passwordScramble.enable=true in usrconf.properties to enable the function to encrypt scrambled passwords.

The usrconf.properties location is as follows.

Windows:

Cosminexus installation directory`\CC\web\containers\`*server name*`\usrconf\usrconf.properties`

UNIX:

`/opt/Cosminexus/CC/web/containers/`*server name*`/usrconf/usrconf.properties`

Example of usrconf.properties settings

Windows:

```
java.security.auth.login.config==C:/Program Files/
Hitachi/Cosminexus/manager/config/jaas.conf
```

```
com.cosminexus.admin.auth.config=C:/Program Files/
Hitachi/Cosminexus/manager/config/ua.conf
```

```
com.cosminexus.admin.auth.passwordScramble.enable=tru
e
```

"`C:/Program Files/Hitachi/Cosminexus/`" is the default Cosminexus installation directory. Replace it with the actual installation directory when needed.

UNIX:

```
java.security.auth.login.config==/opt/Cosminexus/
manager/config/jaas.conf
```

```
com.cosminexus.admin.auth.config=/opt/Cosminexus/
manager/config/ua.conf
```

```
com.cosminexus.admin.auth.passwordScramble.enable=tru
e
```

Settings in web.policy

`web.policy` sets the access permission so that the Web application can carry out operations on the LoginContext class. For details about access permission settings, see the javax.security.auth.AuthPermission class.

To provide enhanced support of custom login module and authentication password encryption, it is required to set access permissions to carry out operations on each of the classes.

The `web.policy` location is as follows.

Windows:

Cosminexus installation directory`\CC\web\containers\`*server name*`\usrconf\web.policy`

UNIX:

`/opt/Cosminexus/CC/web/containers/`*server name*`/usrconf/web.policy`

5.22 Deployment of files

This section describes how to deploy the EAR file used in integrated user management. Import `uastartup.ear` into the J2EE server used in integrated user management, and start it up. `uastartup.ear` is located in the following directory:

- Windows:

  Cosminexus installation directory`\manager\config`

- UNIX:

  `/opt/Cosminexus/manager/config`

Register `uastartup.ear` in Management Server and then import it into the J2EE server. For details about importing J2EE applications, see *13.5 Deployment and undeployment of J2EE applications* in the *uCosminexus Application Server Common Container Functionality Guide*.

When the integrated user management framework function is used, make sure to run `uastartup.ear`. Otherwise, the Management Server resource watch functionality and the `reload userAdmin` subcommand of the `mngsvrutil` command are unavailable.

Chapter

6. Authentication by Application Setup

This chapter describes authentication methods that involve the use of Web applications offered by EJB and Web containers.

6.1 Organization of this chapter

This chapter describes authentication methods that involve the use of Web applications offered by EJB and Web containers.

The table below shows the organization of this chapter.

Table 6-1: Organization of this chapter (Authentication by Application Setup)

Title	Relevant information
Web container-based authentication using DD settings	6.2
Authentication with security identities	6.3

6.2 Web container-based authentication using DD settings

Role-based authentication processes are handled by a Web container. Each user is given one or more roles for user management. The roles are configured using the `<security-constraint>` tag within the DD file (`WEB-INF/web.xml`) included in a J2EE application. For details about how to configure the J2EE application, see *6.2.2 Definitions in DD files*.

Using a Web application, you can define the roles necessary for access to each particular URL within the context. When a Web client requests access to a restricted URL, the authentication process involves two steps:

- Determining whether the access request to be authenticated is from a valid user
- Determining whether the roles given to the user match those required for access

Only a user who is recognized as valid in both steps can access the restricted URL.

The table below shows the organization of this section.

Table 6-2: Organization of this section (Web container-based authentication using DD settings)

Part	Title	Relevant information
Description	Web container-based authentication functionality using DD settings	6.2.1
Implementation	Definitions in DD files	6.2.2
Setup	Setup in an execution environment (J2EE application setup)	6.2.3
Precautions	Precautions for using authentication functionalities	6.2.4

Note: This section does not include information on operation.

6.2.1 Web container-based authentication functionality using DD settings

The following subsection describes the functionality that can be implemented for Web container-based authentication using the DD file (`WEB-INF/web.xml`) settings.

(1) User information management

The Web container defines, holds, and maintains the user names, passwords, and roles of users according to the user management functionality of the J2EE server.

(2) Container security and access permission management

The Web container can restrict Web clients from accessing particular URLs.

To do this, it is necessary to define the following in the DD file (`WEB-INF/web.xml`):

- URL pattern for access restriction
- Security definitions necessary for access, such as role definitions
- Authentication method for obtaining roles defined for users

If the Web client fails to authenticate or if the user does not have the role necessary for access, any attempt to access a URL pattern for which access is restricted results in an error. Note that once authenticated, a client is not authenticated again during the valid session period.

The security definitions and authentication methods should be defined according to the DD file (`WEB-INF/web.xml`) specifications stipulated by Servlet API 2.3.

The Web container offers two types of authentication method: basic and form-based authentication. Both types of authentication can be defined by adding the `<login-config>` tag to the DD file (`WEB-INF/web.xml`) included in the J2EE application. For details about how to configure the J2EE application, see *6.2.2 Definitions in DD files*.

Note:

When using HTTP Server or Microsoft IIS with the Web server

When using either Cosminexus HTTP Server or Microsoft IIS with the Web server, the Web server authentication functionality must be canceled in order to correctly provide basic authentication from the Web container.

For details, see *6.2.4 Precautions for using authentication functionalities*.

Configuring Basic authentication in web.xml

Specify a realm name in the `<realm-name>` tag. If no realm name is specified, `Authentication required` is used as the realm name. Specifying a null character or only a space in the `<realm-name>` tag is treated in the same manner as when the `<realm-name>` tag is omitted.

(3) Program security

If access to a servlet or JSP is restricted by basic or form-based authentication configured using the DD file (`WEB-INF/web.xml`), that servlet or JSP can perform fine-grained security processes at the program level by using the `HttpServletRequest` APIs listed below. These processes include changing the operation according to the user name and role name for the logged-in user.

- `getRemoteUser()`
- `isUserInRole()`
- `getUserPrincipal()`

For details about these APIs, see *Java Servlet Specification* v2.3.

6.2.2 Definitions in DD files

In `web.xml`, specify the settings for Web container-based authentication. The table below shows the definitions used in DD files.

Table 6-3: Definitions used in DD files for Web container-based authentication

Specified tag	Setting
`<security-constraint>` tag	Specifies the security restrictions.
`<login-config>-<auth-method>` tag	Specifies the method for Web container-based authentication.

You can use `web.xml` to configure the settings for a Web application before deploying the application on the J2EE server. You can use an attributes file to configure the same settings for a Web application already deployed on the J2EE server. For details about the settings in attributes files, see *6.2.3 Setup in an execution environment (J2EE application setup)*.

6.2.3 Setup in an execution environment (J2EE application setup)

To configure Web container-based authentication in an execution environment, use the attributes file and server management commands. To configure Web container-based authentication using DD settings, use the WAR attributes file. The table below shows the definitions used in the WAR attributes file.

Table 6-4: Web container-based authentication using DD settings (WAR attributes file)

Specified tag	Setting
`<security-constraint>` tag	Specifies the security restrictions.
`<login-config>-<auth-method>` tag	Specifies the method for Web container-based authentication.

You can use the WAR attributes file to configure the settings for a Web application already deployed on the J2EE server. You can use `web.xml` to configure the same settings for a Web application before deploying the application on the J2EE server. For details about the settings in `web.xml`, see *6.2.2 Definitions in DD files*.

6.2.4 Precautions for using authentication functionalities

The following subsection describes the precautions for using the authentication functionalities of both Web containers and Web servers.

(1) Order of authentication

When the authentication functionalities of both a Web container and a Web server are used in combination, they are performed in the following order:

1. Authentication functionality of the Web server
2. Authentication functionality of the Web container

The authentication functionality of a Web server includes one or more authentication processes. These processes are: Web server-based basic authentication, server authentication with SSL, and client authentication with SSL. When Web container and Web server authentication functionalities are both used, one or more of these processes is used together with the authentication functionality of the Web container.

For details about the authentication functionality and access control functionality of Cosminexus HTTP Server, see the *HTTP Server User Guide*.

(2) Precautions for using both Web server-based and Web container-based basic authentication

When you use both Web server-based and Web container-based basic authentication, the user name and password authenticated by the Web server are passed to the Web container. Thus, it is necessary to define common user information for the Web server and Web container.

Note that after Web server-based authentication, the Web container operation varies depending on how the Web container performs authentication. The Web container operates as follows:

- When the user is authenticated by the Web server but not by the Web container:

  The Web container displays a dialog box for entering a user name and password. This dialog box requires the common user name and password for the Web server and Web container.

- When a user authenticated by the Web server does not have a role that allows access to the Web container:

  An attempt to access a URL pattern for which access is restricted results in an error.

- When a user authenticated by the Web server has a role that allows access to the Web container:

  The dialog box for entering a user name and password does not appear. The user can access the URL pattern for which access is restricted.

(3) Precautions for using Microsoft IIS with a Web server

When using Web container-based authentication, you cannot use the following authentication functionality of Microsoft IIS:

- Digest authentication

  You cannot use digest authentication regardless of whether you use the Web container's authentication functionality. Make sure you cancel the digest authentication settings in Microsoft IIS.

- Integrated Windows authentication

  You cannot use integrated Windows authentication when using Web container-based basic authentication. Make sure you cancel the integrated Windows authentication settings in Microsoft IIS.

6.3 Authentication with security identities

This section describes authentication methods that use security identities.

The table below shows the organization of this section.

Table 6-5: Organization of this section (Authentication with security identities)

Part	Title	Relevant information
Description	Security identity functionality	6.3.1
Implementation	Security implementation in EJB client applications	6.3.2
Setup	Authentication setup with security identities	6.3.3

Note: This section does not include information on operation or precautions.

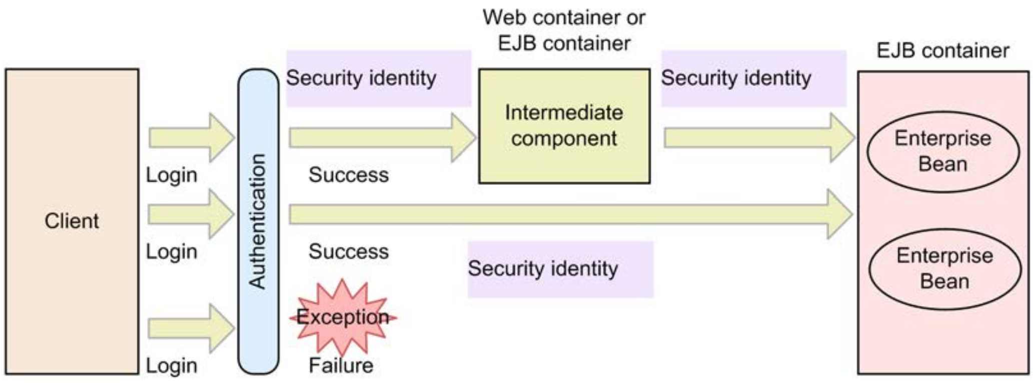
6.3.1 Security identity functionality

By using the security management functionality, you can authenticate users who want to access the Web container or EJB container. Such users are authenticated using their user names and passwords.

On successful authentication when using the security management functionality, authentication information called a *security identity* is created and sent to the Web container or EJB container. If this authentication fails, an exception occurs.

The figure below shows the flow when using a security identity for authentication.

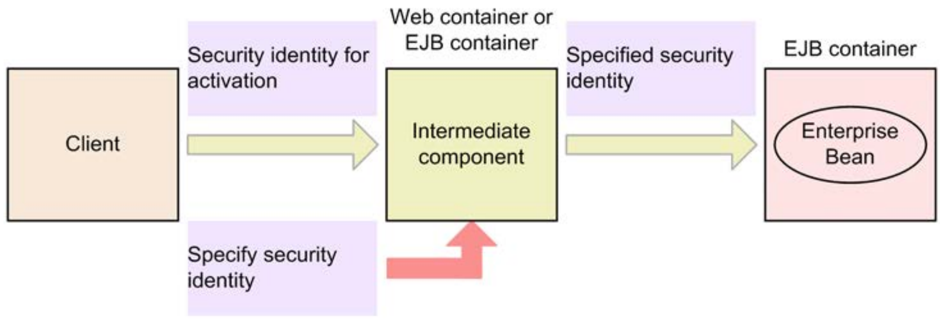
Figure 6-1: Flow when using a security identity for authentication



For security management, you can use the Run As functionality to send the security identity specified with the intermediate component.

If a security identity that differs from the one used to log into the client is specified with the intermediate component that calls the Enterprise Bean, the Run As functionality allows the specified security identity to be used to call the Enterprise Bean. The figure below shows the Run As functionality.

Figure 6-2: Run As functionality


Client
Security identity for activation
Specify security identity
Web container or EJB container
Intermediate component
Specified security identity
EJB container
Enterprise Bean

Note that the application server does not support the following types of security management functionality:

- Encryption of messages for accessing components
- Assignment of signatures to messages
- Authentication with certificates

6.3.2 Security implementation in EJB client applications

EJB client applications can authenticate users by using their user names and passwords as defined in the J2EE server. After a user authenticated by the EJB client application logs in, he or she can call the Enterprise Bean method for which his or her security role is configured.

(1) Implementation procedure

Cosminexus offers APIs to implement security in EJB client applications. The prerequisites and procedure for implementing this security are shown below. For details about the functionality and syntax of the APIs, see *4. APIs Available for EJB Client Applications*, in the *uCosminexus Application Server API Reference Guide*.

Before implementing security, make sure that the following prerequisites are satisfied:

- The user should be registered on the J2EE server.
- The registered user should be assigned a security role.

To implement security in an EJB client application, follow these steps:

1. Import a security API package.

To use the security APIs, import the package shown below.

```
import com.hitachi.software.ejb.security.base.authentication.*
```

2. Obtain the LoginInfoManager object.

   Use a program that calls Enterprise Bean methods to obtain the `LoginInfoManager` object. To obtain the object, use the `getLoginInfoManager` method, which is a static method for the `LogInfoManager` object.

```
LoginInfoManager  lm = LoginInfoManager.getLoginInfoManager();
```

3. Log in with the user name and password.

   After obtaining the `LoginInfoManager` object, call the login method.

```
lm.login(username, password);
```

4. Call the Enterprise Bean method.

   After the `login` method succeeds, call the Enterprise Bean method.

5. Log out.

   After calling the Enterprise Bean method, log out from the J2EE server by using the `logout` method.

```
lm.logout();
```

Note:

To implement security in an EJB client application, you need to add `HiEJBClientStatic.jar` to the class path and compile the file.

(2) Sample program

Below is a sample program for calling the `getAccountID` method, where the Enterprise Bean is named `account`.

```
import com.hitachi.software.ejb.security.base.authentication.*;
    :
  try {
    LoginInfoManager lm = LoginInfoManager.getLoginInfoManager();
    String userName = System.getProperty("username");
    String password = System.getProperty("password");
    if(lm.login(userName , password)) {
      try {
        System.out.println("user:" + userName + "login success");
        Context ctx = new InitialContext();
        java.lang.Object obj = ctx.lookup(appUnitPath + "Account");
        AccountHome aHome =
           (AccountHome)PortableRemoteObject.narrow(obj,AccountHome.class);
        Account account  = aHome.create();
        account.getAccountID();
      } finally {
        lm.logout();
      }
    }
  } catch(NotFoundServerException e) {
    System.out.println("not found server");
  } catch(InvalidUserNameException e) {
    System.out.println("invalid user name");
  } catch(InvalidPasswordException e) {
    System.out.println("invalid password");
  } catch(Exception e) {
    e.printStackTrace();
  }
```

6.3.3 Authentication setup with security identities

To manage security by using security identities, user and role information must be registered using a server management command. The table below shows how to set up J2EE applications for authentication with security identities and for utilization of the Run As functionality.

Table 6-6: J2EE application setup for authentication with security identities and for utilization of the Run As functionality

Functionality	Item	Target to be set	Setting
Authentication with security identities	Specifying whether to perform authentication that uses security identities	Session bean, entity bean, or message-driven bean	Using the `<security-identity>` tag in the session bean, entity bean, or message-driven bean attributes file, specify whether to perform authentication.
Run As functionality	Specifying whether to use Run As functionality		Using the `<run-as>` tag in the session bean, entity bean, or message-driven bean attributes file, specify whether to use the Run As functionality.

Functionality	Item	Target to be set	Setting
	Specifying the security role name for Run As functionality		Using the `<role-name>` tag in the session bean, entity bean, or message-driven bean attributes file, specify the security role name.
	Specifying the principal name for Run As functionality		Using the `<user-name>` tag in the session bean, entity bean, or message-driven bean attributes file, specify the principal name.

To set up security identities, see *9.5 Security definitions (security identities)*.

Chapter

7. SSL/TLS Encryption of Authentication Information and Data

This chapter describes SSL/TLS encryption of communications between the Web server and Web clients. It also describes SSL/TLS authentication.

7.1 Organization of this chapter

This chapter describes data encryption and authentication using SSL/TLS. It also describes the setup method.

The table below shows the organization of this chapter.

Table 7-1: Organization of this chapter (SSL/TLS Encryption of Authentication Information and Data)

Functionality	Relevant information
SSL encryption of authentication information and data	7.2
Using TLSv1.2 for SSL/TLS communication	7.3

7.2 SSL encryption of authentication information and data

SSL encryption of communications between the Web server and Web clients is effective when used with a Web server that supports SSL.

The table below shows the organization of this section.

Table 7-2: Organization of this section (SSL encryption of authentication information and data)

Part	Title	Relevant information
Description	The authentication functionality of the Web server	7.2.1
	Selecting a communication path security level	7.2.2
	Acquiring an SSL certificate	7.2.3
Implementation	Definitions in DD files	7.2.4
Setup	SSL setup with Cosminexus HTTP Server	7.2.5
	Microsoft IIS setup (in Web redirector environments)	7.2.6
	Setup in an execution environment	7.2.7

Note: This section does not include information on operation or precautions.

7.2.1 The authentication functionality of the Web server

The Web server provides server authentication and client authentication functionality.

Server authentication

Server authentication involves encrypting random number data and sending it from the browser to the server, which uses a key exchange certificate to decrypt the data.

Only a server with the correct key exchange certificate knows the secret key for decryption. Thus, a handshake is established only if the server is recognized as legal by the client. In this case, the server does not assign an electronic signature; however, it is possible to reconfirm whether the server is legal after handshake establishment.

Client authentication

Client authentication involves the server sending random number data to the browser, which then assigns an electronic signature to the data and returns it to the server, along with the electronic signature certificate installed on the browser.

The browser assigns an electronic signature to the random number data, thereby indicating to the server that the browser has a secret key. As a result, the server can confirm that the client has the secret key associated with the certificate.

Note that before using the SSL-related functionality described here, you should configure SSL on the web server, which must be either Cosminexus HTTP Server or Microsoft IIS. For details about how to do this, follow the relevant subsections below.

- When using Cosminexus HTTP Server:

  See *7.2.5 SSL setup with Cosminexus HTTP Server*.

- When using Microsoft IIS:

  See *7.2.6 Microsoft IIS setup (in Web redirector environments)*.

7.2.2 Selecting a communication path security level

You can select an encryption level for the communication path from the Web application to the Web container. This selection is for Web server integration.

You can select one of three encryption levels: No protection (**NONE**), Tamper prevention (**INTEGRAL**), or Hiding (**CONFIDENTIAL**). SSL is used if you select the **INTEGRAL** or **CONFIDENTIAL** level.

This means protected pages can be accessed through SSL only. If the browser sends a request via HTTP to access a protected page, the Web container returns an HTTP response requesting the browser to access the same page via HTTPS. The browser then accesses the same page via HTTPS. You can select a security level not only for the entire Web application but also for certain pages within that application. This prevents non-HTTPS access to the protected pages.

Using the DD file (`WEB-INF/web.xml`), select a communication path encryption level for the Web applications included in the J2EE application. For details about how to configure the J2EE application, see *7.2.7 Setup in an execution environment*.

Note:

An attempt to access a protected page when a transfer destination HTTPS port number is not specified for Web server integration will result in a 403 error.

A protected page cannot be accessed regardless of whether a transfer destination HTTPS port number is specified when an in-process HTTP server or simple Web server is used (for compatibility with earlier versions). An attempt to access such a page will result in a 403 error.

Configure the transfer destination HTTP server port number by customizing the J2EE server properties. For details about this customization, see *7.2.7 Setup in an execution environment*.

7.2.3 Acquiring an SSL certificate

By using the servlet API, you can obtain both a certificate used for handshaking and the encryption algorithm selected by handshaking.

Client authentication allows you to obtain an X.509 certificate from the client as a `java.security.cert.X509Certificate` class object.

Note that when using the servlet, you can only obtain a client certificate through SSL client authentication. To perform SSL client authentication, you need to configure this authentication on the Web server.

With the Web container, you can use the servlet API to obtain RSA public keys whose key length is 512 to 2,048 bits or DSA public keys whose key length is 512 to 1,024 bits. You can obtain only those public keys that are supported by the Web server being used (for example, Microsoft IIS supports only RSA public keys).

If Microsoft IIS is used as the Web server for linkage with the Web container, it is impossible to obtain an SSL communication encryption algorithm by using `javax.servlet.request.cipher_suite` as defined in the Servlet v2.3 specifications.

7.2.4 Definitions in DD files

Using `web.xml`, select a communication path encryption level. Definitions in DD files are as follows.

- The *security-constraint*-*user-data-constraint* tag

  Use this tag to select an encryption level for encrypting authentication information and data by using SSL.

You can use `web.xml` to configure the settings for a Web application before deploying the application on the J2EE server. You can use an attributes file to configure the same settings for a Web application already deployed on the J2EE server. For details about the settings in attributes files, see *7.2.7(2) Configuring the J2EE application*.

7.2.5 SSL setup with Cosminexus HTTP Server

For SSL authentication or data encryption with Cosminexus HTTP Server, create a secret key, obtain a certificate from the Certification Authority (CA), and configure the Cosminexus HTTP Server definition file (`httpsd.conf`).

For client authentication, obtain a client certificate and CA certificate and configure the Cosminexus HTTP Server definition file (`httpsd.conf`).

The following describes how to configure SSL authentication, SSL encryption, and client authentication with Cosminexus HTTP Server. For details, see *5. SSL authentication and encryption* in the manual *HTTP Server*.

7.2.6 Microsoft IIS setup (in Web redirector environments)

For details about how to configure Microsoft IIS SSL, see the SSL help for Microsoft IIS.

7.2.7 Setup in an execution environment

To select a communication path encryption level, you need to configure the J2EE server and J2EE application.

(1) Configuring the J2EE server

To configure the J2EE server, use the Easy Setup definition file. Use the `<configuration>` tag of the logical J2EE server (j2ee-server) in the Easy Setup definition file to specify the port number for the Web server.

Specify this port number in the Easy Setup definition file as follows:

- webserver.connector.redirect_https.port

  Specify the HTTPS port number used by the Web server. The specified port number is only effective for Web server integration.

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) Configuring the J2EE application

To configure the J2EE application in an execution environment, use the attributes file and server management commands. To select a communication path encryption level, use the WAR attributes file.

- The *security-constraint-user-data-constraint* tag

  Select the encryption level required for encrypting authentication information and data with SSL.

  Note that you can use the WAR attributes file to configure the settings for a Web application already deployed on the J2EE server. You can use `web.xml` to configure the same settings for a Web application before deploying the application on the J2EE server.

For details about the WAR attributes file, see *3.7 WAR attributes file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

7.3 Using TLSv1.2 for SSL/TLS communication

This section describes the SSL/TLS communication functionality realized by RSA BSAFE SSL-J, which uses Cosminexus Developer's Kit for Java.

The table below shows the organization of this section.

Table 7-3: Organization of this section (Using TLSv1.2 for SSL/TLS communication)

Part	Title	Relevant information
Description	Overview of the SSL/TLS communication functionality realized by RSA BSAFE SSL-J	7.3.1
	Protocols and encryption suites	7.3.2
Implementation	Secure socket communication	7.3.3
Setup	SSL-J provider setup	7.3.4
	Setup in an execution environment (for HTTPS communication)	7.3.5
Operation	Deleting SSL-J providers	7.3.6
Precautions	Precautions for using the SSL-TLS communication functionality realized by SSL-J	7.3.7

To use TLSv1.2 on HTTP Server, specify TLSv1.2 in the `SSLProtocol` directive. For details, see *6.2.7 Directives with names starting with S* in the manual *HTTP Server*.

7.3.1 Overview of the SSL/TLS communication functionality realized by RSA BSAFE SSL-J

Secure SSL/TLS communication is possible with JSSE for Developer's Kit for Java. JSSE provides SSL and TLS protocol frameworks and implementations as standard. Using these frameworks and implementations enables secure SSL/TLS communication. For details about JSSE, see the Java SE 6 documents.

RSA BSAFE SSL-J provides the capability for SSL/TLS communication under security protocols such as TLSv1.2. This functionality is called the SSL/TLS communication functionality realized by SSL-J. RSA BSAFE SSL-J offers JSSE providers (SSL-J providers) based on JSSE's provider architecture.

These SSL-J providers should be registered for using the SSL/TLS communication functionality realized by SSL-J. For details about how to register them, see *7.3.4 SSL-J provider setup*.

To use the SSL/TLS communication functionality realized by SSL-J, you need to configure not only the protocol for the SSL-J providers but also the encryption suite supported by that protocol. The methods for configuring these components vary depending on the type of communication required, as follows:

For secure socket communication:

Use the API to configure the protocol and encryption suite for the socket. For details about how to configure these components, see *7.3.3 Secure socket communication*.

For HTTPS communication:

Use the application server's system properties to configure the protocol and encryption suite. For details about how to configure these components, see *7.3.5 Setup in an execution environment (for HTTPS communication)*.

7.3.2 Protocols and encryption suites

The following subsection describes the protocols and encryption suites compatible with the SSL/TLS communication functionality realized by SSL-J.

(1) Protocols

This communication functionality can be used with the following protocols:

- SSLv3
- TLSv1
- TLSv1.1
- TLSv1.2

Note:

The SSLv2Hello protocol is not supported by this communication functionality. As a result, this communication functionality does not enable communication with devices that only use the SSLv2 protocol. Ensure any associated devices use a protocol that is supported by this communication functionality.

(2) Encryption suites

The table below lists the encryption suites compatible with the SSL/TLS communication functionality realized by SSL-J. It also lists the protocols supported by these encryption suites.

Table 7-4: Encryption suites compatible with the SSL/TLS communication functionality realized by SSL-J, and supported protocols

No.	Encryption suite	Protocol			
		SSLv3	TLSv1	TLSv1.1	TLSv1.2
1	`TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384`	--	--	--	S
2	`TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384`	--	--	--	S
3	`TLS_RSA_WITH_AES_256_CBC_SHA256`	--	--	--	S+D
4	`TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384`	--	--	--	S
5	`TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384`	--	--	--	S
6	`TLS_DHE_RSA_WITH_AES_256_CBC_SHA256`	--	--	--	S+D
7	`TLS_DHE_DSS_WITH_AES_256_CBC_SHA256`	--	--	--	S+D
8	`TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA`	--	S	S	S
9	`TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA`	--	S	S	S
10	`TLS_RSA_WITH_AES_256_CBC_SHA`	S	S	S	S+D
11	`TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA`	--	S	S	S
12	`TLS_ECDH_RSA_WITH_AES_256_CBC_SHA`	--	S	S	S
13	`TLS_DHE_RSA_WITH_AES_256_CBC_SHA`	S	S	S	S+D
14	`TLS_DHE_DSS_WITH_AES_256_CBC_SHA`	S	S	S	S+D
15	`TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256`	--	--	--	S
16	`TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256`	--	--	--	S
17	`TLS_RSA_WITH_AES_128_CBC_SHA256`	--	--	--	S+D
18	`TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256`	--	--	--	S
19	`TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256`	--	--	--	S
20	`TLS_DHE_RSA_WITH_AES_128_CBC_SHA256`	--	--	--	S+D
21	`TLS_DHE_DSS_WITH_AES_128_CBC_SHA256`	--	--	--	S+D
22	`TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA`	--	S	S	S
23	`TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA`	--	S	S	S
24	`TLS_RSA_WITH_AES_128_CBC_SHA`	S	S	S	S+D

No.	Encryption suite	Protocol			
		SSLv3	TLSv1	TLSv1.1	TLSv1.2
25	`TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA`	--	S	S	S
26	`TLS_ECDH_RSA_WITH_AES_128_CBC_SHA`	--	S	S	S
27	`TLS_DHE_RSA_WITH_AES_128_CBC_SHA`	S	S	S	S+D
28	`TLS_DHE_DSS_WITH_AES_128_CBC_SHA`	S	S	S	S+D
29	`TLS_ECDHE_ECDSA_WITH_RC4_128_SHA`	--	S	S	S
30	`TLS_ECDHE_RSA_WITH_RC4_128_SHA`	--	S	S	S
31	`SSL_RSA_WITH_RC4_128_SHA`	S	S	S	S+D
32	`TLS_ECDH_ECDSA_WITH_RC4_128_SHA`	--	S	S	S
33	`TLS_ECDH_RSA_WITH_RC4_128_SHA`	--	S	S	S
34	`TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA`	--	S	S	S
35	`TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA`	--	S	S	S
36	`SSL_RSA_WITH_3DES_EDE_CBC_SHA`	S	S	S	S+D
37	`TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA`	--	S	S	S
38	`TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA`	--	S	S	S
39	`SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA`	S	S	S	S+D
40	`SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA`	S	S	S	S+D
41	`SSL_RSA_WITH_RC4_128_MD5`	S	S	S	S+D

Legend:

S: Encryption suite supported by protocol

S+D: Encryption suite supported by protocol and defined by RFC5246 (TLSv1.2)

--: Encryption suite not supported by protocol

Note: Some encryption suites are likely to become unusable when they are upgraded. They are supported as non-recommended encryption suites for the SSL/TLS communication functionality realized by SSL-J. For information about these encryption suites, see *7.3.7 Precautions for using the SSL/TLS communication functionality realized by SSL-J*.

7.3.3 Secure socket communication

Secure socket communication is provided by using the `SSLSocket` or `SSLServerSocket` class. When using the SSL/TLS communication functionality realized by SSL-J, configure the protocols and encryption suites for both the server-side and client-side sockets. The APIs used are listed below.

Table 7-5: APIs for configuring the protocols and encryption suites

Socket	Protocol	Encryption suite
Server side	`setEnabledProtocols` method of the `SSLServerSocket` class	`setEnabledCipherSuites` method of the `SSLServerSocket` class
Client side	`setEnabledProtocols` method of the `SSLSocket` class	`setEnabledCipherSuites` method of the `SSLSocket` class

A sample file is available, which contains configuration examples for the server-side socket. This file is stored in the following location:

Location of the sample file:

JDK-installation-path`/jre/lib/sslj/sample.txt`

If you wish to limit the use of protocols and encryption suites, use this file only to offer necessary protocols and encryption suites.

Below are the configuration examples for the server-side socket, which are included in the sample file.

Configuration examples for the server-side socket

1.	`SSLContext sslc = SSLContext.getDefault();//`
2.	`SSLServerSocketFactory sslssf = sslc.getServerSocketFactory();//`
3.	`SSLServerSocket sslss = (SSLServerSocket)sslssf.createServerSocket(<port number>);//`
4.	`String[] protocols = new String[]{"SSLv3","TLSv1","TLSv1.1","TLSv1.2"};//`
5.	`sslss.setEnabledProtocols(protocols);//`

6.	`String[] suites = new String[]{"TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384","TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384","TLS_RSA_WITH_AES_256_CBC_SHA256","TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384","TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384","TLS_DHE_RSA_WITH_AES_256_CBC_SHA256","TLS_DHE_DSS_WITH_AES_256_CBC_SHA256","TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA","TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA","TLS_RSA_WITH_AES_256_CBC_SHA","TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA","TLS_ECDH_RSA_WITH_AES_256_CBC_SHA","TLS_DHE_RSA_WITH_AES_256_CBC_SHA","TLS_DHE_DSS_WITH_AES_256_CBC_SHA","TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256","TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256","TLS_RSA_WITH_AES_128_CBC_SHA256","TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256","TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256","TLS_DHE_RSA_WITH_AES_128_CBC_SHA256","TLS_DHE_DSS_WITH_AES_128_CBC_SHA256","TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA","TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA","TLS_RSA_WITH_AES_128_CBC_SHA","TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA","TLS_ECDH_RSA_WITH_AES_128_CBC_SHA","TLS_DHE_RSA_WITH_AES_128_CBC_SHA","TLS_DHE_DSS_WITH_AES_128_CBC_SHA","TLS_ECDHE_ECDSA_WITH_RC4_128_SHA","TLS_ECDHE_RSA_WITH_RC4_128_SHA","SSL_RSA_WITH_RC4_128_SHA","TLS_ECDH_ECDSA_WITH_RC4_128_SHA","TLS_ECDH_RSA_WITH_RC4_128_SHA","TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA","TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA","SSL_RSA_WITH_3DES_EDE_CBC_SHA","TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA","TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA","SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA","SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA","SSL_RSA_WITH_RC4_128_MD5"};//`
7.	`sslss.setEnabledCipherSuites(suites);//`

The examples in the table are described below.

1. Obtains `SSLContext`, which is the SSL-J provider implementation encapsulated in `SSLContext.getDefault()`.
2. Uses `SSLContext` to retrieve `SSLServerSocketFactory`.
3. Uses `SSLServerSocketFactory` to create `SSLServerSocket`.
4. Creates a string array for protocols.
5. Sets the string array above in `SSLServerSocket`.
6. Creates a string array for encryption suites.
7. Sets the string array above in `SSLServerSocket`.

7.3.4 SSL-J provider setup

To use the SSL/TLS communication functionality realized by SSL-J, you need to register the SSL-J providers.

To register the providers, follow these steps:

1. Copy the SSL-J provider files.
2. Edit the Security property file.

Detailed information about these steps is given below.

(1) Copying the SSL-J provider files

Copy the following files from the SSL-J provider directory to the extended functionality directory:

- `sslj.jar`
- `cryptoj.jar`
- `certj.jar`

SSL-J provider directory:

JDK-installation-path`/jre/lib/sslj`

Extended functionality directory:

JDK-installation-path`/jre/lib/ext`

(2) Editing the security property file

Register the SSL-J providers in the security property file.

Security property file:

JDK-installation-path`/jre/lib/security/java.security`

Use the following format for registering the SSL-J providers:

Registration format:

```
security.provider.priority-level=provider-class-name
```

- For *priority-level*, specify the level of provider priority with a positive integer. Specifying a smaller number increases the priority.
- For *provider-class-name*, specify the SSL-J provider class names. These should be `com.rsa.jsafe.provider.JsafeJCE` and `com.rsa.jsse.JsseProvider`.

The Security property file already contains the providers offered by Java SE 6. Specify higher priority levels for the additional SSL-J providers (`com.rsa.jsafe.provider.JsafeJCE` and `com.rsa.jsse.JsseProvider`) than for the existing providers. In addition, reduce the priority levels for the existing providers.

Below is an example of registering the SSL-J providers in the Security property file. The file contents shown in this example might differ from those before registration, depending on how the platform and update are applied, and on differences in the user environment.

Example:

Before registration

```
security.provider.1=sun.security.provider.Sun
security.provider.2=sun.security.rsa.SunRsaSign
security.provider.3=com.sun.net.ssl.internal.ssl.Provider
security.provider.4=com.sun.crypto.provider.SunJCE
security.provider.5=sun.security.jgss.SunProvider
security.provider.6=com.sun.security.sasl.Provider
security.provider.7=org.jcp.xml.dsig.internal.dom.XMLDSigRI
security.provider.8=sun.security.smartcardio.SunPCSC
security.provider.9=sun.security.mscapi.SunMSCAPI
```

After registration

```
security.provider.1=com.rsa.jsafe.provider.JsafeJCE
security.provider.2=com.rsa.jsse.JsseProvider
security.provider.3=sun.security.provider.Sun
security.provider.4=sun.security.rsa.SunRsaSign
security.provider.5=com.sun.net.ssl.internal.ssl.Provider
security.provider.6=com.sun.crypto.provider.SunJCE
security.provider.7=sun.security.jgss.SunProvider
security.provider.8=com.sun.security.sasl.Provider
security.provider.9=org.jcp.xml.dsig.internal.dom.XMLDSigRI
security.provider.10=sun.security.smartcardio.SunPCSC
security.provider.11=sun.security.mscapi.SunMSCAPI
```

Characters in bold type indicate changes due to SSL-J provider registration. These changes have been made as follows:

- The SSL-J provider class names (`com.rsa.jsafe.provider.JsafeJCE` and `com.rsa.jsse.JsseProvider`) have been specified for priority levels 1 and 2.
- The priority levels for the previously registered providers have been changed to levels from 3 to 11.

7.3.5 Setup in an execution environment (for HTTPS communication)

To use the SSL/TLS communication functionality realized by SSL-J, you must configure the following:

- J2EE server
- Batch server
- Batch applications
- Java applications

(1) Configuring the J2EE server

You can use this communication functionality to provide HTTPS communication with applications running on a J2EE server. To configure the J2EE server, use `usrconf.properties` (user property file for J2EE server) or the Easy Setup definition file.

The table below shows the parameters to be specified.

Table 7-6: Settings for the SSL/TLS communication functionality realized by SSL-J for HTTPS communication

Specified parameter	Setting
`https.protocols`	Specifies the protocol.
`https.cipherSuites`	Specifies the encryption suites.

`usrconf.properties` (user property file for J2EE server) contains the default values for protocols and encryption suites. To use default settings that are commented out, delete the hash marks (#) added before the lines. If you want to limit the use of protocols and encryption suites, use this file only to offer necessary protocols and encryption suites.

Below are the default settings in the user property file.

Default settings in `usrconf.properties` (user property file for J2EE server)

```
# JDK SSL-J Settings
# If you want to enable SSL-J, please uncomment.
#https.protocols=SSLv3,TLSv1,TLSv1.1,TLSv1.2
#https.cipherSuites=TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,TLS_ECDHE_RSA_WITH_AES_2
56_CBC_SHA384,TLS_RSA_WITH_AES_256_CBC_SHA256,TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384
,TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384,TLS_DHE_RSA_WITH_AES_256_CBC_SHA256,TLS_DHE_DS
S_WITH_AES_256_CBC_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AE
S_256_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA,TLS_E
CDH_RSA_WITH_AES_256_CBC_SHA,TLS_DHE_RSA_WITH_AES_256_CBC_SHA,TLS_DHE_DSS_WITH_AES_2
56_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_128_CBC_SH
A256,TLS_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDH
_RSA_WITH_AES_128_CBC_SHA256,TLS_DHE_RSA_WITH_AES_128_CBC_SHA256,TLS_DHE_DSS_WITH_AE
S_128_CBC_SHA256,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC
_SHA,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDH_RSA_W
ITH_AES_128_CBC_SHA,TLS_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_DSS_WITH_AES_128_CBC_SH
A,TLS_ECDHE_ECDSA_WITH_RC4_128_SHA,TLS_ECDHE_RSA_WITH_RC4_128_SHA,SSL_RSA_WITH_RC4_1
28_SHA,TLS_ECDH_ECDSA_WITH_RC4_128_SHA,TLS_ECDH_RSA_WITH_RC4_128_SHA,TLS_ECDHE_ECDSA
_WITH_3DES_EDE_CBC_SHA,TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,SSL_RSA_WITH_3DES_EDE_CBC
_SHA,TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA,SSL_DHE
_RSA_WITH_3DES_EDE_CBC_SHA,SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA,SSL_RSA_WITH_RC4_128_MD
5
```

For details about `usrconf.properties` (user property file for J2EE server), see *2.4. usrconf.properties (user property file for J2EE server)* in the *uCosminexus Application Server Definition Reference Guide*.

(2) Configuring the batch server

You can use this communication functionality to provide HTTPS communication with applications running on a batch server. To configure a batch server, use `usrconf.properties` (user property file for batch servers).

For details about `usrconf.properties` (user property file for batch servers), see *3.3 use usrconf.properties (user property file for batch servers)* in the *uCosminexus Application Server Definition Reference Guide*.

(3) Configuring batch applications

To use the SSL/TLS communication functionality realized by SSL-J in batch applications, use `usrconf.properties` (user property file for batch applications). For details about `usrconf.properties` (user property file for batch applications), see *3.7 usrconf.properties (user property file for batch applications)* in the *uCosminexus Application Server Definition Reference Guide*.

(4) Configuring Java applications

To use the SSL/TLS communication functionality realized by SSL-J in Java applications, use `usrconf.properties` (user property file for Java applications). For details about `usrconf.properties` (user property file for Java applications), see *14.3 usrconf.properties (user property file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

7.3.6 Deleting SSL-J providers

If you do not plan to use the SSL/TLS communication functionality realized by SSL-J, you need to delete the registered SSL-J providers.

To delete the providers, follow these steps:

1. Delete the SSL-J provider file.

   Delete the SSL-J provider file, which is located in the extended functionality directory.

2. Edit the Security property file.

   Delete the SSL-J providers from the Security property file. Change the priority levels for the remaining providers back to the previous settings that were configured before registration of the deleted SSL-J providers. For details about the Security property file before SSL-J provider registration, see *7.3.4 SSL-J provider setup*.

7.3.7 Precautions for using the SSL/TLS communication functionality realized by SSL-J

- If, after registering SSL-J providers in the application server, you upgrade it or apply a patch, the SSL-J provider registration is canceled. In such case, register the SSL-J providers again.
- When you use this communication functionality to provide HTTPS communication, if you overwrite the system properties files `https.protocols` and `https.cipherSuites` with new data after using them to specify the protocol and encryption suite but before starting HTTPS communication, then the

new data becomes effective. As a result, communication might be disabled depending on the new data. To prevent this problem, do not overwrite the system properties while the J2EE application is active.

- Some encryption suites supported for this communication functionality are likely to become unavailable due to upgrading. These encryption suites are supported as non-recommended encryption suites for this communication functionality. They might become unavailable in the future. Thus, for continuous SSL/TLS communication, you should use the encryption suites shown in *7.3.2(2) Encryption suites*.

  The non-recommended encryption suites are listed below.

  - TLS_DH_anon_WITH_AES_256_CBC_SHA256
  - TLS_ECDH_anon_WITH_AES_256_CBC_SHA
  - TLS_DH_anon_WITH_AES_256_CBC_SHA
  - TLS_DH_anon_WITH_AES_128_CBC_SHA256
  - TLS_ECDH_anon_WITH_AES_128_CBC_SHA
  - TLS_DH_anon_WITH_AES_128_CBC_SHA
  - TLS_ECDH_anon_WITH_RC4_128_SHA
  - SSL_DH_anon_WITH_RC4_128_MD5
  - TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA
  - SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
  - TLS_RSA_WITH_NULL_SHA256
  - TLS_ECDHE_ECDSA_WITH_NULL_SHA
  - TLS_ECDHE_RSA_WITH_NULL_SHA
  - SSL_RSA_WITH_NULL_SHA
  - TLS_ECDH_ECDSA_WITH_NULL_SHA
  - TLS_ECDH_RSA_WITH_NULL_SHA
  - TLS_ECDH_anon_WITH_NULL_SHA
  - SSL_RSA_WITH_NULL_MD5
  - SSL_RSA_WITH_DES_CBC_SHA
  - SSL_DHE_RSA_WITH_DES_CBC_SHA
  - SSL_DHE_DSS_WITH_DES_CBC_SHA
  - SSL_DH_anon_WITH_DES_CBC_SHA

- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_DHE_DSS_WITH_AES_256_GCM_SHA384
- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_DH_DSS_WITH_AES_256_GCM_SHA384
- TLS_DH_RSA_WITH_AES_256_GCM_SHA384
- TLS_DH_DSS_WITH_AES_256_CBC_SHA256
- TLS_DH_RSA_WITH_AES_256_CBC_SHA256
- TLS_DH_RSA_WITH_AES_256_CBC_SHA
- TLS_DH_DSS_WITH_AES_256_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_DSS_WITH_AES_128_GCM_SHA256
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_DH_RSA_WITH_AES_128_GCM_SHA256
- TLS_DH_DSS_WITH_AES_128_GCM_SHA256
- TLS_DH_RSA_WITH_AES_128_CBC_SHA256
- TLS_DH_DSS_WITH_AES_128_CBC_SHA256

- TLS_DH_RSA_WITH_AES_128_CBC_SHA
- TLS_DH_DSS_WITH_AES_128_CBC_SHA
- SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_DH_DSS_WITH_DES_CBC_SHA
- SSL_DH_RSA_WITH_DES_CBC_SHA
- SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5
- TLS_DH_anon_WITH_AES_256_GCM_SHA384

Chapter

8. Directly Accessing Load Balancers Through the API and Controlling Them via the Operation Management Functionality

You can control load balancers by using the application server's operation management functionality. This chapter describes how to configure a load balancer so that it can be directly accessed via the API. This chapter also describes the settings for using the operation management functionality to control a load balancer.

8.1 Organization of this chapter

By using a load balancer, you can use its functionality to manage multiple servers with a single virtual IP address for efficient traffic distribution and improved performance. You can control load balancers by using the application server's operation management functionality.

This chapter describes the settings required to control the load balancers, directly through the API, using the operation management functionality.

The table below shows the organization of this chapter.

Table 8-1: Organization of this chapter (Directly Accessing Load Balancers Through the API and Controlling Them via the Operation Management Functionality)

Part	Title	Relevant information
Description	Directly accessing a load balancer through the API	8.2
	Load balancer APIs executed using the operation management functionality	8.3
Setup	Load balancer access environment setup	8.4
	Load balancer connection information setup with Management Server (Smart Composer functionality)	8.5
	Load balancer connection information setup with Virtual Server Manager	8.6

Note: This chapter does not include information on implementation, operation, or precautions.

8.2 Directly accessing a load balancer through the API

A load balancer available to the application server should be directly accessed through the API (SOAP or REST architecture). Before accessing the load balancer, you need to configure the settings. The following load balancers are compatible with the available access methods:

Direct access through API (SOAP architecture)

- BIG-IP v9
- BIG-IP v10.1
- BIG-IP v10.2
- BIG-IP v11

Direct access through API (REST architecture)

- AX2500

8.3 Load balancer APIs executed using the operation management functionality

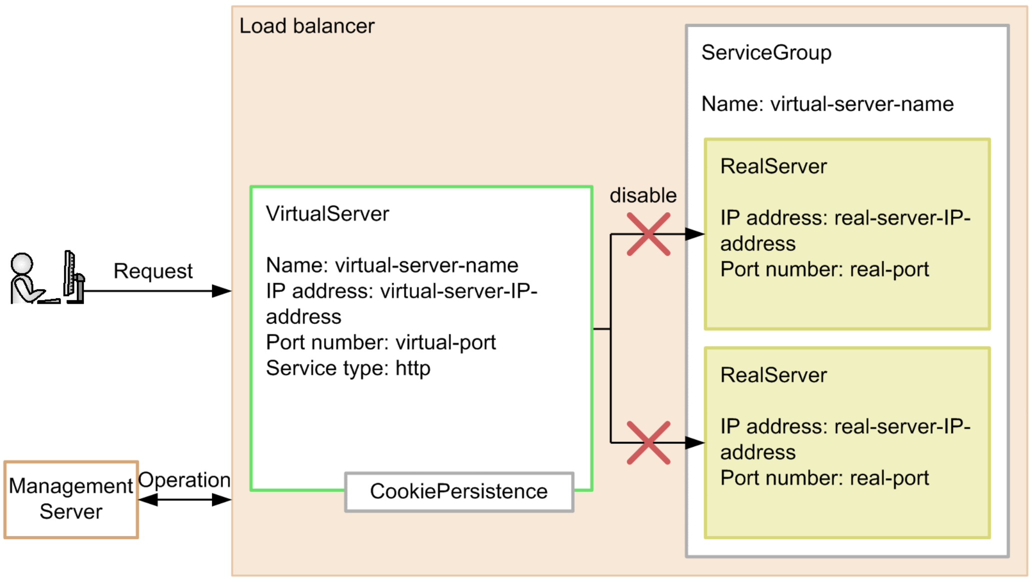
This section describes the load balancer APIs that are executed with the operation management functionality.

(1) Load balancer APIs executed using Management Server (Smart Composer functionality)

The following section describes load balancer APIs that are executed by using Smart Composer functionality.

Below is an example sequence diagram of a load balancer API. In this example, `cmx_build_system` uses cookie switching (for building a Web system).

Figure 8-1: Example load balancer API sequence diagram (cmx_build_system for building a Web system)


Load balancer
ServiceGroup
Name: virtual-server-name
RealServer
IP address: real-server-IP-address
Port number: real-port
disable
VirtualServer
Name: virtual-server-name
IP address: virtual-server-IP-address
Port number: virtual-port
Service type: http
Request
RealServer
IP address: real-server-IP-address
Port number: real-port
Management Server
Operation
CookiePersistence

The `VirtualServer` object represents the load balancer's virtual server, which accepts requests. One `VirtualServer` object is created for each management unit. The `ServiceGroup` object provides service management for requests accepted by the load balancer's `VirtualServer`. One `ServiceGroup` object is created for each `VirtualServer` object. The `RealServer` object represents a real server to which requests accepted by the load balancer's `VirtualServer` are transferred. The number of `RealServer` objects is equal to that of virtual servers belonging to the management unit.

Before accessing the load balancer through an API that uses persistent cookies for ACOS, you need to create `CookiePersistence` (cookie persistence). Using `CookiePersistence`, specify the hold period for each cookie-based session. Delete any used instances of `CookiePersistence` as necessary. For details about how to create and delete `CookiePersistence`, see the documentation for the load balancer used.

Reference note:

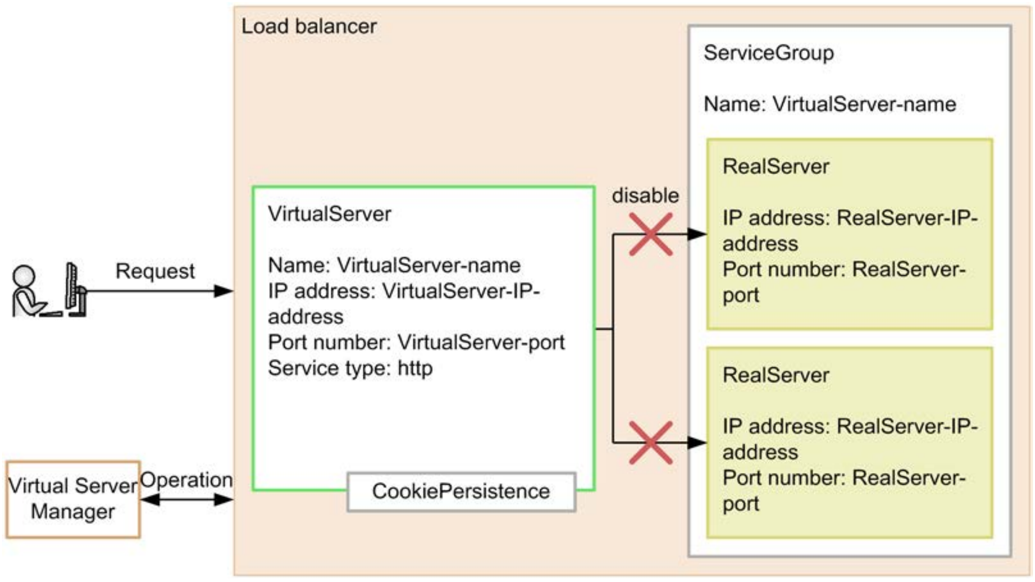
The names `VirtualServer`, `ServiceGroup`, `RealServer`, and `CookiePersistence`, vary depending on the load balancer product.

(2) Load balancer API executed using Virtual Server Manager

The following section describes load balancer APIs executed using Virtual Server Manager.

Below is an example sequence diagram of a load balancer API. In this example, `vmiunit update` uses cookie switching (for building a new system).

Figure 8-2: Example load balancer API sequence diagram (vmiunit update for building a new system)


Load balancer
ServiceGroup
Name: VirtualServer-name
RealServer
IP address: RealServer-IP-address
Port number: RealServer-port
disable
VirtualServer
Name: VirtualServer-name
IP address: VirtualServer-IP-address
Port number: VirtualServer-port
Service type: http
Request
RealServer
IP address: RealServer-IP-address
Port number: RealServer-port
Virtual Server Manager
Operation
CookiePersistence

The `VirtualServer` object represents the load balancer's virtual server, which accepts requests. One `VirtualServer` object is created for each management unit. The `ServiceGroup` object provides service management for requests accepted by the load balancer's `VirtualServer`. One `ServiceGroup` object is created for each `VirtualServer` object. The `RealServer` object represents a real server to which requests accepted by the load balancer's `VirtualServer` are transferred. The number of `RealServer` objects is equal to that of virtual servers belonging to the management unit.

Before accessing the load balancer through an API that uses persistent cookies for ACOS, you need to create `CookiePersistence` (cookie persistence). Using `CookiePersistence`, specify the hold period for each cookie-based session. Delete any used instances of `CookiePersistence` as necessary. For details about how to create and delete `CookiePersistence`, see the documentation for the load balancer used.

Reference note:

The names `VirtualServer`, `ServiceGroup`, `RealServer`, and `CookiePersistence`, vary depending on the load balancer product.

8.4 Load balancer access environment setup

To directly access a load balancer through an API, the load balancer access environment must be configured on the host that provides the operation management functionality.

(1) Access list (ACL) settings (ACOS)

If you are using a version of ACOS that is earlier than 2.4.3-P7, create the access list on a server machine that runs either Management Server or Virtual Server Manager. The necessary settings are given below. For details about how to create an access list, see the ACOS document.

- ID: `1`
- Action: `Permission`#
- Source address: `Multiple`#

#

To restrict access to the load balancer, specify arbitrary values for the Action and Source address attributes.

Note:

If you specify a number other than `1` for the ID attribute when creating the ACL, the load balancer will not be directly accessible via an API (REST architecture).

(2) Creating a cookie persistence template

To maintain a session through cookies, create a cookie persistence template on the host that provides the operation management functionality. The necessary settings are given below. For details about how to create a cookie persistence template, see the load balancer document.

- Cookie name: arbitrary value
- Expire: `0`

  If you specify `0` for the Expire attribute, only the current session is maintained.

(3) Configuring a trust store

By direct access through API, you communicate with the load balancer via HTTP or HTTPS. HTTPS communication requires a trust store that contains a reliable certificate. If you use HTTPS, specify or omit https in one of the following properties files.

For controlling the load balancer with Management Server:

- `lb.API.protocol.`*load-balancer-management-IP-address* in `lb.properties`

For controlling the load balancer with Virtual Server Manager:

- `lb.API.protocol` in *LB-connection-distinguished-name*`.properties`
- `lb.API.protocol` in `tierlb.properties`

Before communicating via HTTPS, follow the steps below to configure the trust store.

1. Obtain an SSL server certificate from the load balancer.

   For details about how to obtain an SSL certificate, see the load balancer document.

2. Execute JDK's `keytool` command on the host that provides the operation management functionality. The SSL server certificate obtained in step 1 will be registered in the trust store.

   Below is an example of executing JDK's `keytool` command.

```
Cosminexus-installation-directory/jdk/bin/keytool -import -file loadbalancer.cer -alias
loadbalancer -keystore C:\work\loadbalancer.keystore -storepass keystore_pass
```

   For details about this command, see the JDK document.

Note:

If you register the certificate in a non-default trust store (other than cacerts) for JDK, use the `javax.net.ssl.trustStore` parameter in `lb.properties` to specify the SSL server certificate's absolute path. If you register the certificate in the default trust store (cacerts), the absolute path does not need to be specified.

For BIG-IP, the default trust store (cacerts) must always be used.

This default trust store for JDK (cacerts) is located under *Cosminexus-installation-directory*`/jdk/jre/lib/security`. The initial password is `changeit`.

(4) hosts file settings (BIG-IP)

If you will be controlling BIG-IP from Management Server or Virtual Server Manager, register the host name and IP address of BIG-IP in the `hosts` file. However, there is no need to register this information in the `hosts` file when you have selected direct connection using ssh protocol to connect to BIG-IP.

8.5 Load balancer connection information setup with Management Server (Smart Composer functionality)

To configure load balancer access with the Smart Composer functionality, set the connection information in the load balancer definition property file (`lb.properties`) on the host that runs Management Server.

This section shows examples of configuring the connection information for the load balancers BIG-IP and AX2500 for direct access through an API.

BIG-IP (BIG-IP v9, BIG-IP v10.1, BIG-IP v10.2, or BIG-IP v11)

```
lb.list=192.168.100.10

lb.connect_type.192.168.100.10=API
#lb.API.port.192.168.100.10=443
lb.API.user.192.168.100.10=user01
lb.API.passwd.192.168.100.10=user01pw
#lb.API.API.timeout.192.168.100.10=10
```

AX2500

```
lb.list=192.168.10.100
lb.enable_passwd.192.168.10.100=adminpw

lb.connect_type.192.168.10.100=API
lb.API.user.192.168.10.100=user01
lb.API.passwd.192.168.10.100=user01pw
#lb.API.port.192.168.10.100=443
#lb.API.cookie_persistence_template.MyWebSystem.192.168.10.100=SC_COOKIE_TEMPNAME
#lb.API.API.timeout.192.168.10.100=10
javax.net.ssl.trustStore=C:\\work\\ACOS.keystore
javax.net.ssl.trustStorePassword=keystore_pass
```

For details about `lb.properties` (load balancer definition property file), see *4.5 lb.properties (load balancer definition property file)* in the *uCosminexus Application Server Definition Reference Guide*.

8.6 Load balancer connection information setup with Virtual Server Manager

If you are operating a load balancer in combination with the management unit, you can use Virtual Server Manager or the management unit to define load balancer connection information such as the type of load balancer used and the method of access.

8.6.1 Configuring load balancer connection information with Virtual Server Manager

To configure load balancer access with Virtual Server Manager, set the connection information in the load balancer access setup property file (*LB-connection-distinguished-name*`.properties`) on the server machine that is used for virtual-system management. The file name represented by *LB-connection-distinguished-name* should be a string of 31 or fewer characters starting with a single-byte alphabetic character. It can include alphanumeric characters, underscores (_), and hyphens (`-`).

Below are examples of configuring the connection information for the load balancers BIG-IP v9 (`lb_BIG-IPv9.properties`) and AX2500 (`lb_AX2500.properties`) for direct access through an API.

BIG-IP v9 (lb_BIG-IPv9.properties)

```
lb.type=BIG-IPv9
lb.host=192.168.2.14
lb.protocol=API
lb.port=443
lb.user=user01
lb.password=user01pw
lb.timeout=10
```

AX2500 (lb_AX2500.properties)

```
lb.type=ACOS
lb.host=192.168.2.13
lb.protocol=API
lb.port=443
lb.user=user01
lb.password=user01pw
lb.persistence.cookie-insert.templatename=VMI_COOKIE_TEMPNAME
lb.timeout=10
javax.net.ssl.trustStore=C:\\work\\ACOS.keystore
javax.net.ssl.trustStorePassword=keystore_pass
```

To use the load balancer connection information above, specify *LB-connection-distinguished-name* for the `lb.use` key in the property file (`tier.properties`) for each tier. For the configuration examples above, enter `lb_BIG-IPv9` to use BIG-IP v9 and `lb_AX2500` to use AX2500.

8.6.2 Configuring load balancer connection information with the management unit

To configure load balancer connection information with the management unit, the system creator should set the connection information in the load balancer access setup property file (`tierlb.properties`) for each tier that is managed with the management unit.

Below is an example of configuring the connection information for the load balancer BIG-IP v9 for direct access through an API.

```
lb.type=BIG-IPv9
lb.host=192.168.2.14
lb.protocol=API
lb.port=443
lb.user=user01
lb.password=user01pw
lb.timeout=10
```

To use the load balancer connection information above, specify `:unit:` for the `lb.use` key in the property file (`tier.properties`) for each tier.

Chapter
9. Server Management Command-based Security Role and Application Setup

This chapter describes how to set security roles and J2EE application security by using the server management commands.

9.1 Organization of this chapter

This chapter describes how to set security roles and J2EE application security by using the server management commands.

The table below shows how the chapter is organized

Table 9-1: Organization of this chapter (Server management command-based security role and application setup)

Part	Title	Relevant information
Setup	Security role setup	*9.2*
	Definition of security role references	*9.3*
	Security definition (method permission)	*9.4*
	Security definition (security identities)	*9.5*

Note: This chapter does not include information on implementation, operation, or precautions.

9.2 Security role setup

The following settings are required to manage users based on security roles.

The user and role settings are managed per J2EE server.

9.2.1 Setting users

Set up the users.

Execute the following command to register a user to the J2EE server.

Execute form

```
cjaddsec [server name] [-nameserver provider URL] -type user -name user name -password
password
```

Example

```
    cjaddsec MyServer -type user -name aps_m -password tiger
```

For details about the `cjaddsec` command, see *cjaddsec (Adding Users and Roles)* in the *uCosminexus Application Server Command Reference Guide*.

9.2.2 Setting roles

Set up and associate roles with users. In addition, add reference security roles to the Enterprise Beans, servlets, and JSP.

(1) Registering roles

Execute the following command to register a role to the J2EE server.

Execute form

```
cjaddsec [server name] [-nameserver provider URL] -type role -name role name
```

Example

```
    cjaddsec MyServer -type role -name manage
```

For details about the `cjaddsec` command, see *cjaddsec (Adding Users and Roles)* in the *uCosminexus Application Server Command Reference Guide*.

(2) Registering roles to users

Execute the following command to add a user to a role.

Execute form

```
cjmapsec [server name] [-nameserver provider URL] -role role name -user user name [-user user
name]
```

Example

```
cjmapsec MyServer -role manager -user aps_m
```

For details about the `cjmapsec` command, see *cjmapsec (Mapping Users and Roles)* in the *uCosminexus Application Server Command Reference Guide*.

(3) Adding security roles to Enterprise Beans

Define the Enterprise Bean security role settings.

(a) Attribute file to be edited

EJB-JAR attribute file

(b) Obtaining the attribute file to be edited and setting up attributes

■ Obtaining the attribute file

Execute the following command to obtain the EJB-JAR attribute file.

Execute form

```
cjgetappprop [server name] [-nameserver provider URL] -name J2EE application name -type ejb
-resname EJB-JAR display name -c EJB-JAR attribute file path
```

Example

```
cjgetappprop MyServer -name adder -type ejb -resname adder
-c C:\home\adder_ejb.xml
```

■ Setting up attributes

Execute the following command to reflect the EJB-JAR attribute file values.

Execute form

```
cjsetappprop [server name] [-nameserver provider URL] -name J2EE application name -type ejb
-resname EJB-JAR display name -c EJB-JAR attribute file path
```

Example

```
cjsetappprop MyServer -name adder -type ejb -resname adder
-c C:\home\adder_ejb.xml
```

(c) Attribute settings to be edited

The following table lists the Enterprise Bean security role (<security-role>) settings.

Item	Required	Tag name
Description	O	`<description>`
Role name	R	`<role-name>`
Security role name	O	`<linked-to>`

Legend:

R: Required, O: Optional

For details about the property settings, see subsection *3.3.1 Specification of the EJB-JAR attribute file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

(4) Adding security roles to servlets and JSP

Define the servlets and JSP security role settings.

(a) Attribute file to be edited

WAR attribute file

(b) Obtaining the attribute file to be edited and setting up attributes

■ Obtaining the attribute file

Execute the following command to obtain the WAR attribute file.

Execute form

```
cjgetappprop [server name] [-nameserver provider URL] -name J2EE application name -type war
-resname WAR display name -c WAR attribute file path
```

Example

```
cjgetappprop MyServer -name adder -type war -resname adder
-c C:\home\adder_war.xml
```

■ Setting up attributes

Execute the following command to reflect the WAR attribute file values.

Execute form

```
cjsetappprop [server name] [-nameserver provider URL] -name J2EE application name -type war
-resname WAR display name -c WAR attribute file path
```

Example

```
cjsetappprop MyServer -name adder -type war -resname adder
-c C:\home\adder_war.xml
```

(c) Attribute settings to be edited

The following table lists the Web application (servlets and JSP) security role reference (<security-role>) settings.

Item	Required	Tag name
Description	O	`<description>`
Role name	R	`<role-name>`
Security role name	O	`<linked-to>`

Legend:

R: Required, O: Optional

For details about the property settings, see subsection *3.7.1 Specification of the WAR attribute file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

9.3 Definition of security role references

Define a security role check reference to one or more methods of the Enterprise Bean and WAR, which are part of the J2EE application. This security check is different from the security services provided by containers.

9.3.1 Defining Enterprise Bean security role references

Define Enterprise Bean security role references.

(1) Attribute files to be edited

Edit the attributes files for each type of Enterprise Bean.

- Session Bean attribute file
- Entity Bean attribute file

(2) Obtaining the attribute file to be edited and setting up attributes

■ **Obtaining the attribute file**

Execute the following command to obtain the Enterprise Bean attribute file.

Execute form

```
cjgetappprop [server name] [-nameserver provider URL] -name J2EE application name -type ejb
-resname EJB-JAR display name/enterprise bean display name -c enterprise bean attribute file path
```

Example

```
cjgetappprop MyServer -name adder -type ejb -resname adder/
adder_eb -c C:\home\adder_ejb.xml
```

■ **Setting up attributes**

Execute the following command to reflect the Enterprise Bean attribute file values.

Execute form

```
cjsetappprop [server name] [-nameserver provider URL] -name J2EE application name -type ejb
-resname EJB-JAR display name/enterprise bean display name -c enterprise bean attribute file path
```

Example

```
cjsetappprop MyServer -name adder -type ejb -resname adder/
adder_eb -c C:\home\adder_ejb.xml
```

(3) Attribute settings to be edited

The following table lists the Enterprise Bean security role reference (<security-role-ref>) settings.

Item	Required	Tag name
Description	O	`<description>`
Security role reference name	R	`<role-name>`
Linked security role name#	O	`<role-link>`

Legend:

R: Required, O: Optional

#: Specify a set role name. To learn more about setting role names, see *9.2.2 Setting roles*. When setting up the EJB-JAR attribute file after setting <role-link>, the <role-link> value is cleared. Define the security role reference again.

For details about property settings, see:

- Subsection *3.4.1 Specification of the session bean attribute file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*
- Subsection *3.5.1 Specification of the entity bean attribute file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*

9.3.2 Defining servlet and JSP security role references

Define the Web application (servlets and JSP) security role references.

(1) Attribute files to be edited

Servlet attribute file

(2) Obtaining the attribute file to be edited and setting up attributes

■ Obtaining the attribute file

Execute the following command to obtain the servlet attribute file.

Execute form

```
cjgetappprop [server name] [-nameserver provider URL] -name J2EE application name -type war
-resname WAR display name/servlet and JSP display name -c servlet attribute file path
```

Example

```
cjgetappprop MyServer -name adder -type war -resname adder/
```

```
adder_sv -c C:\home\adder_war.xml
```

■ Setting up attributes

Execute the following command to reflect the WAR attribute file values.

Execute form

```
cjsetappprop [server name] [-nameserver provider URL] -name J2EE application name -type war
-resname WAR display name/servlet and JSP display name -c servlet attribute file path
```

Example

```
cjsetappprop MyServer -name adder -type war -resname adder/
adder_sv -c C:\home\adder_war.xml
```

(3) Attribute settings to be edited

The following table lists the Web application (servlets and JSP) security role reference (<security-role-ref>) settings.

Item	Required	Tag name
Description	O	`<description>`
Security role reference name	R	`<role-name>`
Linked security role name#	O	`<role-link>`

Legend:

R: Required, O: Optional

#: Specify a set role name. To learn more about setting role names, see *9.2.2 Setting roles*.

For details about the property settings, see subsection *3.9.1 Specification of the servlet attribute file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

9.4 Security definition (Method permission)

This section explains how to set method permissions. The method permission definition defines access control based on security roles. You can allow or deny access permissions for all users.

The method permissions can be set on the following methods:

- ■ Session Bean
  - Home interface `create` method
  - Component interface business and `remove` methods
- ■ Entity Bean
  - Home interface `create`, `finder`, and `home` methods
  - Component interface business and `remove` methods

Note that permissions are invalid for the following methods. The method permission defined by the component interface `remove` method is used to check the access permissions of these methods.

- javax.ejb.EJBHome `remove(javax.ejb.Handle handle)` method
- javax.ejb.EJBHome `remove(Object primaryKey)` method
- javax.ejb.EJBLocalHome `remove(Object primaryKey)` method

Note:

When the <Enable Scheduling> property is specified for a Stateless Session Bean of the CTM application, do not set `security` role-based access permissions on the home interface create method. The deployment will fail.

(1) Attribute files to be edited

Edit the following attributes files for each type of Enterprise Beans.

- Session Bean attribute file
- Entity Bean attribute file

(2) Obtaining the attribute file to be edited and setting up attributes

■ Obtaining the attribute file

Execute the following command to obtain the Enterprise Bean attribute file.

Execute form

```
cjgetappprop [server name] [-nameserver provider URL] -name J2EE application name -type ejb
-resname EJB-JAR display name/enterprise bean display name -c enterprise bean attribute file path
```

Example

```
cjgetappprop MyServer -name adder -type ejb -resname adder/
adder-eb -c C:\home\adder_ejb.xml
```

■ Setting up attributes

Execute the following command to reflect the Enterprise Bean attribute file values.

Execute form

```
cjsetappprop [server name] [-nameserver provider URL] -name J2EE application name -type ejb
-resname EJB-JAR display name/enterprise bean display name -c enterprise bean attribute file path
```

Example

```
cjsetappprop MyServer -name adder -type ejb -resname adder/
adder-eb -c C:\home\adder_ejb.xml
```

(3) Attribute settings to be edited

The following table lists the security definition (method permission) settings (<method_permission>).

Item	Required	Tag name
Description	O	`<description>`
Role name	O#	`<role-name>`
With method authentication	O#	`<unchecked>`
Method description	O	`<method> - <description>`
Interface type	O	`<method> - <intf>`
Method name	O	`<method> - <name>`

Legend: O: Optional

Note: When the security definition (method permission) settings (<method-permission>) are set as annotations, they cannot be changed.

#: To enable security management, specify either a role name or method authentication as shown below:

- To allow or deny access permissions based on security roles:

  Specify a role name (<role-name>).

- To grant access permissions to all users:

  Specify whether method authentication is enabled (<unchecked>).

To deny access permissions for all users, add information regarding the method with no access permission to <method> under <exclude-list> instead of <method-permission>.

For details about property settings, see:

- Subsection *3.4.1 Specification of the session bean attribute file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*
- Subsection *3.5.1 Specification of the entity bean attribute file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*

9.5 Security definition (Security identities)

There are the following two types of security identity settings.

- Runtime identity information used by Enterprise Beans
- Runtime identity information used by servlets

9.5.1 Enterprise Bean security identities

Define the Enterprise Bean security identities.

The two types of security identities, "UseCallerIdentity" and "RunAs", can be set.

- `UseCallerIdentity`

  The caller's security identity is used when a method is executed.

  Specify a security identity to be associated with an execution thread when an Enterprise Bean home or component interface method is executed.

- `RunAs`

  Operations were performed in accordance with the role identity specified by `role name`.

(1) Attribute files to be edited

Edit the following attributes files for each type of Enterprise Beans.

- Session Bean attribute file
- Entity Bean attribute file
- Message-driven Bean attribute file

(2) Obtaining the attribute file to be edited and setting up attributes

■ **Obtaining the attribute file**

Execute the following command to obtain the Enterprise Bean attribute file.

Execute form

```
cjgetappprop [server name] [-nameserver provider URL] -name J2EE application name -type ejb
-resname EJB-JAR display name/enterprise bean display name -c enterprise bean attribute file path
```

Example

```
cjgetappprop MyServer -name adder -type ejb -resname addr/
adder_eb -c C:\home\adder_ejb.xml
```

■ Setting up attributes

Execute the following command to reflect the Enterprise Bean attribute file values.

Execute form

```
cjsetappprop [server name] [-nameserver provider URL] -name J2EE application name -type ejb
-resname EJB-JAR display name/enterprise bean display name -c enterprise bean attribute file path
```

Example

```
cjsetappprop MyServer -name adder -type ejb -resname adder/
adder_eb -c C:\home\adder_ejb.xml
```

(3) Attribute settings to be edited

The following table lists the Enterprise Bean security definition (security identity) settings (<security-identity>).

Item	Required	Tag name
Description	O	`<description>`
With security identity setting	O#	`<use-caller-identity>`
Description on role identity	O	`<run-as> - <description>`
Security role name	O#	`<run-as> - <role-name>`
Name specified in security role	O	`<run-as> - <user-name>`

Legend: O: Optional

#: Set either of the following items depending on whether the caller's security identity is used when a method is executed.

- The caller's security identity is used when a method is executed:

  Specify whether the security identity is set (<use-caller-identity>).
- The caller's security identity is not used when a method is executed:

  Set role identity information (<run-as>).
- Only set role identity (<run-as>) information for message-driven beans.

For details about property settings, see:

- Subsection *3.4.1 Specification of the session bean attribute file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*

- Subsection *3.5.1 Specification of the entity bean attribute file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*
- Subsection *3.6.1 Specification of the message-driven bean attribute file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*

9.5.2 Servlet and JSP security identities

Define the servlet and JSP security identities.

Specify the runtime identity information that is used by a servlet when EJB is invoked.

(1) Attribute files to be edited

Servlet attribute file

(2) Obtaining the attribute file to be edited and setting up attributes

■ Obtaining the attribute file

Execute the following command to obtain the servlet attribute file.

Execute form

```
cjgetappprop [server name] [-nameserver provider URL] -name J2EE application name -type war
-resname WAR display name/servlet and JSP display name -c servlet attribute file path
```

Example

```
cjgetappprop MyServer -name adder -type war -resname adder/
adder_sv -c C:\home\adder_war.xml
```

■ Setting up attributes

Execute the following command to reflect the servlet attribute file values.

Execute form

```
cjsetappprop [server name] [-nameserver provider URL] -name J2EE application name -type war
-resname WAR display name/servlet and JSP display name -c servlet attribute file path
```

Example

```
cjsetappprop MyServer -name adder -type war -resname adder/
adder_sv -c C:\home\adder_war.xml
```

(3) Attribute settings to be edited

The following table lists the Web application (servlets and JSP) security role definition (security identity) settings.

Item	Required	Tag name
Description on role identity	O	`<run-as> - <description>`
Security role name	R	`<run-as> - <role-name>`
Name specified in security role	R	`<run-as> - <user-name>`

Legend:

R: Required, O: Optional

For details about the property settings, see subsection *3.9.1 Specification of the servlet attribute file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

Chapter

10. Management Portal-based Integrated User Management Operation (INTENTIONALLY DELETED)

Chapter

11. Management Portal-based Repository Management (Integrated User Management) (INTENTIONALLY DELETED)

Chapter

12. Resource Monitoring (Integrated User Management) (INTENTIONALLY DELETED)

Chapter 13. Commands Used in Integrated User Management

This chapter describes the input formats and functions, etc., used in integrated user management.

13.1 List of commands used in integrated user management

The following table lists the commands used in integrated user management.

Table 13-1: List of commands used in integrated user management

Command name	Category	Overview
convpw	Password encryption	Encrypts the password field content in the ldif file and outputs the results in the standard output
ssoexport	Referencing the single sign-on information repository	Places the content in the single sign-on information repository and outputs it in the standard output in the CSV format
ssogenkey	Creating encryption key files	Creates the encryption key file to be registered or referenced in the single sign-on information repository
ssoimport	Registering the single sign-on information repository	Reads the CSV file storing the single sign-on authentication information and registers the file content to the single sign-on information repository
uachpw	Password change	Changes the passwords used to access to the LDAP directory and DB servers in the integrated user management configuration file

13.2 Details of commands used in integrated user management

The following are the input formats and functions, etc., of the commands used in integrated user management.

Command directory

The commands used in integrated user management are stored in either of the following directories.

- Windows:

  Cosminexus installation directory`\manager\bin\`

- UNIX:

  `/opt/Cosminexus/manager/bin/`

Common specifications

Exit code

The following table lists the exit code of commands used in integrated user management.

Table 13-2: Exit code of commands used in integrated user management

Exit code	Meaning
0	The command succeeded.
1	An error occurred during command execution.
2	Either the command or argument at the time of server startup is incorrect.

convpw (Password encryption)

Format

`convpw [-f {md5|sha1}]` *ldif_file_name password_attribute*

Function

This command encrypts the ldif file when registering it in the user information repository. The command reads the specified ldif file, encrypts the contents specified by <password_attribute>, and then outputs the result in the standard output. Excluding the attribute name value specified by <password_attribute>, the ldif file content is output in the standard output as is.

When the attribute name specified by <password_attribute> is not found, the file content is output in the standard output as is.

This command can be executed by users with root privilege or permissions to execute the command. To learn more about how to grant permissions to execute the command to specific users, see *mngenvsetup (Configuring management groups)* in the *uCosminexus Application Server Command Reference Guide*.

Arguments

-f {md5|sha1}

This specifies the format used to encrypt the value specified by "password" in the file specified by <ldif_file_name>. When omitted, the default "sha1" is used. This argument is not case sensitive.

- md5

  The value is encrypted in the MD5 format.

- sha1

  The value is encrypted in the SHA-1 format.

<ldif_file_name>

This specifies the name of the ldif file storing the user information in which the password is to be converted.

<password_attribute>

This specifies the attribute name used when the password field content is converted.

Note

When Japanese is included in the ldif file to be converted, convert it to UTF-8 and then encode it in base64. For details about LDIF, see *RFC 2849 "The LDAP Data Interchange Format (LDIF) - Technical Specification"*.

ssoexport (Referencing the single sign-on information repository)

Format

`ssoexport [-n` *realm name*`] [-u` *user ID*`] [-scramble]` *useradmin_configfile*

Function

This outputs the user information stored in the single sign-on information repository in the standard output in the CSV format.

When the user information is changed in the single sign-on information repository, the information output by the `ssoexport` command is placed and edited in the CSV file, and then it is re-registered with the `ssoimport` command.

When the user information is fetched by using the `ssoexport` command, the actual data in "SECRETDATA" is decrypted.

The realm and user ID must be contained in the user information so that the `ssoexport` command converts and outputs the user information in the CSV format. Otherwise, the command does not output the user information.

This command can be executed by the users with root privilege or permissions to execute the command. To learn more about how to grant permissions to execute the command to specific users, see *mngenvsetup (Configuring management groups)* in the *uCosminexus Application Server Command Reference Guide*.

Arguments

-n <realm name>

This specifies the realm name to be searched for. When omitted, all the realm names are searched.

-u <user ID>

This specifies the user ID to be searched for. Wildcards (*) can be used in the user ID. Enclose a wildcard (*) with double quotation marks (").

Examples:

- -u "*"

  All users are fetched.
- -u "Ta*"

  The users whose user ID starts with "Ta" are fetched.
- -u "*no"

  The users whose user ID ends with "no" are fetched.

When omitted, all the user IDs are searched.

-scramble

Use this argument when passwords are scrambled by using the password change command (`uachpw`).

<useradmin_configfile>

This specifies the integrated user management configuration file (`ua.conf`).

Examples of input and output

The following are the examples of input and output when the realm name is "RealmA" and when the users starting with "s9" are to be fetched.

Input

Windows:

```
C:\>ssoexport -n RealmA -u "s9*" "C:\Program
Files\Hitachi\Cosminexus\manager\config\ua.conf"
```

UNIX:

```
% ssoexport -n RealmA -u "s9*" /opt/Cosminexus/manager/
config/ua.conf
```

Output

```
SecurityDomain,USERID,SECRETDATA,PUBLICDATA,LINK_J2EE,LINK_
REALMA
RealmA,s981234,abfdef,,
RealmA,s991234,ghijkl,,
```

Notes

- When no information is stored in the single sign-on information repository or the user information that corresponds to the specified realm name or user ID cannot be obtained, this command exits with the header only.
- Do not stop the LDAP directory server when the `ssoexport` command is executed. The command may exit without error messages.
- Do not execute the `ssoexport` and `ssoimport` commands simultaneously.
- Integrity is not ensured for the application that manages users at the destination registered by the `ssoimport` command. In other words, no error occurs even when the corresponding realm entry (or user entry) is not present in the single sign-on information repository. Be aware that no information is output when the `ssoexport` command is executed to reference the information. For the application that manages users at the destination, the `ssoexport` command outputs the value that corresponds to the realm entry immediately below the base DN.

ssogenkey (Creating encryption key files)

Format

`ssogenkey` *useradmin_configfile*

Function

The single sign-on authentication information is encrypted and saved, and it is decrypted when referenced. This command creates the encryption key used to encrypt and decrypt the information.

This command can be executed by the users with root privilege or permissions to execute the command. To learn more about how to grant permissions to execute the command to specific users, see *mngenvsetup (Configuring management groups)* in the *uCosminexus Application Server Command Reference Guide*.

Arguments

<useradmin_configfile>

This specifies the integrated user management configuration file (`ua.conf`).

Notes

- When the specified file is already present, create the backup in the same directory as the specified file by adding ".n" to the name.
- This command does not access the single sign-on information repository content. When the single sign-on authentication information has been already registered in the single sign-on information repository, execute the `ssoexport` command to fetch all the information, and then register it with the `ssoimport` command.

ssoimport (Registering the single sign-on information repository)

Format

`ssoimport {-a|-m|-d|-x} [-p] [-scramble]` *csvfile_name useradmin_configfile*

Function

This registers the CSV file obtained from the application that manages users (or the CSV file that is obtained from the application and then edited) in the single sign-on information repository. The actual data in the item ID "SECRETDATA" is encrypted when it is registered in the single sign-on information repository.

This command can be executed by the users with root privilege or permissions to execute the command. To learn more about how to grant permissions to execute the command to specific users, see *mngenvsetup (Configuring management groups)* in the *uCosminexus Application Server Command Reference Guide*.

Arguments

-a

This adds the file content specified by <csvfile_name> to the single sign-on information repository. When the user entry already exists in the single sign-on information repository, it outputs a warning message without adding the user entry, and then it continues the next process.

-m

This overwrites the single sign-on information repository with the file content specified by <csvfile_name>. When the user entry is not present in the single sign-on information repository, it adds the user entry.

-d

This deletes the file content specified by <csvfile_name> from the single sign-on information repository. When the user entry is not present in the single sign-on information repository, it outputs a warning message and continues the next process.

-x

This updates the single sign-on information repository according to the line operation instruction. For details about the line operation of CSV single sign-on authentication information files, see *14.4 CSV files containing single sign-on authentication information*.

-p

This outputs the list of realm and user names that are added, changed, or updated in the standard output.

-scramble

Use this argument when passwords are scrambled by using the password change command (`uachpw`).

<csvfile_name>

This specifies the CSV file to be registered in the single sign-on information repository.

<useradmin_configfile>

This specifies the integrated user management configuration file (`ua.conf`).

Input

In the following example, `userdata.csv` is the CSV file to be registered, and `ua.conf` is the single sign-on configuration file.

To add the CSV file content to the single sign-on information repository:

```
ssoimport -a userdata.csv ua.conf
```

To delete the CSV file content from the single sign-on information repository:

```
ssoimport -d userdata.csv ua.conf
```

To register the CSV file content in the single sign-on information repository in accordance with the operation specified in the actual data under OPERATION:

```
ssoimport -x userdata.csv ua.conf
```

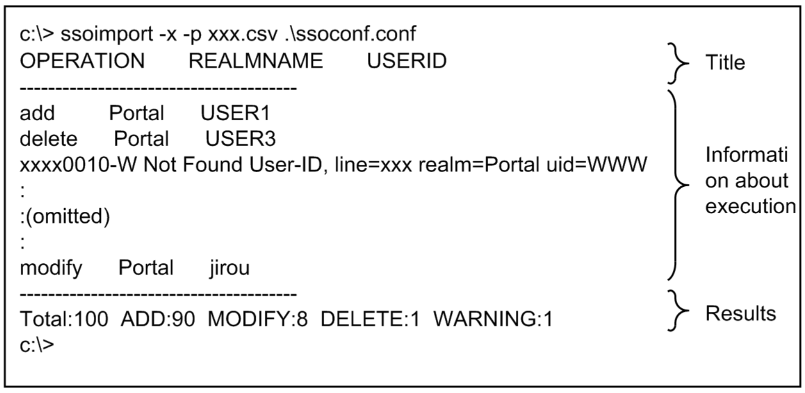
Output messages

The `ssoimport` command reads each line of the CSV file specified by the option and registers (changes or deletes) it in the single sign-on information repository. When the command is executed with the `-p` option, it outputs the information regarding the execution in the standard output. Note that warning and error messages occurred at the execution are output in the standard error output.

When the command is executed without the `-p` option, it outputs "results" only.

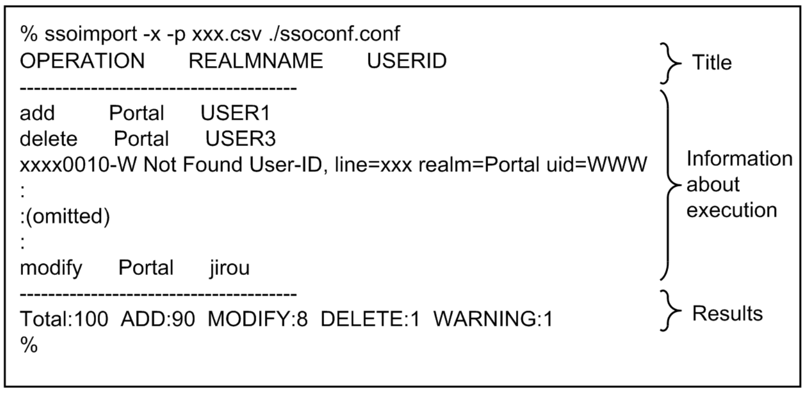
The following figure shows an example of output.

Figure 13-1: Example of the "ssoimport" command output (in Windows)



```
c:\> ssoimport -x -p xxx.csv .\ssoconf.conf
OPERATION      REALMNAME      USERID
---------------------------------------
add        Portal     USER1
delete     Portal     USER3
xxxx0010-W Not Found User-ID, line=xxx realm=Portal uid=WWW
:
:(omitted)
:
modify     Portal     jirou
---------------------------------------
Total:100  ADD:90  MODIFY:8  DELETE:1  WARNING:1
c:\>
```

Figure 13-2: Example of the "ssoimport" command output (in UNIX)



```
% ssoimport -x -p xxx.csv ./ssoconf.conf
OPERATION      REALMNAME      USERID
---------------------------------------
add        Portal     USER1
delete     Portal     USER3
xxxx0010-W Not Found User-ID, line=xxx realm=Portal uid=WWW
:
:(omitted)
:
modify     Portal     jirou
---------------------------------------
Total:100  ADD:90  MODIFY:8  DELETE:1  WARNING:1
%
```

The information about the execution is shown under `OPERATION`, `REALMNAME`, and

`USERID`.

Information about the execution

The corresponding information under the `OPERATION`, `REALMNAME`, and `USERID` headers are displayed together with the warning and error messages generated during the execution.

OPERATION

One of the operation types listed in the following table is displayed.

Table 13-3: ssoimport command operation types

Item	Description
add	The information has been added.
modify	The information has been changed (overwritten).
delete	The information has been deleted.

REALMNAME

The target realm name is displayed. The value is specified in "REALMNAME" of the CSV file.

USERID

The target user ID is displayed. The value is specified in "USERID" of the CSV file.

Execution results

The following table lists the execution results to be displayed.

Table 13-4: ssoimport command execution results

Item	Description
Total	This indicates the number of target lines.
ADD	This indicates the number of entries added to the single sign-on information repository.
MODIFY	This indicates the number of entries changed in the single sign-on information repository.
DELETE	This indicates the number of entries deleted from the single sign-on information repository.
WARNING	This indicates the number of warning messages generated during the execution.

Notes

- When deleting the user information of the JAAS user management application, execute `ssoexport` to obtain the user information, delete the users of the application that manages users, and then update the information with the `-m` option.
- Do not execute the `ssoexport` and `ssoimport` commands simultaneously.
- Integrity is not ensured for the application that manages users at the destination registered by the `ssoimport` command. In other words, no error occurs even when the corresponding realm entry (or user entry) is not present in the single sign-on information repository. Be aware that no information is output when the `ssoexport` command is executed to reference the information. For the application that manages users at the destination, the `ssoexport` command outputs the value that corresponds to the realm entry immediately below the base DN.

uachpw (Password change)

Format

`uachpw [-scramble] [-ldap.`*n password*`] [-db.`*n password*`]`
useradmin_configfile

Function

This changes the passwords used to access to the LDAP directory and DB servers in the integrated user management configuration file (`ua.conf`). It can also scramble passwords.

This command can be executed by the users with root privilege or permissions to execute the command. To learn more about granting permissions to execute the command to specific users, see *mngenvsetup (Configuring management groups)* in the *uCosminexus Application Server Command Reference Guide*.

Arguments

-scramble

Use this argument when scrambling the password to be changed.

-ldap.<n> <password>

This changes the password used to access the LDAP directory server specified by <n>. <n> contains the LDAP number defined in the integrated user management configuration file. <password> contains a new password.

-db.<n> <password>

This changes the password used to access to the DB server specified by <n>. <n> contains the JDBC number defined in the integrated user management configuration file. <password> contains a new password.

<useradmin_configfile>

This specifies the integrated user management configuration file (`ua.conf`). This argument is not optional.

Input

In the following example, the password defined in the integrated user management configuration file (`ua.conf`) is changed and scrambled by using the `-scramble` option.

- The LDAP access information 0 password is changed to "diradmin".
- The LDAP access information 1 password is changed to "administrator".
- The DB access information 0 password is changed to "tiger".

```
% uachpw -scramble -ldap.0 diradmin -ldap.1 administrator -db.0
tiger ua.conf
```

Notes

- When scrambling the password with the `-scramble` option, set com.cosminexus.admin.auth.passwordScramble.enable of `usrconf.properties` to true. Otherwise, the LDAP directory and DB servers will not be accessible, as the password is not decrypted.
- The maximum number of characters in the password is 30.
- The options are not case sensitive.
- Do not execute commands concurrently.
- Do not include files other than the integrated user management configuration file (`ua.conf`) in the arguments.

Chapter

14. Files Used by Integrated User Management

This chapter describes the format, location, functionality, specifiable options, etc. of the files used by Integrated User Management.

14.1 List of files used by integrated user management

The table below lists the files used by integrated user management.

Table 14-1: List of files used by integrated user management

File name	Classification	Description	Relevant information
`jaas.conf`	JAAS configuration file	Configures the settings necessary for using the user authentication library and single sign-on library functionality.	14.2
`ua.conf`	Integrated user management configuration file	Configuration file for using the JAAS-compatible user management and single sign-on functionality.	14.3
(optional)	CSV file for single sign-on authentication information	Configures authentication information for single sign-on.	14.4

14.2 jaas.conf (JAAS configuration file)

(1) Format

This is a JAAS configuration file that is necessary in order to use the user authentication library and single sign-on library functionality.

Options can be specified as follows:

```
Application {
  login-module-name Flag ModuleOptions;
};
```

(2) File location

- In Windows

  Cosminexus-installation-directory`\manager\config\`

- In UNIX

  `/opt/Cosminexus/manager/config/`

(3) Functionality

This file is used to configure settings that are necessary in order to use the user authentication library and single sign-on library functionality. Before using these features, a JAAS configuration file must be created and distributed to each host. Before distributing the file, take necessary precautions against possible eavesdropping.

(4) Options to be specified

Following table describes the options, along with their names.

Option name	Description
`Application`	Specify an application name. We recommend using a name that can uniquely identify the application. The specified name is used to instantiate the `LoginContext` class. Names starting with the character strings shown below are used by Cosminexus. Therefore, do not specify an application name starting with any of the following characters: • `jp.co.hitachi.soft` • `com.hitachi.software` • `com.cosminexus`

Option name	Description
Login module name	Specify the authentication engine to be used. Specify one of the following login modules: • `WebPasswordLoginModule` Specify this to use a password for user authentication. • `WebCertificateLoginModule` Specify this to use a client certificate for user authentication. • `WebPasswordLDAPLoginModule` Specify this to use the authentication functionality of an LDAP directory server for user authentication. • `WebPasswordJDBCLoginModule` Specify this to use a database as the user information repository. • `DelegationLoginModule` Specify this to call a custom login module. • `WebSSOLoginModule` Specify this to use the single sign-on functionality.
`Flag`	Specify a flag to be used to change the behavior according to whether the correct login module was called by `LoginContext`. For details about the flag to be specified, see the appropriate JAAS documentation.
`ModuleOptions`	Specify the options necessary to run the login module. For details about the options to be specified, see the sections from *(5) Options to be specified for WebPasswordLoginModule* to *(10) Options to be specified for WebPasswordLDAPLoginModule*.

(5) Options to be specified for WebPasswordLoginModule

The table below shows the options to be specified for `WebPasswordLoginModule`, along with their default values.

Option	Description	Default value
`com.cosminexus.admin.auth.ldap.r`	Specify an LDAP configuration number defined in *14.3(3) Repository access-related parameters*. The specified value must be a number that identifies a configuration that can reference the user information repository. The value must be enclosed by `""`. To specify more than one LDAP configuration number, separate the numbers by a comma (`,`). If more than one LDAP configuration number is specified and the first LDAP directory server specified goes down, the system automatically switches to a different LDAP directory server. The specified value is used by any functionality that needs to reference the repository, for example, when using `WebPasswordLoginModule` for login.	`0`

Option	Description	Default value
`com.cosminexus.admin.auth.ldap.w`	Specify an LDAP configuration number defined in *14.3(3) Repository access-related parameters*. The specified value must be a number that identifies a configuration that can update the user information repository. The value must be enclosed by `""`. To specify more than one LDAP configuration number, separate the numbers by a comma (`,`). If more than one LDAP configuration number is specified and the first LDAP directory server specified goes down, the system automatically switches to a different LDAP directory server. The specified value is used by any functionality that manages the contents of the repository, such as the password change functionality.	`0`
`com.cosminexus.admin.auth.sso.ldap.w`	Specify an LDAP configuration number defined in *14.3(3) Repository access-related parameters*. Specify this option when you are using the `PasswordUtil` class to change both the password and single sign-on authentication information, or when you are using the LDAP connection failover to change single sign-on authentication information. The specified value must be a number that identifies a configuration that can update the single sign-on in information repository. The value must be enclosed by `""`. To specify more than one LDAP configuration number, separate the numbers by a comma (`,`). If more than one LDAP configuration number is specified and the first LDAP directory server specified goes down, the system automatically switches to a different LDAP directory server. The specified value supersedes any value specified in `com.cosminexus.admin.auth.sso.ldap.w` within the integrated user management configuration file.	*value-specified-in-the-integrated-user-management-configuration-file*
`com.cosminexus.admin.auth.realm`	Specify the realm to be authenticated as a character string.	None
`com.cosminexus.admin.auth.keep_password`	Specify `true` or `false` to control whether to hold the password of a user who has logged into the realm in an integrated user management session. The specification is not case-sensitive. If you specify `true`, the password is to be held. If you specify `false`, the password is not to be held. If a user is already logged into the realm, the already-held password is not overwritten even if `true` is specified. If `false` is specified for this option and a user uses `WebPasswordLDAPLoginModule` to log into the same realm in the same session, he or she is required to enter his or her user ID and password for each subsequent login. The specified value supersedes any value specified in `com.cosminexus.admin.auth.keep_password` within the integrated user management configuration file.	*value-specified-in-the-integrated-user-management-configuration-file*

Option	Description	Default value
com.cosminexus.admin.auth.keep_password.encrypt	If true is specified in com.cosminexus.admin.auth.keep_password, specify true or false to control whether to encrypt a password that is held. The specification is not case-sensitive. If you specify true, the password is to be encrypted. If you specify false, the password is not to be encrypted. The specified value supersedes any value specified in com.cosminexus.admin.auth.keep_password.encrypt within the integrated user management configuration file.	*value-specified-in-the-integrated-user-management-configuration-file*
com.cosminexus.admin.auth.gsession.keep_password	If the session failover functionality of integrated user management is enabled and true is specified in com.cosminexus.admin.auth.keep_password, specify true or false in this option to control whether to manage a password held in an integrated user management session with the session failover functionality. If true is specified: The password is held in the global session. If false is specified: The password is not held in the global session. If a session failover occurs and a user uses WebPasswordLDAPLoginModule to log into the same realm in the same session, he or she is required to enter his or her user ID and password for each subsequent login. Specification example: com.cosminexus.admin.auth.gsession.keep_password=true	*value-specified-in-the-integrated-user-management-configuration-file*

(6) Options to be specified for WebSSOLoginModule

The table below shows the options to be specified for WebSSOLoginModule, along with their default values.

Option	Description	Default value
com.cosminexus.admin.auth.sso	Specify the identifier of the login module to be called from WebSSOLoginModule. The specified identifier is used to read necessary information from the JAAS-compatible user management configuration file. If this is omitted, the Cosminexus standard login module (WebPasswordLoginModule) is assumed.	WebPasswordLoginModule

Option	Description	Default value
`com.cosminexus.admin.auth.sso.ldap.r`	Specify an LDAP configuration number defined in *14.3(3) Repository access-related parameters*. The specified value must be a number that identifies a configuration that can reference the single sign-on information repository. The value must be enclosed by `""`. To specify more than one LDAP configuration number, separate the numbers by a comma (`,`). If more than one LDAP configuration number is specified and the first LDAP directory server specified goes down, the system automatically switches to a different LDAP directory server. The specified value is used by any functionality that needs to reference the repository, for example, when using `WebSSOLoginModule` to perform single sign-on. The specified value supersedes any value specified in `com.cosminexus.admin.auth.sso.ldap.r` within the integrated user management configuration file.	*value-specified-in-the-integrated-user-management-configuration-file*
`com.cosminexus.admin.auth.sso.ldap.w`	Specify an LDAP configuration number defined in *14.3(3) Repository access-related parameters*. The specified value must be a number that identifies a configuration that can update the single sign-on information repository. The value must be enclosed by `""`. To specify more than one LDAP configuration number, separate the numbers by a comma (`,`). If more than one LDAP configuration number is specified and the first LDAP directory server specified goes down, the system automatically switches to a different LDAP directory server. The specified value is used by any functionality that needs to update the repository, such as the password change functionality. The specified value supersedes any value specified in `com.cosminexus.admin.auth.sso.ldap.w` within the integrated user management configuration file.	*value-specified-in-the-integrated-user-management-configuration-file*
`com.cosminexus.admin.auth.realm`	Specify the realm to be authenticated as a character string.	None

(7) Options to be specified for DelegationLoginModule

The table below shows the options to be specified for `DelegationLoginModule`, along with their default values.

Option	Description	Default value
`com.cosminexus.admin.auth.custom.lm`	Specify the name of a custom login module (or class name) to be called by `DelegationLoginModule`, as a character string. The specified name must be a fully qualified name.	None
`com.cosminexus.admin.auth.realm`	Specify the realm to be authenticated as a character string.	None

(8) Options to be specified for WebCertificateLoginModule

The table below shows the options to be specified for

`WebCertificateLoginModule`, along with their default values.

Option	Description	Default value
`com.cosminexus.admin.auth.ldap.r`	Specify an LDAP configuration number defined in *14.3(3) Repository access-related parameters*. The specified value must be a number that identifies a configuration that can reference the user information repository. The value must be enclosed by `""`. To specify more than one LDAP configuration number, separate the numbers by a comma (`,`). If more than one LDAP configuration number is specified and the first LDAP directory server specified goes down, the system automatically switches to a different LDAP directory server. The specified value is used by any functionality that needs to reference the repository, for example, when using `WebCertificateLoginModule` for login.	`0`
`com.cosminexus.admin.auth.realm`	Specify the realm to be authenticated as a character string.	None

(9) Options to be specified for WebPasswordJDBCLoginModule

The table below shows the options to be specified for `WebPasswordJDBCLoginModule`, along with their default values.

Option	Description	Default value
`com.cosminexus.admin.auth.jdbc.r`	Specify a JDBC configuration number defined in *14.3(3) Repository access-related parameters*. The specified value must be a number that identifies a configuration that can reference the user information repository. The value must be enclosed by `""`. The specified value is used by any functionality that needs to reference the repository, for example, when using `WebPasswordJDBCLoginModule` for login.	`0`
`com.cosminexus.admin.auth.realm`	Specify the realm to be authenticated as a character string.	None
`com.cosminexus.admin.auth.keep_password`	Specify `true` or `false` to control whether to hold the password of a user who has logged into the realm in an integrated user management session. The specification is not case-sensitive. If you specify `true`, the password is to be held. If you specify `false`, the password is not to be held. If a user is already logged into the realm, the already-held password is not overwritten even if `true` is specified. If `false` is specified for this option and a user uses `WebPasswordLDAPLoginModule` to log into the same realm in the same session, he or she is required to enter his or her user ID and password for each subsequent login. The specified value supersedes any value specified in `com.cosminexus.admin.auth.keep_password` within the integrated user management configuration file.	*value-specified-in-the-integrated-user-management-configuration-file*

Option	Description	Default value
`com.cosminexus.admin.auth.keep_password.encrypt`	If `true` is specified in `com.cosminexus.admin.auth.keep_password`, specify `true` or `false` to control whether to encrypt a password that is held. The specification is not case-sensitive. If you specify `true`, the password is to be encrypted. If you specify `false`, the password is not to be encrypted. The specified value supersedes any value specified in `com.cosminexus.admin.auth.keep_password.encrypt` within the integrated user management configuration file.	*value-specified-in-the-integrated-user-management-configuration-file*
`com.cosminexus.admin.auth.gsession.keep_password`	If the session failover functionality of integrated user management is enabled and `true` is specified in `com.cosminexus.admin.auth.keep_password`, specify `true` or `false` in this option to control whether to manage a password held in an integrated user management session with the session failover functionality. If `true` is specified: The password is held in the global session. If `false` is specified: The password is not held in the global session. If a session failover occurs and a user uses `WebPasswordLDAPLoginModule` to log into the same realm in the same session, he or she is required to enter his or her user ID and password for each subsequent login. Specification example: `com.cosminexus.admin.auth.gsession.keep_password=true`	*value-specified-in-the-integrated-user-management-configuration-file*

(10) Options to be specified for WebPasswordLDAPLoginModule

The table below shows the options to be specified for `WebPasswordLDAPLoginModule`, along with their default values.

Option	Description	Default value
`com.cosminexus.admin.auth.ldap.r`	Specify an LDAP configuration number defined in *14.3(3) Repository access-related parameters*. The specified value must be a number that identifies a configuration that can reference the user information repository. The value must be enclosed by `""`. To specify more than one LDAP configuration number, separate the numbers by a comma (`,`). If more than one LDAP configuration number is specified and the first LDAP directory server specified goes down, the system automatically switches to a different LDAP directory server. The specified value is used by any functionality that needs to reference the repository, for example, when using `WebPasswordLDAPLoginModule` for login.	`0`

Option	Description	Default value
`com.cosminexus.admin.auth.ldap.w`	Specify an LDAP configuration number defined in *14.3(3) Repository access-related parameters*. The specified value must be a number that identifies a configuration that can update the user information repository. The value must be enclosed by `""`. To specify more than one LDAP configuration number, separate the numbers by a comma (`,`). If more than one LDAP configuration number is specified and the first LDAP directory server specified goes down, the system automatically switches to a different LDAP directory server. The specified value is used by any functionality that manages the contents of the repository, such as the password change functionality.	`0`
`com.cosminexus.admin.auth.sso.ldap.w`	Specify an LDAP configuration number defined in *14.3(3) Repository access-related parameters*. Specify this option when you are using the `PasswordUtil` class to change both the password and single sign-on authentication information, or when you are using the LDAP connection failover to change single sign-on authentication information. The specified value must be a number that identifies a configuration that can update the single sign-on information repository. The value must be enclosed by `""`. To specify more than one LDAP configuration number, separate the numbers by a comma (`,`). If more than one LDAP configuration number is specified and the first LDAP directory server specified goes down, the system automatically switches to a different LDAP directory server. The specified value supersedes any value specified in `com.cosminexus.admin.auth.sso.ldap.w` within the integrated user management configuration file.	*value-specified-in-the-integrated-user-management-configuration-file*
`com.cosminexus.admin.auth.realm`	Specify the realm to be authenticated as a character string.	None
`com.cosminexus.admin.auth.keep_password`	Specify `true` or `false` to control whether to hold the password of a user who has logged into the realm in an integrated user management session. The specification is not case-sensitive. If you specify `true`, the password is to be held. If you specify `false`, the password is not to be held. If a user is already logged into the realm, the already-held password is not overwritten even if `true` is specified. If `false` is specified for this option and a user uses `WebPasswordLDAPLoginModule` to log into the same realm in the same session, he or she is required to enter his or her user ID and password for each subsequent login. The specified value supersedes any value specified in `com.cosminexus.admin.auth.keep_password` within the integrated user management configuration file.	*value-specified-in-the-integrated-user-management-configuration-file*

Option	Description	Default value
`com.cosminexus.admin.auth.keep_password.encrypt`	If `true` is specified in `com.cosminexus.admin.auth.keep_password`, specify `true` or `false` to control whether to encrypt a password that is held. The specification is not case-sensitive. If you specify `true`, the password is to be encrypted. If you specify `false`, the password is not to be encrypted. The specified value supersedes any value specified in `com.cosminexus.admin.auth.keep_password.encrypt` within the integrated user management configuration file.	*value-specified-in-the-integrated-user-management-configuration-file*
`com.cosminexus.admin.auth.gsession.keep_password`	If the session failover functionality of integrated user management is enabled and `true` is specified in `com.cosminexus.admin.auth.keep_password`, specify `true` or `false` in this option to control whether to manage a password held in an integrated user management session with the session failover functionality. If `true` is specified: The password is held in the global session. If `false` is specified: The password is not held in the global session. If a session failover occurs and a user uses `WebPasswordLDAPLoginModule` to log into the same realm in the same session, he or she is required to enter his or her user ID and password for each subsequent login. Specification example: `com.cosminexus.admin.auth.gsession.keep_password=true`	*value-specified-in-the-integrated-user-management-configuration-file*

14.3 ua.conf (integrated user management configuration file)

(1) Format

This is a configuration file for using the JAAS-compatible user management and single sign-on functionality.

(2) File location

- In Windows

  Cosminexus-installation-directory`\manager\config\`

- In UNIX

  `/opt/Cosminexus/manager/config/`

(3) Repository access-related parameters

Repository access-related parameters define information related to access to the JAAS-compatible user management repository (LDAP directory server or database). These definitions are used to access the repository from relevant login modules or various commands.

You can define an item more than once by appending a different configuration number (either LDAP or JDBC). Configuration numbers must start with 0 and increment by 1. If a number is skipped, the current set of definitions is terminated there. In the following example, it is assumed that 0 and 1 are defined. (The number 3 is ignored because 2 is skipped.)

Example

```
java.naming.provider.url.0=ldap://localhost:389
java.naming.provider.url.1=ldap://localhost:389
java.naming.provider.url.3=ldap://localhost:389
#java.naming.provider.url.3 is ignored.
```

JNDI and JDBC can have their own sets of definitions and each set starts with "0".

(a) JNDI parameters

JNDI parameters define information that is necessary in order to use JNDI to access the LDAP directory server. You can specify a parameter for accessing the LDAP directory server more than once by incrementing the LDAP configuration number from 0.

Option	Description	Default value
`java.naming.provider.url`	Specify the URL of the repository (LDAP directory server) as a character string. For details, see the appropriate Java JNDI description.	None
`java.naming.security.principal`	Specify the identifier of the authenticator to be used to access the repository (LDAP directory server), as a character string. For details, see the appropriate Java JNDI description.	None
`java.naming.security.credentials`	Specify the password corresponding to `java.naming.security.principal.n`, as a character string. For details, see the appropriate Java JNDI description.	None
`com.cosminexus.admin.auth.ldap.basedn`	Specify the base DN of the repository under JAAS-compatible user management, as a character string.	None
`com.cosminexus.admin.auth.ldap.attr.userid`	Specify the attribute name of a user login ID, as a character string.	`uid`
`com.cosminexus.admin.auth.ldap.search.userrdn`	Specify `true` or `false` to control whether to search the repository for a user entry (RDN) under JAAS-compatible user management. If the attribute name of a user login ID differs from the user entry (RDN), specify `true`. The specification is not case-sensitive.	`false`
`com.cosminexus.admin.auth.ldap.search.scope`	If the repository is to be searched for a user entry (RDN) under JAAS-compatible user management, specify a search level of `onelevel` (search to one level below only) or `subtree` (search to all levels below). The specification is not case-sensitive.	`onelevel`
`com.cosminexus.admin.auth.ldap.attr.password`	Specify the attribute name of a user password, as a character string.	`userPassword`
`com.cosminexus.admin.auth.ldap.pool.enable`	Specify `true` or `false` to control whether to use LDAP connection pools. The specification is not case-sensitive.	`false`
`com.cosminexus.admin.auth.ldap.pool.max`	Specify the maximum number of LDAP connection pools. If an incoming request causes the maximum number to be exceeded, the system waits for a pool to become empty. Specify an integer from `0` to `2147483647`. If the specified value is equal to or less than `0`, `100` is assumed.	`100`

Option	Description	Default value
`com.cosminexus.admin.auth.ldap.pool.max_spare`	Specify the maximum number of empty LDAP connection pools. The specified maximum number might be exceeded temporarily, but will be adjusted at intervals of the time specified by `com.cosminexus.admin.auth.ldap.pool.gc_interval`. Specify an integer from `0` to `2147483647`. If the specified value exceeds that specified for `com.cosminexus.admin.auth.ldap.pool.max`, the value specified for `com.cosminexus.admin.auth.ldap.pool.max` is assumed. If the specified value is equal to or less than `0`, half of the value specified for `com.cosminexus.admin.auth.ldap.pool.max` is assumed. If the value specified for `com.cosminexus.admin.auth.ldap.pool.max` is an odd number, the value is rounded down. If the specified value is `1`, `1` is assumed.	`50`
`com.cosminexus.admin.auth.ldap.pool.min_spare`	Specify the number of new pools to be established when the number of empty LDAP connection pools becomes `0` (including when the number is initialized). Specify an integer from `0` to `2147483647`. If the specified value exceeds that specified for com.cosminexus.admin.auth.ldap.pool.max_spare, the value specified for com.cosminexus.admin.auth.ldap.pool.max_spare is assumed. If the specified value is equal to or less than `0`, half of the value specified for `com.cosminexus.admin.auth.ldap.pool.max_spare` is assumed. If the value specified for `com.cosminexus.admin.auth.ldap.pool.max_spare` is an odd number, the value is rounded down. If the specified value is `1`, `1` is assumed.	`10`
`com.cosminexus.admin.auth.ldap.pool.gc_interval`	Specify the time interval for adjusting the number of empty LDAP connection pools as an integer from `0` to `2147483647` (in seconds). See the description for `com.cosminexus.admin.auth.ldap.pool.max_spare`. If the specified value is equal to or less than `0`, this functionality does not work. (The number of pools increases to that specified for `com.cosminexus.admin.auth.ldap.pool.max` and no pool is deleted.)	`60`
`com.cosminexus.admin.auth.ldap.conn.retry.count`	Specify the number of retries to be made if LDAP connection fails, as an integer from `0` to `2147483647`.	`1`
`com.cosminexus.admin.auth.ldap.conn.retry.wait`	Specify the time interval (in ms) for retries to be made if LDAP connection fails, as an integer from `0` to `2147483647`.	`0`

Option	Description	Default value
`com.cosminexus.admin.auth.ldap.certificate.attr.userid`	Specify a character string indicating the attribute name to be used as a user ID after decomposition of a DN stored in a certificate. The specification is not case-sensitive. If there are two or more instances of the same attribute name when extracting a user ID, the first value found is used.	`cn`
`com.cosminexus.admin.auth.ldap.password.encrypt`	Specify the format of passwords stored in the repository. `WebPasswordLoginModule` uses the specified format to compare passwords. • `sha1`: SHA-1 format • `none`: Plain text • `md5`: MD5 format The specified character string is not case-sensitive. If the string other than the above is specified, `sha1` is assumed. If `com.cosminexus.admin.auth.ldap.password.encrypt.ex` is specified, this parameter is ignored.	`sha1`
`com.cosminexus.admin.auth.ldap.password.encrypt.ex`	If the password format used is not one of those provided as standard, specify the fully qualified name of the class for password conversion. If this parameter is omitted or if the specified value cannot be found, the format specified in `com.cosminexus.admin.auth.ldap.password.encrypt` is used to compare passwords.	None
`com.cosminexus.admin.auth.ldap.directory.kind`	Specify the type of LDAP directory server to be connected. `AD`: Specify this to use Active Directory. `ETC`: Specify this to use an LDAP directory server other than Active Directory. If the value other than the above is specified, `ETC` is set.	`ETC`
`com.cosminexus.admin.auth.ldap.conn.read_timeout`	Specify the read timeout for the LDAP directory server as an integer from `0` to `3600` (in seconds). If `0` is specified, no timeout occurs.	`3`
`com.sun.jndi.ldap.connect.timeout`	Specify the connection timeout (in ms) for the LDAP directory server, as an integer equal to or greater than `0`. If the specified integer is equal to or less than `0`, the timeout value used in the network protocol, such as TCP, is used.	LDAP provider specification

(b) JDBC parameters

JDBC parameters define information that is necessary in order to use JDBC to access the database. You can specify a parameter for accessing the database more than once by incrementing the JDBC configuration number from 0.

Option	Description	Default value
`com.cosminexus.admin.auth.jdbc.driver`	Specify the class name of a JDBC driver corresponding to the database to be used. Specify the location of the JDBC driver in the class path of the J2EE server.	`JP.co.Hitachi.soft.DBPSV_Driver.JdbcDbpsvDriver`
`com.cosminexus.admin.auth.jdbc.conn.url`	Specify the URL for connecting to the database, as a character string. The URL must be in the following format: Specification example: `jdbc:<subprotocol>:<subname>`	None
`com.cosminexus.admin.auth.jdbc.conn.user`	Specify a character string indicating a database user connecting as a proxy. If this is omitted, it is assumed that there is no database user connecting as a proxy.	No proxy
`com.cosminexus.admin.auth.jdbc.conn.password`	Specify the password of a database user connecting as a proxy, as a character string. If `com.cosminexus.admin.auth.jdbc.conn.user` is omitted, this parameter is ignored. If `com.cosminexus.admin.auth.jdbc.conn.user` is specified and this parameter is omitted, a null character is assumed.	Null character
`com.cosminexus.admin.auth.jdbc.pool.enable`	Specify `true` or `false` to control whether to use JDBC connection pools. The specification is not case-sensitive.	`false`
`com.cosminexus.admin.auth.jdbc.pool.max`	Specify the maximum number of JDBC connection pools. If an incoming request causes the maximum number to be exceeded, the system waits for a pool to become empty. Specify an integer from `0` to `2147483647`. If the specified value is equal to or less than `0`, `100` is assumed.	`100`
`com.cosminexus.admin.auth.jdbc.pool.max_spare`	Specify the maximum number of empty JDBC connection pools. The specified maximum number might be exceeded temporarily, but will be adjusted at intervals of the time specified by `com.cosminexus.admin.auth.jdbc.pool.gc_interval`. Specify an integer from `0` to `2147483647`. If the specified value exceeds that specified for `com.cosminexus.admin.auth.jdbc.pool.max`, the value specified for `com.cosminexus.admin.auth.jdbc.pool.max` is assumed. If the specified value is equal to or less than `0`, half of the value specified for `com.cosminexus.admin.auth.jdbc.pool.max` is assumed. If the value specified for `com.cosminexus.admin.auth.jdbc.pool.max` is an odd number, the value is rounded down. If the specified value is `1`, `1` is assumed.	`50`

Option	Description	Default value
`com.cosminexus.admin.auth.jdbc.pool.min_spare`	Specify the number of new pools to be established when the number of empty JDBC connection pools becomes `0` (including when the number is initialized). Specify an integer from `0` to `2147483647`. If the specified value exceeds that specified for com.cosminexus.admin.auth.jdbc.pool.max_spare, the value specified for com.cosminexus.admin.auth.jdbc.pool.max_spare is assumed. If the specified value is equal to or less than `0`, half of the value specified for `com.cosminexus.admin.auth.jdbc.pool.max_spare` is assumed. If the value specified for `com.cosminexus.admin.auth.jdbc.pool.max_spare` is an odd number, the value is rounded down. If the specified value is `1`, `1` is assumed.	`10`
`com.cosminexus.admin.auth.jdbc.pool.gc_interval`	Specify the time interval for adjusting the number of empty JDBC connection pools as an integer from `0` to `2147483647` (in seconds). See the description for `com.cosminexus.admin.auth.jdbc.pool.max_spare`. If the specified value is equal to or less than `0`, this functionality does not work. The number of pools increases to that specified for `com.cosminexus.admin.auth.jdbc.pool.max` and no pool is deleted.	`60`
`com.cosminexus.admin.auth.jdbc.conn.retry.count`	Specify the number of retries to be made if JDBC connection fails, as an integer from `0` to `2147483647`.	`1`
`com.cosminexus.admin.auth.jdbc.conn.retry.wait`	Specify the time interval (in ms) for retries to be made if JDBC connection fails, as an integer from `0` to `2147483647`.	`0`
`com.cosminexus.admin.auth.jdbc.sql`	Specify the SQL `SELECT` statement for searching for a password. The specified `SELECT` statement must be in the following format: (Specification format) `SELECT` *column-name* `FROM` *table-name* `WHERE` *search-condition* The search condition can only contain a "`?`" as an `IN` parameter placeholder. The value is replaced with the user ID specified during authentication.	None

Option	Description	Default value
`com.cosminexus.admin.auth.jdbc.password.type`	Specify the value type of the column containing the password. Specify one of the following values to indicate the type supported in the Java language. • `string`: • A password value is taken as a String type from the database. It is equivalent to the SQL data type `CHAR/VARCHAR/LONGVARCHAR`. • `byte`: A password value is taken as a `byte[]` type from the database. It is equivalent to the SQL data type `VARBINARY/LONGVARBINARY`. The specified keyword is not case-sensitive. If the string other than the above keywords is specified, `string` is assumed.	`string`
`com.cosminexus.admin.auth.jdbc.password.encrypt`	Specify the format of passwords stored in the repository. `WebPasswordLoginModule` uses the specified format to compare passwords. • `sha1`: SHA-1 format • `none`: Plain text • `md5`: MD5 format The specified keyword is not case-sensitive. If the value other than the above keywords is specified, `none` is assumed. If `com.cosminexus.admin.auth.jdbc.password.encrypt.ex` is specified, this parameter is ignored. If `sha1` or `md5` is specified in this parameter, specify `byte` in the `com.cosminexus.admin.auth.jdbc.password.type` parameter.	`none`
`com.cosminexus.admin.auth.jdbc.password.encrypt.ex`	If the password format used is not one of those provided as standard, specify the fully qualified name of the class for password conversion. If this parameter is omitted or if the specified parameter cannot be found, the encryption format specified in `com.cosminexus.admin.auth.ldap.password.encrypt` is used to compare passwords.	None

(c) API parameters

API parameters define information used by APIs when referencing or updating the user information repository on the LDAP directory server.

Append `.`<*name*> to items. The `.`<*name*> element is an identifier that indicates use of an API to reference or update the user information repository. The specified *name* is also specified in the `LdapUserDataManager` constructor.

name format

Application's-Java-package-name`.`*internal-name*

Internal name: Character string consisting of alphanumeric characters (`A`-`Z`, `a`-`z`, and `0-9`), and period (`.`).

Example
```
    com.cosminexus.admin.auth.api.repository.ldap.config.<c
    om.cosminexus.admin.auth.Example>=1
```

The `name` element can be defined more than once by changing the value. To define the `name` element more than once, each instance must be made unique in the configuration file. Specify the `name` element as a character string consisting of alphanumeric characters (`A-Z`, `a-z`, and `0-9`), and period (`.`). If any other character is used, the `name` element might not be recognized correctly.

Option	Description	Default value
`com.cosminexus.admin.auth.api.repository.ldap.config`	Specify the identifier (or an LDAP configuration number specified in the appropriate JNDI parameter) indicating the LDAP directory server used by API.	None

(d) Single sign-on parameters

Single sign-on parameters are necessary in order to use the single sign-on functionality. Two different types of information must be specified. One is for information used by `WebSSOLoginModule`, and the other is for definition information used for calling a custom login module. Append `.name` to definitions of the latter type (options starting from `com.cosminexus.admin.auth.sso.lm` in the following table). The `.name` element is the identifier of a login module called from `WebSSOLoginModule`.

The `.name` element can be defined more than once by changing the value. The specified `.name` is used in the JAAS configuration.

Format

item-used-by-WebSSOLoginModule=*value*

definition-for-calling-the-custom-login-module`.name`=*value*

Example

```
com.cosminexus.admin.auth.sso.keyfile=d:/tmp/DES3key.key
com.cosminexus.admin.auth.sso.lm.krb5=com.sun.security.modu
le.Krb5LoginModule
com.cosminexus.admin.auth.sso.param.userid.Krb5=javax.secur
ity.auth.login.name
...
```

Option	Description	Default value
com.cosminexus.admin.auth.sso.keyfile	Specify the absolute path and name of the file containing key information for encryption when registering single sign-on information. If this file is not specified, a LoginException will occur when using the single sign-on functionality for login or during use of the password change functionality (using the PasswordUtil class). If com.cosminexus.admin.auth.sso.encrypt=none, the value specified in this parameter is ignored.	None
com.cosminexus.admin.auth.sso.encrypt	Specify the product to be used to encrypt single sign-on authentication information (SecretData). • KEYMATECRYPTO: For compatibility with 05-00. If this value is specified, JCE is used. • JCE: JCE is used. • NONE: No encryption functionality is used. The specified keyword is not case-sensitive.	NONE
com.cosminexus.admin.auth.sso.ldap.r	Specify the LDAP configuration number defined at the beginning of this section. The specified value must be a number that identifies a configuration that can reference the single sign-on information repository. The specified value is used by any functionality that needs to reference the repository, for example, when using WebSSOLoginModule to perform a single sign-on.	0
com.cosminexus.admin.auth.sso.ldap.w	Specify the LDAP configuration number defined at the beginning of this section. The specified value must be a number that identifies a configuration that can update the single sign-on in information repository. The specified value is used by any functionality that manages the repository, such as the password change functionality or SSOExport and SSOImport commands.	0
com.cosminexus.admin.auth.sso.lm	Specify the login module name (or class name) of each application called by WebSSOLoginModule Specify the full package name.	None
com.cosminexus.admin.auth.sso.param.userid	Specify the name of the parameter for passing a user ID registered in the single sign-on information repository. The specified value is set by WebSSOLoginModule before the login() method is called. Note that this occurs only if the value is authenticated.	com.cosminexus.admin.auth.sso.userid
com.cosminexus.admin.auth.sso.param.secdat	Specify the name of the parameter for passing encrypted information registered in the single sign-on information repository. The key with the specified parameter name and the key value are set by WebSSOLoginModule before the login() method is called. Note that the key and its value are set only if they are already authenticated. Decrypted data is set for the value.	com.cosminexus.admin.auth.sso.secdat

Option	Description	Default value
`com.cosminexus.admin.auth.sso.param.pubdat`	Specify the name of the parameter for passing non-encrypted information registered in the single sign-on information repository. The specified value is set by `WebSSOLoginModule` before the `login()` method is called. Note that this occurs only if the value is authenticated. Be careful not to use a duplicate parameter name when specifying `com.cosminexus.admin.auth.sso.param.userid`, `com.cosminexus.admin.auth.sso.param.secdat`, or `com.cosminexus.admin.auth.sso.param.pubdat`. If a duplicate parameter name is used, the contents are not guaranteed.	`com.cosminexus.admin.auth.sso.pubdat`

(e) Custom login module parameters

Custom login module parameters are necessary in order to call a custom login module from `DelegationLoginModule` or `WebSSOLoginModule`.

Option	Description	Default value
`com.cosminexus.admin.auth.custom.modules`	Specify the absolute path of the directory containing a custom login module and its related classes (such as `Principal` and `Credential` classes).	None

(f) Cosminexus standard login module parameters

Cosminexus standard login module parameters define general information for the Cosminexus standard login modules.

Option	Description	Default value
`com.cosminexus.admin.auth.keep_password`	Specify `true` or `false` to control whether to hold the password of a user who has logged into the realm in an integrated user management session. The specification is not case-sensitive. If you specify `true`, the password is to be held. If you specify `false`, the password is not to be held. If a user is already logged into the realm, the already-held password is not overwritten even if `true` is specified. If `false` is specified for this option and a user uses `WebPasswordLDAPLoginModule` to log into the same realm in the same session, he or she is required to enter his or her user ID and password for each subsequent login.	`false`
`com.cosminexus.admin.auth.keep_password.encrypt`	If `true` is specified in `com.cosminexus.admin.auth.keep_password`, specify `true` or `false` to control whether to encrypt a password that is held. The specification is not case-sensitive. If you specify `true`, the password is to be encrypted. If you specify `false`, the password is not to be encrypted.	`true`

Option	Description	Default value
`com.cosminexus.admin.auth.param_check.enable`	If `true` is specified for `com.cosminexus.admin.auth.param_check.enable` and one of the following Cosminexus standard login modules is used to log in with a login user name starting or ending with a space, an exception will occur: • `WebPasswordLoginModule` • `WebPasswordJDBCLoginModule` • `WebCertificateLoginModule` • `WebPasswordLDAPLoginModule`	`true`
`com.cosminexus.admin.auth.gsession.keep_password`	If the session failover functionality of integrated user management is enabled and `true` is specified in `com.cosminexus.admin.auth.keep_password`, specify `true` or `false` in this option to control whether to manage a password held in an integrated user management session with the session failover functionality. If `true` is specified: The password is held in the global session. If `false` is specified: The password is not held in the global session. If a session failover occurs and a user uses `WebPasswordLDAPLoginModule` to log into the same realm in the same session, he or she is required to enter his or her user ID and password for each subsequent login. Specification example: `com.cosminexus.admin.auth.gsession.keep_password=true`	`false`

(g) Other parameters

A trace file contains definition information related to the entire user management using the Cosminexus standard login module.

Option	Description	Default value
`com.cosminexus.admin.auth.trace.prefix`	Specify the full path and name of the trace file (without an extension). In output, the specified value has an extension of `.`*`n`*`.log` appended. (*n* indicates the number of files from `1` to the maximum number of files (up to `16`).) If this specification is omitted, no trace log is output.	None
`com.cosminexus.admin.auth.trace.level`	Specify the trace level as a number. Trace information is output for levels equal to or below the specified level. `0`: If a login or logout fails, a trace log is output. `5`: If a login or logout succeeds or times out, a trace log is output.	`0`

Option	Description	Default value
`com.cosminexus.admin.auth.trace.rotate`	Specify the number of trace files as a number from `1` to `16`.	`4`
`com.cosminexus.admin.auth.trace.size`	Specify the maximum size of a trace file as a number from `4096` to `2147483647`. If a log file exceeds the specified size, subsequent logs are recorded in a new file with the next file number. If the final log file (the file with a file number equal to the maximum number of files) reaches the maximum file size, log file `1` is overwritten.	`65536`
`com.cosminexus.admin.auth.sfo.disable`	If a session failover filter is set, disable the session failover support of integrated user management. If `true` is specified: Disables the session failover support. If `false` is specified: Enables the session failover support.	`false`

14.4 CSV files containing single sign-on authentication information

To specify single sign-on authentication information, you must create a file in CSV format. The CSV file is described below.

14.4.1 Basic CSV file specifications

A comma (`,`) is used to separate each item. A new line is used to separate each record.

An item that comprises a character string separated by a comma is treated as a piece of data (or data field), regardless of whether the string is enclosed in double-quotations (`"`). To include a comma in an item, enclose the entire item in double quotations.

Example: "`ou=Cosminexus,o=HitachiHitachi`" is specified as an RDN name.

```
...,"ou=Cosminexus,o=Hitachi",...
```

To include a double quotation in an item, specify two double quotations in succession and enclose the entire item by double quotations.

Example: To set "`pass"wd`" as an Alias:

```
...,"pass""wd",...
```

A space preceding or following a comma (`,`) is included in the item.

14.4.2 Definition file for acquiring user information

(1) Additional CSV file specifications

In addition to the specifications shown in *14.4.1 Basic CSV file specifications*, the following specifications are added for specifying lists of attributes:

- The specification order of items is predetermined.
- If two successive commas are specified, it is assumed that the option between the commas (`,`) is omitted.

(2) Specification formats

End each line with a new line and specify the following items in the line, separated by commas:

Format	Item		
Format 1	#		
Format 2	Attribute name	Alias	Subcontext

Format 1

This format is used to specify a comment. If a line starts with # (in the first column), any text between the # and the end of the line is assumed to be a comment.

Format 2

This format is used to specify the information shown below within a line.

Table 14-2: Information to be specified (definition file for acquiring user information)

Functionality	Meaning	Attribute
Attribute name	Specify a name that starts with an alphabetic character and consists of alphabetic (ASCII) and numeric characters, and hyphens. The alphabetic characters are not case-sensitive.	Required
Alias	Specify the name for referencing the program.	Optional
Subcontext	In order to obtain information for a non-authenticated user entry, specify the RDN relative to the user entry for the entry to be obtained.	Optional

14.4.3 Definition file for adding or modifying user information

This file is used to specify the object classes of LDAP directory server entries.

(1) Additional CSV file specifications

In addition to the specifications shown in *14.4.1 Basic CSV file specifications*, the following specifications are added for specifying lists of attributes:

- The specification order of items is predetermined.
- If two successive commas are specified, it is assumed that the option between the commas (`,`) is omitted.

(2) Specification formats

End each line with a new line and specify the following items in the line, separated by commas:

Format	Item	
Format 1	#	
Format 2	Subcontext	Object class[,Object class...]

Format 1

This format is used to specify a comment. If a line starts with # (in the first column), any text between the # and the end of the line is assumed to be a comment.

Format 2

This format is used to specify the information shown below within a line.

Table 14-3: Information to be specified (definition file for adding or modifying user information)

Functionality	Meaning	Attribute
Subcontext	Specify the RDN relative to the user entry used for authentication. If this is omitted, the user entry is assumed.	Optional
Object class	Specify the object class of the subcontext. To specify two or more subcontexts, separate them by commas.	Required

14.4.4 Definition file for user mapping and authentication information

(1) Additional CSV file specifications

In addition to the specifications shown in *14.4.1 Basic CSV file specifications*, the following specifications are added for specifying lists of attributes:

- The first line contains header information, and the second and subsequent lines contain data to be registered.
- The type of information of each item is determined by the header.
- If two successive commas are specified, it is assumed that the option between the commas (`,`) is omitted.

(2) Specification formats

Information to be specified in the first line

Specify the header information shown in the table below. Each item must consist of ASCII characters and be separated by a comma (`,`). Item IDs can be specified in any order.

Table 14-4: Header information to be specified (definition file for user mapping and authentication information)

Item ID	Specification	Description	Attribute
`REALMNAME`	Register identification	Specify the name of a realm. User entries are created under the specified name.	Required
`USERID`		User ID	Required
`SECRETDATA`	Authentication information	Data is to be encrypted and saved.	Optional

Item ID	Specification	Description	Attribute
`PUBLICDATA`		Data is to be saved without encryption.	Optional
`LINK_`*xxxx*	Destination system user	Specify the name of a user of an application that has user management functionality. (*xxxx* must be a `REALMNAME`.)	Optional
`OPERATION`	Line operation command	Specify a line operation type. A file can contain specifications of addition, change, and deletion.	Optional

If the specified name is not an item ID, the field is ignored.

`LINK_`*xxxx* is an item ID created for each realm registered in the single sign-on repository.

Second and subsequent lines

Specify actual data to be registered, separating each item by a comma (`,`).

(3) User definition for applications that have JAAS-compatible user management

The *xxxx* portion of a `LINK_`*xxxx* item ID indicates the name of a realm representing an application that has user management functionality for the connection destination. To define the connection destination, specify a user ID in the *xxxx* field. To add to or modify the specification, use the following operations:

Add

This adds to the given `LINK_`*xxxx* item the user ID of an application (or realm) to which you wish to assign user management functionality for the connection destination.

Modify

This modifies the given `LINK_`*xxxx* item to the user ID of an application (or realm) to which you wish to assign user management functionality for the connection destination.

Delete

This deletes the user ID from the given `LINK_`*xxxx* item (or specifies nothing in it).

14.4.5 CSV file specification example

This example assumes users with the names *taro*, *hanako*, and *jirou* under a realm named *Portal*, and a user named *k010000* under a realm named *RealmA*. Under the realm name *J2EE*, the users *hanako* and *jirou* use the *Admin* and *DBMgr* user IDs respectively. In this case, the CSV file contains the information shown below. (This example assumes that the user ID and password of a user are the same.)

REALMNAME	USERID	SECRETDATA	PUBLICDATA	LINK_J2EE	LINK_REALMA
Portal	taro	taro	developer	--	k010000
Portal	hanako	hanako	--	Admin	k010000
Portal	jirou	jirou	--	DBMgr	k010000
RealmA	k010000	k010000	--	--	--
J2EE	Admin	Admin	--	--	--
J2EE	DBMgr	DBMgr	--	--	--

Specification example

```
REALMNAME,USERID,SECRETDATA,PUBLICDATA,LINK_J2EE,LINK_REALM
A
Portal,taro,taro,developer,,k010000
Portal,hanako,hanako,,Admin,k010000
Portal,jirou,jirou,,DBMgr,k010000
RealmA,k010000,k010000,,,
J2EE,Admin,Admin,,,
J2EE,DBMgr,DBMgr,,,
```

14.4.6 Line operation

Line operation is a type of functionality that provides line addition, modification, or deletion in the single sign-on information repository according to a value specified in the `OPERATION` field within a CSV file. The administrator can insert any number of `OPERATION` fields. To enable this functionality, the `-x` option must be specified in the `ssoimport` command. The `-x` option is not compatible with the `-a`, `-m`, and `-d` options.

The types and purposes of operations that can be specified in the `OPERATION` field are shown in the table below.

Table 14-5: Types and purposes of operations that can be specified in the `OPERATION` field

Operation	Purpose
`A` or `a`	Add the specified line to the single sign-on information repository. If the user entry already exists in the single sign-on information repository, the system issues a warning message and continues the processing.
`M` or `m`	Overwrite the single sign-on information repository with the specified line. If the user entry does not exist in the single sign-on information repository, the user entry is added.

Operation	Purpose
`D` or `d`	Deletes the specified line from the single sign-on information repository. If the user entry does not exist in the single sign-on information repository, the system issues a warning message and continues the processing.

Note

Operations are not case-sensitive. Any null character preceding or succeeding an operation is ignored. The tab character is invalid.

If a character specified in the operation field is not one of `A`, `a`, `M`, `m`, `D`, or `d`, the system issues a warning and skips the line. In this case, check to see whether the specified operation is correct and retry.

An example of a CSV file with the `OPERATION` field added is shown below.

OPERATION	REALMNAME	USERID	SECRETDATA	PUBLICDATA	LINK_J2EE
`A`	Portal	taro	taro	developer	--
`D`	Portal	hanako	hanako	--	Admin
`M`	Portal	jirou	jirou	--	DBMgr
--	RealmA	k010000	k010000	--	--
`M`	J2EE	Admin	Admin	--	--

Specification example

```
OPERATION,REALMNAME,USERID,SECRETDATA,PUBLICDATA,LINK_J2EE
A,Portal,taro,taro,developer,
D,Portal,hanako,hanako,,Admin
M,Portal,jirou,jirou,,DBMgr
,RealmA,k010000,k010000,,
M,J2EE,Admin,Admin,,
```

Chapter

15. APIs Used with the Integrated User Management Framework

This chapter describes the APIs and exception classes that are used with the integrated user management framework.

15.1 List of APIs for the integrated user management framework

APIs and exception classes are used when user authentication is implemented with the libraries for the integrated user management framework. The table below lists these APIs and exception classes.

Table 15-1: List of APIs and exception classes for the integrated user management framework

Class/interface name	Functionality	API type
`AttributeEntry` class	Manages attribute names paired with aliases.	User authentication library
`ChangeDataFailedException` class	Exception class called by the `SSODataListener` interface implementation class	Single sign-on library (exception class)
`DelegationLoginModule` class	JAAS login module implementation class. Calls a custom login module.	Cosminexus standard login module
`LdapSSODataManager` class	References or updates information in the single sign-on information repository on the LDAP directory server.	Single sign-on library
`LdapUserDataManager` class	References or updates information in the user information repository on the LDAP directory server.	User authentication library
`LdapUserEnumeration` interface	References a list of user IDs.	User authentication library
`LoginUtil` class	Checks for users who have logged into a session for integrated user management.	User authentication library
`ObjectClassEntry` class	Holds the object class of an entry on the LDAP directory server.	User authentication library
`PasswordCryptography` interface	Encrypts a password entered by a user.	User authentication library
`PasswordUtil` class	Changes a password entered by a user.	User authentication library
`Principal` interface	References the user ID authenticated by `WebPasswordLoginModule`.	User authentication library
`SSOData` class	Holds single sign-on authentication information.	Single sign-on library

Class/interface name	Functionality	API type
`SSODataEvent` class	Holds updated single sign-on authentication information.	Single sign-on library
`SSODataListener` interface	Reports the update of single sign-on authentication information.	Single sign-on library
`SSODataListenerException` class	Exception class that is called when an exception occurs in the authentication information listener class for single sign-on.	Single sign-on library (exception class)
`UserAttributes` interface	References the credentials that were created when `WebPasswordLoginModule` authenticated the user.	User authentication library
`UserData` class	Holds user information.	User authentication library
`WebCertificateCallback` class	JAAS `Callback implementation class`. Holds the results of SSL authentication performed by a Web server.	User authentication library
`WebCertificateHandler` class	JAAS `CallbackHandler` implementation class. Reads necessary information that indicates the results of SSL authentication performed by a Web server.	User authentication library
`WebCertificateLoginModule` class	JAAS login module implementation class. Obtains user attributes from certificates authenticated by a Web server.	Cosminexus standard login module
`WebLogoutCallback` class	JAAS `Callback implementation class`. Holds information about a user who logs out.	User authentication library
`WebLogoutHandler` class	JAAS `CallbackHandler` implementation class. Reads necessary information about a user who logs out.	User authentication library
`WebPasswordCallback` class	JAAS `Callback implementation class`. Holds authentication information such as a password.	User authentication library
`WebPasswordHandler` class	JAAS `CallbackHandler` implementation class. Reads necessary information for password authentication.	User authentication library
`WebPasswordJDBCLoginModule` class	JAAS login module implementation class. Accesses a database by using JDBC, and authenticates the password.	Cosminexus standard login module

Class/interface name	Functionality	API type
`WebPasswordLDAPLoginModule` class	JAAS login module implementation class. Performs authentication according to the results of binding to the LDAP directory server.	Cosminexus standard login module
`WebPasswordLoginModule` class	JAAS login module implementation class. Authenticates passwords for Web applications.	Cosminexus standard login module
`WebSSOCallback` class	JAAS `Callback implementation class` offered by the single sign-on library. Obtains information required for `WebSSOLoginModule`.	Single sign-on library
`WebSSOHandler` class	JAAS `CallbackHandler` implementation class offered by the single sign-on library. Reads information required for `WebSSOLoginModule`.	Single sign-on library
`WebSSOLoginModule` class	JAAS login module implementation class. Calls other login modules for single sign-on.	Cosminexus standard login module
`Exception` classes	API exception classes used for integrated user management.	Exception class

15.2 The AttributeEntry class

Description

Represents not only the name and alias of an attribute fetched from the user management repository, but also a tuple containing subcontexts from the user management context. After user authentication, the specified attribute is bound to the public credential of the subject via alias. If no alias is specified, it is bound to that public credential via attribute name.

The package name of this class is `com.cosminexus.admin.auth`.

Syntax

```
class AttributeEntry
{
public AttributeEntry(String attr,
                          String alias,
                          String subcontext);
public AttributeEntry(String attr,
                          String alias);
public AttributeEntry(String attr);
public AttributeEntry();

public String getAlias();
public String getAttributeName();
public String getSubcontext();
public void setAlias(String alias);
public void setAttributeName(String attr);
public void setSubcontext(String subcontext);
}
```

Constructor and methods

Constructor/method name	Functionality
`AttributeEntry` constructor	Creates an instance of the `AttributeEntry` class.
`getAlias` method	Obtains the alias specified by the `setAlias` method or constructor.
`getAttributeName` method	Obtains the attribute name specified by the `setAttributeName` method or constructor.
`getSubcontext` method	Obtains the subcontext specified by the `setSubcontext` method or constructor.
`setAlias` method	Stores the alias specified by the parameter into the object.
`setAttributeName` method	Stores the attribute name specified by the parameter into the object.

Constructor/method name	Functionality
`setSubcontext` method	Stores the subcontext specified by the parameter into the object.

The AttributeEntry constructor

Description

Creates an instance of the `AttributeEntry` class.

Syntax

```
public AttributeEntry(String attr,
                          String alias,
                          String subcontext);

public AttributeEntry(String attr,
                      String alias);

public AttributeEntry(String attr);

public AttributeEntry();
```

Parameters

`attr`:

Specifies the attribute name stored in the repository.

`alias`:

Specifies the alias associated with the attribute name.

`subcontext`:

Specifies the subcontext.

Exceptions

None

The getAlias method

Description

Obtains the value specified by the `setAlias` method or constructor. If that value does not exist, the `getAlias` method returns `null` when it is called.

Syntax

```
public String getAlias();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The getAttributeName method

Description

Obtains the value specified by the `setAttributeName` method or constructor. If that value does not exist, the `getAttributeName` method returns `null` when it is called.

Syntax

```
public String getAttributeName();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The getSubcontext method

Description

Obtains the value specified by the `setSubcontext` method or constructor. If that value does not exist, the `getSucontext` method returns `null` when it is called.

Syntax

```
public String getSubcontext();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The setAlias method

Description

Stores the value specified by the parameter into the object. If a value already exists, it is overwritten when the `setAlias` method is called.

Syntax

```
public void setAlias(String alias);
```

Parameter

`alias`:

Specifies the alias associated with the attribute name.

Exceptions

None

Return value

None

The setAttributeName method

Description

Stores the value specified by the parameter into the object. If a value already exists, it is overwritten when the `setAttributeName` method is called.

Syntax

```
public void setAttributeName(String attr);
```

Parameter

`attr`:

Specifies the attribute name.

Exceptions

None

Return value

None

The setSubcontext method

Description

Stores the value specified by the parameter into the object. If a value already exists, it is overwritten when the `setSubcontext` method is called.

Syntax

```
public void setSubcontext(String subcontext);
```

Parameter

`subcontext`:

Specifies the subcontext.

Exceptions

None

Return value

None

15.3 The ChangeDataFailedException class

Description

An exception class that is called when the `SSODataListener` interface implementation class fails at the time of adding, correcting, or deleting data.

The package name of the `ChangeDataFailedException` class is `com.cosminexus.admin.auth.api.repository.event`.

Syntax

```
class ChangeDataFailedException extends UAException
{
public ChangeDataFailedException();
public ChangeDataFailedException(String msg);
}
```

Constructor

Constructor name	Functionality
`ChangeDataFailedException` constructor	Creates an instance of the `ChangeDataFailedException` class.

The ChangeDataFailedException constructor

Description

Uses the error message specified by the parameter to create an instance of the `ChangeDataFailedException` class.

Syntax

```
public ChangeDataFailedException();

public ChangeDataFailedException(String msg);
```

Parameter

`msg`:

Specifies the error message.

Exceptions

None

15.4 The DelegationLoginModule class

Description

A JAAS login module implementation class offered by the user authentication library. It calls a custom login module. Its package name is `com.cosminexus.admin.auth.login`.

15.5 The LdapSSODataManager class

Description

References or updates information stored in the single sign-on information repository on the LDAP directory server.

The package name of the `LdapSSODataManager` class is `com.cosminexus.admin.auth.api.repository.ldap`.

Syntax

```
class LdapSSODataManager
{
  public LdapSSODataManager(String realm);

  public LdapUserEnumeration listUsers()
    throws NamingException;
  public LdapUserEnumeration listUsers(String uid)
    throws NamingException;
  public SSOData getSSOData(String uid)
    throws NamingException;
  public void addSSOData(String uid,
                            SSOData SSOData)
    throws SSODataListenerException, NamingException,
    CryptoException, UnsatisfiedLinkError,
SecurityException;
  public void removeSSOData(String uid)
    throws SSODataListenerException, NamingException,
    CryptoException, UnsatisfiedLinkError,
SecurityException;
  public void modifySSOData(String uid,
                               SSOData SSOData)
    throws SSODataListenerException, NamingException,
    CryptoException, UnsatisfiedLinkError,
SecurityException;
  public SSODataListener[] getSSODataListeners();
  public void addSSODataListener(SSODataListener listener);
  public void removeSSODataListener(SSODataListener
listener);
}
```

Constructor and methods

Constructor/method name	Functionality
`LdapSSODataManager` constructor	Creates an instance of the `LdapSSODataManager` class.
`addSSOData` method	Adds single sign-on authentication information.

Constructor/method name	Functionality
`addSSODataListener` method	Registers an authentication information listener for single sign-on.
`getSSOData` method	Obtains single sign-on authentication information.
`getSSODataListeners` method	Obtains an array of `SSODataListener` objects.
`listUsers` method (syntax 1)	Obtains a list of all user IDs.
`listUsers` method (syntax 2)	Obtains a list of user IDs.
`modifySSOData` method	Corrects single sign-on authentication information.
`removeSSOData` method	Deletes single sign-on authentication information.
`removeSSODataListener` method	Deletes the `SSODataListener` object.

The LdapSSODataManager constructor

Description

Creates an instance of the `LdapSSODataManager` class.

Syntax

```
public LdapSSODataManager(String realm);
```

Parameter

`realm`:

Specifies the name of a realm to be accessed by the created instance.

Exceptions

None

The addSSOData method

Description

Adds single sign-on authentication information for the specified user. If this information already exists, an exception occurs.

When authentication information listeners for single sign-on are registered in this object, the `ssoDataAdded` method is called for all such listeners.

Syntax

```
public void addSSOData(String uid,
                           SSOData SSOData)
```

```
    throws SSODataListenerException, NamingException,
    CryptoException, UnsatisfiedLinkError, SecurityException;
```

Parameters

`uid`:

Specifies the user ID.

`ssoData`:

Specifies the `SSOData` object that holds the single sign-on authentication information.

Exceptions

`com.cosminexus.admin.auth.api.repository.event.SSODataListenerException`:

An attempt to update authentication information for another system has failed.

`com.cosminexus.admin.auth.CryptoException`:

An attempt to read the encryption key file has failed or an attempt to decrypt SecretData has failed due to the use of the wrong encryption key file.

`java.lang.UnsatisfiedLinkError`:

An attempt to read the single sign-on library has failed.

`java.lang.SecurityException`:

SecurityManager is present, and read access to the file using SecurityManager's `checkRead` method has been rejected.

`javax.naming.CommunicationException`:

An attempt to connect to the LDAP directory server has failed.

`javax.naming.NameAlreadyBoundException`:

Single sign-on authentication information already exists for the specified user.

Other JNDI exceptions:

Events such as a bind DN specification error

Return value

None

The addSSODataListener method

Description

Registers an authentication information listener for single sign-on in this object. This is done to inform other systems of the changes made by adding, correcting, or deleting single sign-on authentication information.

Syntax

```
public void addSSODataListener(SSODataListener listener);
```

Parameter

`listener`:

Specifies the `SSODataListener` object. If `null` is specified, nothing happens.

Exceptions

None

Return value

None

The getSSOData method

Description

Obtains single sign-on authentication information.

Syntax

```
public SSOData getSSOData(String uid)
  throws NamingException;
```

Parameter

`uid`:

Specifies the user ID.

Exceptions

`javax.naming.CommunicationException`:

An attempt to connect to the LDAP directory server has failed.

`javax.naming.NameNotFoundException`:

The specified user ID is missing.

Other JNDI exceptions:

Events such as a bind DN specification error

Return value

The `SSOData` object that holds single sign-on authentication information

The getSSODataListeners method

Description

Obtains an array of `SSODataListener` objects registered in this object. If the array is not registered, this method returns an array of size 0.

Syntax

```
public SSODataListener[] getSSODataListeners();
```

Parameters

None

Exceptions

None

Return value

The array of `SSODataListener` objects registered in this object

The listUsers method (syntax 1)

Description

Obtains a list of all user IDs. If the `addSSOData` or `removeSSOData` method is called, the result might or might not be reflected in the previously returned `LdapUserEnumeration` object.

Syntax

```
public LdapUserEnumeration listUsers()
  throws NamingException;
```

Parameters

None

Exceptions

`javax.naming.CommunicationException`:

An attempt to connect to the LDAP directory server has failed.

Other JNDI exceptions:

Events such as a bind DN specification error

Return value

The `LdapUserEnumeration` object that holds a list of user IDs.

The listUsers method (syntax 2)

Description

Obtains a list of user IDs. If the `addSSOData` or `removeSSOData` method is called, the result might or might not be reflected in the previously returned `LdapUserEnumeration` object.

Syntax

```
public LdapUserEnumeration listUsers(String uid)
  throws NamingException;
```

Parameter

`uid`:

Specifies the user ID. The user ID can include a wildcard (`*`). If this parameter is omitted or if `null` is specified, the method obtains a list of all user IDs.

Exceptions

`javax.naming.CommunicationException`:

An attempt to connect to the LDAP directory server has failed.

Other JNDI exceptions:

Events such as a bind DN specification error

Return value

The `LdapUserEnumeration` object that holds a list of user IDs.

The modifySSOData method

Description

Corrects single sign-on authentication information. If the specified user does not exist, an exception occurs.

When authentication information listeners for single sign-on are registered in this object, the `ssoDataModfied` method is called for all such listeners.

With the `modifySSOData` method, existing information is overwritten with only the modified authentication information that is specified after creation of the `SSOData` object.

Suppose, for example, the existing single sign-on authentication information in the repository includes the elements shown below.

Authentication information name	SecretData	PublicData	Mapping	
			Realm	User ID
`Value`	`secret`	`public`	`RealmA`	`user1`
			`RealmB`	`admin`

In the parameter of this method, the following code is used to specify the created `SSOData` object:

```
SSOData data = new SSOData();
data.setMapping("RealmA", "user2");
```

The repository stores the following new single sign-on authentication information:

Authentication information name	SecretData	PublicData	Mapping	
			Realm	User ID
`Value`	`secret`	`public`	`RealmA`	`user2`
	--	--	--	--

Legend:

--: No information is stored.

Syntax

```
public void modifySSOData(String uid,
                                SSOData SSOData)
  throws SSODataListenerException, NamingException,
  CryptoException, UnsatisfiedLinkError, SecurityException;
```

Parameters

`uid`:

Specifies the user ID.

`ssoData`:

Specifies the `SSOData` object that holds the single sign-on authentication information.

Exceptions

`com.cosminexus.admin.auth.api.repository.event.SSODataListenerException`:

An attempt to update authentication information for another system has failed.

`com.cosminexus.admin.auth.CryptoException`:

An attempt to read the encryption key file has failed or an attempt to decrypt SecretData has failed due to the use of the wrong encryption key file.

`java.lang.UnsatisfiedLinkError`:

An attempt to read the single sign-on library has failed.

`java.lang.SecurityException`:

SecurityManager is present, and read access to the file using SecurityManager's `checkRead` method has been rejected.

`javax.naming.CommunicationException`:

An attempt to connect to the LDAP directory server has failed.

`javax.naming.NameNotFoundException`:

The specified user ID is missing.

Other JNDI exceptions:

Events such as a bind DN specification error

Return value

None

The removeSSOData method

Description

Deletes single sign-on authentication information. If the specified user does not exist, an exception occurs.

When authentication information listeners for single sign-on are registered in this object, the `ssoDataRemoved` method is called for all such listeners.

Syntax

```
public void removeSSOData(String uid)
  throws SSODataListenerException, NamingException,
  CryptoException, UnsatisfiedLinkError, SecurityException;
```

Parameter

`uid`:

Specifies the user ID.

Exceptions

`com.cosminexus.admin.auth.api.repository.event.SSODataListener Exception`:

An attempt to update authentication information for another system has failed.

`com.cosminexus.admin.auth.CryptoException`:

An attempt to read the encryption key file has failed or an attempt to decrypt SecretData has failed due to the use of the wrong encryption key file.

`java.lang.UnsatisfiedLinkError`:

An attempt to read the single sign-on library has failed.

`java.lang.SecurityException`:

SecurityManager is present, and read access to the file using SecurityManager's `checkRead` method has been rejected.

`javax.naming.CommunicationException`:

An attempt to connect to the LDAP directory server has failed.

`javax.naming.NameNotFoundException`:

The specified user ID is missing.

Other JNDI exceptions:

Events such as a bind DN specification error

Return value

None

The removeSSODataListener method

Description

Deletes the specified `SSODataListener` object from this object. If the specified object is not registered, nothing happens.

Syntax

```
public void removeSSODataListener(SSODataListener listener);
```

Parameter

`listener`:

Specifies the `SSODataListener` object.

Exceptions

None

Return value

None

15.6 The LdapUserDataManager class

Description

References or updates information in the user information repository on the LDAP directory server.

Exclusive control is provided for each object of this class, allowing only one of the `addUserData`, `modifyUserData`, `removeUserData`, and `getUserData` methods at a time to access that object.

Do not use the same repository for different objects at the same time.

The package name of the `LdapUserDataManager` class is `com.cosminexus.admin.auth.api.repository.ldap`.

Syntax

```
class LdapUserDataManager
{
  public LdapUserDataManager(String name)
    throws ConfigError;
  public LdapUserDataManager(String name,
                                 AttributeEntry[] aliases)
    throws ConfigError, FormatError;
  public LdapUserDataManager(String name,
                                 String aliasesFile)
    throws ConfigError, FormatError, IOException,
           FileNotFoundException,
    SecurityException;
  public LdapUserDataManager(String name,
                                 AttributeEntry[] aliases,
                             ObjectClassEntry[] ocEntries)
    throws ConfigError, FormatError;
  public LdapUserDataManager(String name,
                                 AttributeEntry[] aliases,
                                 String objclassesFile)
    throws ConfigError, FormatError, IOException,
           FileNotFoundException,SecurityException;
  public LdapUserDataManager(String name,
                                 String aliasesFile,
                             ObjectClassEntry[] ocEntries)
    throws ConfigError, FormatError, IOException,
           FileNotFoundException,SecurityException;
  public LdapUserDataManager(String name,
                                 String aliasesFile,
                                 String objclassesFile)
    throws ConfigError, FormatError, IOException,
           FileNotFoundException,SecurityException;
  public LdapUserEnumeration listUsers()
    throws NamingException;
  public LdapUserEnumeration listUsers(String uid)
    throws NamingException;
  public UserData getUserData(String uid)
    throws NamingException;
  public void addUserData(String uid,
                               UserData UserData)
    throws ObjectClassError, NamingException;
  public void addUserData(String uid,
                               UserData UserData,
                               String name, String value)
    throws ObjectClassError, NamingException;
  public void removeUserData(String uid)
    throws NamingException;
  public void modifyUserData(String uid, UserData UserData)
    throws ObjectClassError, NamingException;
}
```

Constructor and methods

Constructor/method name	Functionality
`LdapUserDataManager` constructor	Creates an instance of the `LdapUserDataManager` class.
`addUserData` method (syntax 1)	Adds a user. Uses uid as DN of a user entry.
`addUserData` method (syntax 2)	Adds a user. Uses an arbitrary attribute as DN of a user entry.
`getUserData` method	Obtains user information.
`listUsers` method (syntax 1)	Obtains a list of all user IDs.
`listUsers` method (syntax 2)	Obtains a list of user IDs.
`modifyUserData` method	Corrects user information.
`removeUserData` method	Deletes a user.

The LdapUserDataManager constructor

Description

Creates an instance of the `LdapUserDataManager` class. User attribute information and object classes can be specified or omitted in an object or file.

Syntax

```
public LdapUserDataManager(String name)
```

```
  throws ConfigError;

public LdapUserDataManager(String name,
                               AttributeEntry[] aliases)
  throws ConfigError, FormatError;

public LdapUserDataManager(String name,
                               String aliasesFile)
  throws ConfigError, FormatError, IOException,
FileNotFoundException,
  SecurityException;

public LdapUserDataManager(String name,
                               AttributeEntry[] aliases,
                               ObjectClassEntry[] ocEntries)
  throws ConfigError, FormatError;

public LdapUserDataManager(String name,
                               AttributeEntry[] aliases,
                               String objclassesFile)
  throws ConfigError, FormatError, IOException,
FileNotFoundException,
  SecurityException;

public LdapUserDataManager(String name,
                               String aliasesFile,
                               ObjectClassEntry[] ocEntries)
  throws ConfigError, FormatError, IOException,
FileNotFoundException,
  SecurityException;

public LdapUserDataManager(String name,
                               String aliasesFile,
                               String objclassesFile)
  throws ConfigError, FormatError, IOException,
FileNotFoundException,
  SecurityException;
```

Parameters

`name`:

Specifies the setup name of the LDAP directory server to be accessed. This name is defined in the configuration file for user management.

`aliases`:

Specifies the array of `AttributeEntry` objects as user attribute information to be referenced or updated. If the specified parameter lacks necessary information, a `FormatError` exception occurs. If this parameter is omitted or if `null` is specified, the attribute cannot be referenced or updated, but the password can be updated.

`aliasesFile`:

Specifies the file name as user attribute information to be referenced or updated. If the specified parameter lacks necessary information, a `FormatError` exception occurs. If this parameter is omitted or if `null` is specified, the attribute cannot be referenced or updated, but the password can be updated.

`ocEntries`:

Specifies the array of object classes to be used for creating or correcting entries on the LDAP directory server. If the specified parameter lacks necessary information, a `FormatError` exception occurs. If this parameter is omitted or if `null` is specified, an `ObjectClassError` exception occurs when user information is added or changed.

`objclassesFile`:

Specifies the name of the file that defines the object classes of entries on the LDAP directory server. If the specified parameter lacks necessary information, a `FormatError` exception occurs. If this parameter is omitted or if `null` is specified, an `ObjectClassError` exception occurs when user information is added or changed.

Exceptions

`java.io.FileNotFoundException`:

The specified file cannot be opened because it is missing or is a directory, or because of some other reason (when the exception is thrown in the constructor of the `FileInputStream` class).

`java.lang.SecurityException`:

SecurityManager is present, and read access to the file using SecurityManager's `checkRead` method has been rejected.

`java.io.IOException`:

An attempt to read the file has failed.

`com.cosminexus.admin.common.ConfigError`:

The setup name was not found in the configuration file for integrated user management.

`com.cosminexus.admin.common.FormatError`:

One or more of the `aliases`, `aliasesFile`, `ocEntries`, and `objclassesFile` parameters lacks necessary information or contains extra information.

The addUserData method (syntax 1)

Description

Adds a user. If the user already exists, an exception occurs.

The attribute (`uid`) and value of the user ID are used for the DN of a user entry created on the LDAP directory server.

The user entry is created immediately below the base DN. If the user attribute information specified by the constructor includes the attributes of a subcontext, an entry for the subcontext is also created.

If an exception occurs during the subcontext update after this method is called, user information is incompletely updated. In such case, remove the cause and use the `removeUserData` method to delete the user, and then call this method again.

Syntax

```
public void addUserData(String uid,
                            UserData UserData)
  throws ObjectClassError, NamingException;
```

Parameters

`uid`:

Specifies the user ID.

`userData`:

Specifies the `UserData` object that holds the user information.

Exceptions

`com.cosminexus.admin.auth.api.repository.ldap.ObjectClassError`:

An object class necessary for creating an entry on the LDAP directory server is not specified.

`javax.naming.CommunicationException`:

An attempt to connect to the LDAP directory server has failed.

`javax.naming.NameAlreadyBoundException`:

The specified user ID already exists.

Other JNDI exceptions:

Events such as a bind DN specification error

Return value

None

Remarks

`UserData` objects acquired using the `getUserData` method do not contain passwords. If you specify such a `UserData` object in the parameters of the `addUserData` method, you cannot make a complete copy of the user. You need to newly configure a password.

The addUserData method (syntax 2)

Description

Adds a user. If the user already exits, an exception occurs.

The attribute name and value specified by this method are used for the DN of a user entry created on the LDAP directory server.

The user entry is created immediately below the base DN. If the user attribute information specified by the constructor includes the attributes of a subcontext, an entry for the subcontext is also created.

If an exception occurs during the subcontext update after this method is called, user information is incompletely updated. In such case, remove the cause and use the `removeUserData` method to delete the user, and then call this method again.

Syntax

```
public void addUserData(String uid,
                             UserData UserData,
                             String name,
                             String value)
  throws ObjectClassError, NamingException;
```

Parameters

`uid`:

Specifies the user ID.

`userData`:

Specifies the `UserData` object that holds the user information.

`name`:

Specifies the attribute name to be used for the DN of the user entry.

`value`:

Specifies the attribute value to be used for the DN of the user entry.

Exceptions

`com.cosminexus.admin.auth.api.repository.ldap.ObjectClassError`:

An object class necessary for creating an entry on the LDAP directory server is not specified.

`javax.naming.CommunicationException`:

An attempt to connect to the LDAP directory server has failed.

`javax.naming.NameAlreadyBoundException`:

The specified user ID already exists.

Other JNDI exceptions:

Events such as a bind DN specification error

Return value

None

Remarks

`UserData` objects acquired using the `getUserData` method do not contain passwords. If you specify such a `UserData` object in the parameters of the `addUserData` method, you cannot make a complete copy of the user. You need to newly configure a password.

The getUserData method

Description

Obtains user information. The acquired `UserData` object does not contain a password.

Syntax

```
public UserData getUserData(String uid)
  throws NamingException;
```

Parameter

`uid`:

Specifies the user ID.

Exceptions

`javax.naming.CommunicationException`:

An attempt to connect to the LDAP directory server has failed.

`javax.naming.NameNotFoundException`:

The specified user ID is missing.

Other JNDI exceptions:

Events such as a bind DN specification error

Return value

The `UserData` object that holds the user information

The listUsers method (syntax 1)

Description

Obtains a list of all user IDs. If the `addUserData` or `removeUserData` method is called, the result might or might not be reflected in the previously returned `LdapUserEnumeration` object.

Syntax

```
public LdapUserEnumeration listUsers()
  throws NamingException;
```

Parameters

None

Exceptions

`javax.naming.CommunicationException`:

An attempt to connect to the LDAP directory server has failed.

Other JNDI exceptions:

Events such as a bind DN specification error

Return value

The `LdapUserEnumeration` object that holds the list of user IDs

The listUsers method (syntax 2)

Description

Obtains a list of user IDs. If the `addUserData` or `removeUserData` method is called, the result might or might not be reflected in the previously returned `LdapUserEnumeration` object.

Syntax

```
public LdapUserEnumeration listUsers(String uid)
  throws NamingException;
```

Parameter

`uid`:

Specifies the user ID. The user ID can include a wildcard (`*`). If this parameter is omitted or if `null` is specified, this method obtains a list of all user IDs.

Exceptions

`javax.naming.CommunicationException`:

An attempt to connect to the LDAP directory server has failed.

Other JNDI exceptions:

Events such as a bind DN specification error

Return value

The `LdapUserEnumeration` object that holds the list of user IDs

The modifyUserData method

Description

Corrects user information. If the specified user does not exist, an exception occurs.

With this method, existing attributes are overwritten with only the modified attributes that are specified after creation of the `UserData` object.

Suppose, for example, the existing user information in the repository includes the attributes shown below.

Attribute name	Full name	Tel
Value	Hitachi Taro	111-1111
		222-2222

In the parameter of this method, the following code is used to specify the created `UserData` object:

```
UserData data = new UserData();
data.addAttribute("tel", "111-2222");
```

The repository stores the following new user information:

Attribute name	Full name	Tel
`Value`	`Hitachi Taro`	`111-2222`
		--

Legend:

--: No information is stored.

If an exception occurs during the subcontext update after this method is called, user information is incompletely updated. In this case, remove the cause and then call this method again.

Syntax

```
public void modifyUserData(String uid,
                                UserData UserData)
  throws ObjectClassError, NamingException;
```

Parameters

`uid`:

Specifies the user ID.

`userData`:

Specifies the `UserData` object that holds the user information.

Exceptions

`com.cosminexus.admin.auth.api.repository.ldap.ObjectClassError`:

The object class necessary for creating an entry on the LDAP directory server is not specified.

`javax.naming.CommunicationException`:

An attempt to connect to the LDAP directory server has failed.

`javax.naming.NameNotFoundException`:

The specified user ID is missing.

Other JNDI exceptions:

Events such as a bind DN specification error

Return value

None

The removeUserData method

Description

Deletes a user. If the specified user does not exist, an exception occurs. The specified user entry and any entries below it are deleted on the LDAP directory server.

If an exception occurs during the subcontext update after this method is called, user information is incompletely updated. In this case, remove the cause and then call this method again.

Syntax

```
public void removeUserData(String uid)
  throws NamingException;
```

Parameter

`uid`:

Specifies the user ID.

Exceptions

`javax.naming.CommunicationException`:

An attempt to connect to the LDAP directory server has failed.

`javax.naming.NameNotFoundException`:

The specified user ID is missing.

Other JNDI exceptions:

Events such as a bind DN specification error

Return value

None

15.7 The LdapUserEnumeration interface

Description

References a list of user IDs.

The package name of the `LdapUserEnumeration` interface is `com.cosminexus.admin.auth.api.repository.ldap`.

Syntax

```
interface LdapUserEnumeration extends java.util.Enumeration
{
  public boolean hasMore()
    throws NamingException;
  public boolean hasMoreElements();
  public String next()
    throws NamingException;
  public Object nextElement();
  public close()
    throws NamingException;
}
```

Methods

Method name	Functionality
`close` method	Closes an object.
`hasMore` method	Checks whether the list contains any more user IDs. (`NamingException`: Called)
`hasMoreElements` method	Checks whether the list contains any more user IDs. (`NamingException`: Not called)
`next` method	Obtains the next user ID from the list. (`NamingException`: Called; Return value type: string)
`nextElement` method	Obtains the next user ID from the list. (`NamingException`: Not called; Return value type: object)

The close method

Description

Closes this object and releases the resources in use. If the `hasMore` or `hasMoreElements` method is repeatedly called until the return of `false`, calling this method is unnecessary.

Syntax

```
public void close()
  throws NamingException;
```

Parameters

None

Exception

`javax.naming.NamingException`:

A `NamingException` has occurred while the object is closed.

Return value

None

The hasMore method

Description

Checks whether the list contains any more user IDs.

Syntax

```
public boolean hasMore()
  throws NamingException;
```

Parameters

None

Exception

`javax.naming.NamingException`:

A NamingException has occurred while the method is determing whether there are any more user IDs.

Return values

`true`:

Another user ID has been found.

`false`:

No more user IDs have been found.

The hasMoreElements method

Description

Checks whether the list contains any more user IDs. If an exception occurs, this method returns `false`.

Syntax

```
public boolean hasMoreElements();
```

Parameters

None

Exceptions

None

Return values

`true`:

Another user ID has been found.

`false`:

No more user IDs have been found.

The next method

Description

Obtains the next user ID from the list.

Syntax

```
public String next()
  throws NamingException;
```

Parameters

None

Exceptions

`java.util.NoSuchElementException`:

There were no more user IDs in the list when this method was called.

`javax.naming.NamingException`:

A NamingException occurred while the method was being executed to obtain the next user ID.

Return value

The next user ID

The nextElement method

Description

Obtains the next user ID from the list. This method differs from the `next` method in that it does not cause a `NamingException` and that it returns an object-type value. Reference the value by casting its `Object` object into a string. If `NamingException` occurs during the execution of this method, it returns `null`.

Syntax

```
public Object nextElement();
```

Parameters

None

Exception

`java.util.NoSuchElementException`:

There were no more user IDs in the list when this method was called.

Return value

The next user ID

15.8 The LoginUtil class

Description

Checks for a user who has logged into a session for integrated user management.

The package name of the `LoginUtil` class is `com.cosminexus.admin.auth.util`.

Syntax

```
class LoginUtil
{
  public static boolean check(HttpServletRequest request,
                              HttpServletResponse response);
  public static boolean check(HttpServletRequest request,
                               HttpServletResponse response,
                                    String realmName);
}
```

Methods

Method name	Functionality
`check` method (syntax 1)	Checks for a user who has logged into a session.
`check` method (syntax 2)	Checks for a user who has logged into a session. Available for checking for a user who has logged into a particular realm.

Remarks

If you bind the subject created at login to `HttpSession`, then, based on whether the subject has a `Principal`, you can determine whether there is a currently logged-in user without using the `check` method of this class. When doing this, you should not stop the session by using the integrated user management functionality.

The check method (syntax 1)

Description

Checks for a user who has logged into a session. If at least one user is found who has logged into a realm within the session, this method returns `true`.

Syntax

```
public static boolean check(HttpServletRequest request,
                            HttpServletResponse response);
```

Parameters

`request`:

Specifies the reference to `HttpServletRequest` that was passed to JSP/ Servlet. If `null` is specified, a `NullPointerException` occurs.

`response`:

Specifies the reference to `HttpServletResponse` that was passed to JSP/ Servlet. If `null` is specified, a `NullPointerException` occurs.

Exception

`java.lang.NullPointerException`:

One of the parameters for this method has been specified as `null`.

Return values

`true`:

A currently logged-in user has been found.

`false`:

A currently logged-in user has not been found.

The check method (syntax 2)

Description

Checks for a user who has logged into a session. To check for a user who has logged into a particular realm, `realmName` is used.

Syntax

```
public static boolean check(HttpServletRequest request,
                                          HttpServletResponse response,
                                          String realmName);
```

Parameters

`request`:

Specifies the reference to `HttpServletRequest` that was passed to JSP/ Servlet. If `null` is specified, `NullPointerException` occurs.

`response`:

Specifies the reference to `HttpServletResponse` that was passed to JSP/ Servlet. If `null` is specified, `NullPointerException` occurs.

`realmName`:

Used to check for a user who has logged into a particular realm. If `null` is specified, `NullPointerException` occurs.

Exception

`java.lang.NullPointerException`:

One of the parameters for this method has been specified as `null`.

Return values

`true`:

A currently logged-in user has been found.

`false`:

A currently logged-in user has not been found.

15.9 The ObjectClassEntry class

Description

Holds the object class of a user entry or subcontext to be created on the LDAP directory server.

The package name of the `ObjectClassEntry` class is `com.cosminexus.admin.auth.api.repository.ldap`.

Syntax

```
class ObjectClassEntry
{
public ObjectClassEntry();
public ObjectClassEntry(String[] objectClasses);
public ObjectClassEntry(String subcontext,
                            String[] objectClasses);

public void setObjectClasses(String[] objectClasses);
public String[] getObjectClasses();
public void setSubcontext(String subcontext);
public String getSubcontext();
}
```

Constructor and methods

Constructor/method name	Functionality
`ObjectClassEntry` constructor	Creates an instance of the `ObjectClassEntry` class.
`getObjectClasses` method	Obtains the object class specified by the `setObjectClasses` method or constructor.
`getSubcontext` method	Obtains the subcontext specified by the `setSubcontext` method or constructor.
`setObjectClasses` method	Stores an object class into an object.
`setSubcontext` method	Stores a subcontext into an object.

The ObjectClassEntry constructor

Description

Creates an instance. If you specify an object class in the parameter, it is stored into this object as the object class of the user entry.

Syntax

```
public ObjectClassEntry();

public ObjectClassEntry(String[] objectClasses);

public ObjectClassEntry(String subcontext,
                                String[] objectClasses);
```

Parameters

`subcontext`:

Specifies the subcontext. If this parameter is omitted or if either `null` or a null character (`""`) is specified, the user entry is used.

Specify a character string that is the same as the subcontext specified for the `AttributeEntry` object.

`objectClasses`:

Specifies the object class of the user entry by using an array of strings. If this parameter is omitted or if `null` is specified, nothing is stored.

Exceptions

None

The getObjectClasses method

Description

Obtains the value specified by the `setObjectClasses` method or constructor. If that value does not exist, the `getObjectClasses` method returns `null` when it is called.

Syntax

```
public String[] getObjectClasses();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The getSubcontext method

Description

Obtains the value specified by the `setSubcontext` method or constructor. If that value does not exist, the `getSubcontext` method returns `null` when it is called.

Syntax

```
public String getSubcontext();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The setObjectClasses method

Description

Stores an object class into this object. If a value already exists when the `setObjectClasses` method is called, it is overwritten.

Syntax

```
public void setObjectClasses(String[] objectClasses);
```

Parameter

`objectClasses`:

Specifies the object class by using an array of strings.

Exceptions

None

Return value

None

The setSubcontext method

Description

Stores a subcontext into this object. If a value already exists when the `setSubcontext` method is called, it is overwritten.

Syntax

```
public void setSubcontext(String subcontext);
```

Parameters

`subcontext`:

Specifies the subcontext. If `null` or a null character (`""`) is specified, the user entry is used.

Specify a character string that is the same as the subcontext specified for the `AttributeEntry` object.

Exceptions

None

Return value

None

15.10 The PasswordCryptography interface

Description

Encrypts an input password.

The package name of the `PasswordCryptograpy` interface is `com.cosminexus.admin.auth.security`.

Syntax

```
interface PasswordCryptography
{
  public byte[] encrypt(byte[] plain);
}
```

Method

Method name	Functionality
`encrypt` method	Uses the encryption format registered in the repository to encrypt a password.

The encrypt method

Description

Uses the encryption format registered in the repository to encrypt a password.

Syntax

```
public byte[] encrypt(byte[] plain);
```

Parameter

`plain`:

Specifies the user-defined password (plain) that is stored when the login module calls this method.

Exceptions

None

Return value

The encrypted result

15.11 The PasswordUtil class

Description

Changes a user's password.

The package name of the `PasswordUtil` class is `com.cosminexus.admin.auth.util`.

Syntax

```
class PasswordUtil
{
  public static void changePassword(String name,
                                          String uid,
                                        String oldPassword,
                                        String newPassword)
    throws LoginException,
            SecurityException;

}
```

Method

Method name	Functionality
`changePassword` method	Changes a password.

The changePassword method

Description

Changes the password of a user who is correctly authenticated with the specified `name`, `uid`, and `oldPassword` parameters. When single sign-on authentication information is registered, the contents of the single sign-on information repository are also changed.

This method is a static method.

Syntax

```
public static void changePassword(String name,
                                  String uid,
                                  String oldPassword,
                                  String newPassword)
    throws LoginException,
      SecurityException;
```

Parameters:

`name`:

Specifies the application name (`name`) of the login module (`LoginContext`) used for authentication.

`uid`:

Specifies the user ID to be changed.

`oldPassword`:

Specifies the old password.

`newPassword`:

Specifies the new password.

Exceptions

`javax.security.auth.login.LoginException`:

Information necessary for authentication is missing or the user ID or password is wrong.

`java.lang.SecurityException`:

Access permission is not granted.

Return value

None

Remarks

- If the realm name, encryption key file, and information for access to the single sign-on information repository are all defined, single sign-on authentication information is changed. If not (that is, if definitions for single sign-on are not provided), that authentication information is not changed.
- If an exception (`NamingException`) occurs in the repository or if encryption fails while single sign-on authentication information is registered, then this method fails with `LoginException`. In this case, the password change is rolled back. If this rollback fails, `LoginException` occurs. For details about the exception class, see *15.32 Exception classes*.
- If an application specified by `name` does not use `WebPasswordLoginModule` or `WebPasswordLDAPLoginModule`, `LoginException` occurs.
- If the LDAP directory server is Active Directory, specify by `name` applications that use `WebPasswordLDAPLoginModule`.

15.12 The Principal interface

Description

References the user ID authenticated by `WebPasswordLoginModule`. With Hitachi's implementation classes, anything created by inheriting the `java.security.Principal` interface can be bound to an authenticated subject. Therefore, when referencing a `Principal`, first obtain it from the subject and then use the `getName` method to reference it. The package name of the `Principal` interface is `java.security`.

15.13 The SSOData class

Description

Holds single sign-on authentication information.

The package name of the `SSOData` class is `com.cosminexus.admin.auth.api.repository.ldap`.

Syntax

```
class SSOData
{
  public SSOData();

  public void setSecretData(String secretData)
    throws CryptoException, UnsatisfiedLinkError,
SecurityException;
  public void setPublicData(String publicData);
  public String getPublicData();
  public Enumeration getMappingRealms();
  public String getMapping(String realm);
  public void setMapping(String realm,
                             String uid);
  public void removeMapping(String realm);
}
```

Constructor and methods

Constructor/method name	Functionality
`SSOData` constructor	Creates an instance of the `SSOData` class.
`getMapping` method	Obtains a user ID associated with a realm name.
`getMappingRealms` method	Obtains a list of realms.
`getPublicData` method	Obtains PublicData.
`removeMapping` method	Deletes the specified realm.
`setMapping` method	Holds the name of a destination realm and user ID.
`setPublicData` method	Holds PublicData.
`setSecretData` method	Holds SecretData.

The SSOData constructor

Description

Creates an instance of the `SSOData` class.

Syntax

```
public SSOData();
```

Parameters

None

Exceptions

None

The getMapping method

Description

Obtains a user ID associated with the realm name from the mapping information stored in this object. If the user ID for the specified realm name does not exist, this method returns `null`.

Syntax

```
public String getMapping(String realm);
```

Parameter

`realm`:

Specifies the name of the destination realm.

Exceptions

None

Return value

The user ID

The getMappingRealms method

Description

Obtains a list of realm names from the mapping information stored in this object.To reference a realm name, get the `Object` object by executing the `nextElement` method for the `Enumeration` object obtained using this method. Reference the value obtained by casting the `Object` object into a string.

Syntax

```
public Enumeration getMappingRealms();
```

Parameters

None

Exceptions

None

Return value

The `Enumeration` object that holds the list of realm names

The getPublicData method

Description

Obtains PublicData stored in the object. If the value does not exist, the `getPublicData` method returns `null` when it is called.

Syntax

```
public String getPublicData();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The removeMapping method

Description

Removes the specified realm from the mapping information stored in this object. If the specified realm name does not exist, nothing happens.

Syntax

```
public void removeMapping(String realm);
```

Parameter

`realm`:

Specifies the destination realm name.

Exceptions

None

Return value

None

The setMapping method

Description

Stores the destination realm name and user ID specified by the parameters into this object. If a user ID for the same realm name already exists, the user ID is overwritten.

Syntax

```
public void setMapping(String realm,
                                String uid);
```

Parameters

`realm`:

Specifies the destination realm name.

`uid`:

Specifies the user ID for the destination realm.

Exceptions

None

Return value

None

The setPublicData method

Description

Stores PublicData specified by the parameter into this object. If a value already exists, it is overwritten.

Syntax

```
public void setPublicData(String publicData);
```

Parameter

`publicData`:

Specifies PublicData.

Exceptions

None

Return value

None

The setSecretData method

Description

Encrypts SecretData and stores it into this object. If SecretData already exists, it is overwritten.

Syntax

```
public void setSecretData(String secretData);
```

Parameter

`secretData`:

Specifies SecretData.

Exceptions

`com.cosminexus.admin.auth.CryptoException`:

An attempt to encrypt SecretData has failed because the encryption key file cannot be read.

`java.lang.UnsatisfiedLinkError`:

An attempt to read the single sign-on library has failed.

`java.lang.SecurityException`:

SecurityManager is present, and read access to the file using SecurityManager's `checkRead` method has been rejected.

Return value

None

15.14 The SSODataEvent class

Description

Stores updated single sign-on authentication information.

The package name of the `SSODataEvent` class is `com.cosminexus.admin.auth.api.repository.event`.

Syntax

```
class SSODataEvent
{
  public SSODataEvent(String uid,
                      String secretData,
                      String publicData,
                      String oldSecretData,
                      String oldPublicData);

  public String getUserId();
  public String getSecretData();
  public String getPublicData();
  public String getOldSecretData();
  public String getOldPublicData();
}
```

Constructor and methods

Constructor/method name	Functionality
`SSODataEvent` constructor	Creates an instance of the `SSODataEvent` class.
`getOldPublicData` method	Obtains old PublicData.
`getOldSecretData` method	Obtains old SecretData.
`getPublicData` method	Obtains PublicData.
`getSecretData` method	Obtains SecretData.
`getUserId` method	Obtains a user ID.

The SSODataEvent constructor

Description

Creates an instance. Stores the user ID, SecretData, PublicData, old SecretData, and old PublicData specified in the parameters.

Syntax

```
public SSODataEvent(String uid,
                    String secretData,
                    String publicData,
                    String oldSecretData,
                    String oldPublicData);
```

Parameters

`uid`:

Specifies the user ID.

`secretData`:

Specifies SecretData.

`publicData`:

Specifies PublicData.

`oldSecretData`:

Specifies old SecretData.

`oldPublicData`:

Specifies old PublicData.

Exceptions

None

The getOldPublicData method

Description

Obtains old PublicData stored in this object.

Syntax

```
public String getOldPublicData();
```

Parameters

None

Exceptions

None

Return value

This method returns old PublicData if it is specified. If it is not specified, it returns `null`.

The getOldSecretData method

Description

Obtains old SecretData stored in this object.

Syntax

```
public String getOldSecretData();
```

Parameters

None

Exceptions

None

Return value

This method returns old SecretData if it is specified. If it is not specified, it returns `null`.

The getPublicData method

Description

Obtains PublicData stored in this object.

Syntax

```
public String getPublicData();
```

Parameters

None

Exceptions

None

Return value

This method returns PublicData if it is specified. If it is not specified, it returns `null`.

The getSecretData method

Description

Obtains SecretData stored in this object.

Syntax

```
public String getSecretData();
```

Parameters

None

Exceptions

None

Return value

This method returns SecretData if it is specified. If it is not specified, it returns `null`.

The getUserId method

Description

Obtains the user ID stored in this object.

Syntax

```
public String getUserId();
```

Parameters

None

Exceptions

None

Return value

The user ID

15.15 The SSODataListener interface

Description

This interface should be implemented in the authentication information listener class for single sign-on, which receives notifications when single sign-on authentication information is updated.

If you want to update authentication information for other systems simultaneously by using single sign-on authentication information, create a class that implements this interface. Also, use the `addSSODataListener` method to register an instance (object) of the created class in the `LdapSSODataManager` object.

The package name of the `SSODataListener` interface is `com.cosminexus.admin.auth.api.repository.event`.

The methods of the `SSODataListener` interface can be called by using the methods of the `LdapSSODataManager` class. In such cases, `SSODataEvent` objects are passed as parameters.

The table below lists the calling methods of the `LdapSSODataManager` class and the called methods of the `SSODataListener` interface. It also lists the values stored in `SSODataEvent` objects passed as parameters.

Table 15-2: Values stored in SSODataEvent objects

Calling method of the LdapSSODataManager class	Called method of the SSODataListener interface	Values stored in the SSODataEvent object				
		User ID	SecretData	PublicData	Old SecretData	Old PublicData
`addSSOData` method	`ssoDataAdded` method	Yes	Yes	Yes	--	--
`modifySSOData` method	`ssoDataModified` method	Yes	Yes	Yes	Yes	Yes
`removeSSOData` method	`ssoDataRemoved` method	Yes	Yes	Yes	--	--

Legend:

Yes: Stored.

--: Not stored.

Create a class to throw a `ChangeDataFailedException` that contains a message indicating the cause of a problem with the `ssoDataAdded`, `ssoDataModified`, or `ssoDataRemoved` method. An `SSODataListenerException` that contains the exception object occurs in the caller of the `LdapSSODataManager` method.

Syntax

```
interface SSODataListener extends java.util.EventListener
{
  public void SSODataAdded(SSODataEvent event)
    throws ChangeDataFailedException;
  public void SSODataModified(SSODataEvent event)
    throws ChangeDataFailedException;
  public void SSODataRemoved(SSODataEvent event)
    throws ChangeDataFailedException;
}
```

Methods

Method name	Functionality
`ssoDataAdded` method	Called when adding single sign-on authentication information.
`ssoDataModified` method	Called when changing single sign-on authentication information.
`ssoDataRemoved` method	Called when deleting single sign-on authentication information.

The ssoDataAdded method

Description

Called when adding single sign-on authentication information.

Syntax

```
public void SSODataAdded(SSODataEvent event)
  throws ChangeDataFailedException;
```

Parameter

`event`:

Stores the single sign-on authentication information.

Exception

`com.cosminexus.admin.auth.api.repository.event.ChangeDataFailedException`:

An attempt to update authentication information for another system has failed.

Return value

None

The ssoDataModified method

Description

Called when changing single sign-on authentication information.

Syntax

```
public void SSODataModified(SSODataEvent event)
  throws ChangeDataFailedException;
```

Parameter

`event`:

Stores the single sign-on authentication information.

Exception

`com.cosminexus.admin.auth.api.repository.event.ChangeDataFailedException`:

An attempt to update authentication information for another system has failed.

Return value

None

The ssoDataRemoved method

Description

Called when deleting single sign-on authentication information.

Syntax

```
public void SSODataRemoved(SSODataEvent event)
  throws ChangeDataFailedException;
```

Parameter

`event`:

Stores the single sign-on authentication information.

Exception

`com.cosminexus.admin.auth.api.repository.event.ChangeDataFailedException`:

An attempt to update authentication information for another system has failed.

Return value

None

15.16 The SSODataListenerException class

Description

An exception class that is called when an exception occurs in the authentication information listener class for single sign-on.

The package name of the `SSODataListenerException` class is `com.cosminexus.admin.auth.api.repository.event`.

Syntax

```
class SSODataListenerException extends UAException
{
  public SSODataListenerException();
  public SSODataListenerException(String msg);

  public void setException(SSODataListener listener,
                    ChangeDataFailedException exception);
  public SSODataListener[] getListeners();
  public ChangeDataFailedException
getException(SSODataListener listener);
}
```

Constructor and methods

Constructor/method name	Functionality
`SSODataListenerException` constructor	Creates an instance of the `SSODataListenerException` class.
`getException` method	Obtains an exception object.
`getListeners` method	Obtains listeners stored in the exception.
`setException` method	Holds an exception object.

The SSODataListenerException constructor

Description

Creates an instance of the `SSODataListenerException` class by using the error message specified by the parameter.

Syntax

```
public SSODataListenerException();

public SSODataListenerException(String msg);
```

Parameter

`msg`:

Specifies the error message.

Exceptions

None

The getException method

Description

Obtains an exception object stored in this object. If the specified listener does not contain an exception object, this method returns `null`.

Syntax

```
public ChangeDataFailedException getException(SSODataListener
listener);
```

Parameter

`listener`:

Specifies the listener object where an exception occurred.

Exceptions

None

Return value

The `ChangeDataFailedException` object

The getListeners method

Description

Obtains all of the listeners stored in this object.

Syntax

```
public SSODataListener[] getListeners();
```

Parameters

None

Exceptions

None

Return value

The array of listener objects

The setException method

Description

Stores the exception object that caused an exception into this object. If an exception already exists for the same listener, it is overwritten.

Syntax

```
public void setException(SSODataListener listener,
                          ChangeDataFailedException exception);
```

Parameters

`listener`:

Specifies the listener object where an exception occurred.

`exception`:

Specifies the `ChangeDataFailedException` object that was generated in the listener.

Exceptions

None

Return value

None

15.17 The UserAttributes interface

Description

After user authentication, obtains attributes bound to the subject.

The package name of the `UserAttributes` interface is `com.cosminexus.admin.auth`.

Syntax

```
interface UserAttributes
{
  public Object getAttribute(String alias)
    throws IllegalStateException;
  public Enumeration getAttributes(String alias)
    throws IllegalStateException;
  public void addAttribute(String alias,
                               Object attr)
    throws IllegalStateException;
  public Enumeration getAttributeNames()
    throws IllegalStateException;
  public void removeAttribute(String alias)
    throws IllegalStateException;
  public int size()
    throws IllegalStateException;
  public Enumeration getAliases()
    throws IllegalStateException;
}
```

Methods

Method name	Functionality
`addAttribute` method	Adds an attribute to the subject.
`getAttribute` method	Obtains an attribute bound to the subject.
`getAttributeNames` method	Obtains a list of attribute names bound to the subject.
`getAttributes` method	Obtains all of the attributes bound to the subject.
`removeAttribute` method	Deletes an attribute bound to the subject.
`size` method	Obtains the total number of attributes bound to the subject.
`getAliases` method	Not recommended. Use the `getAttributeNames` method.

Remarks

If this object is invalid, calling a method causes `java.lang.IllegalStateException`. Note that this exception inherits from `java.lang.RuntimeException` and can therefore be compiled without being defined in `catch` or `throws`.

The addAttribute method

Description

Adds an attribute to the subject. Two or more attribute values can be associated with the same attribute. Attributes added to a subject with this method are not reflected in the user management repository.

Syntax

```
public void addAttribute(String alias,
                         Object attr)
  throws IllegalStateException;
```

Parameters

`alias`:

Specifies the attribute name to be bound to the subject.

`attr`:

Specifies the attribute value to be bound to the subject.

Exception

`java.lang.IllegalStateException`:

If the object is invalid, `java.lang.IllegalStateException` occurs. Note that this exception inherits from `java.lang.RuntimeException` and can therefore be compiled without being defined in `catch` or `throws`.

This exception occurs when both of the following conditions coexist:

- The subject that has this object is read-only.
- The logout process is performed using the `logout` method.

Return value

None

The getAttribute method

Description

Obtains an attribute bound to the subject. The requestor casts the returned object to reference the value. If the same attribute has two or more values, this method returns the first object found.

Syntax

```
public Object getAttribute(String alias)
  throws IllegalStateException;
```

Parameter

`alias`:

Specifies the attribute name bound to the subject. If the alias of the attribute is specified in the `AttributeEntry` class, specify that alias.

Exception

`java.lang.IllegalStateException`:

If the object is invalid, `java.lang.IllegalStateException` occurs. Note that this exception inherits from `java.lang.RuntimeException` and can therefore be compiled without being defined in `catch` or `throws`.

This exception occurs when both of the following conditions coexist:

- The subject that has this object is read-only.
- The logout process is performed using the `logout` method.

Return value

This method returns the attribute value bound to the subject if that value is found. If the value is not found, it returns `null`.

The getAttributeNames method

Description

Obtains a list of attribute names bound to the subject. If the alias of the attribute is specified in the AttributeEntry class, this method returns that alias.

Syntax

```
public Enumeration getAttributeNames()
  throws IllegalStateException;
```

Parameters

None

Exception

`java.lang.IllegalStateException`:

If the object is invalid, `java.lang.IllegalStateException` occurs. Note that this exception inherits from `java.lang.RuntimeException` and can therefore be compiled without being defined in `catch` or `throws`.

This exception occurs when both of the following conditions coexist:

- The subject that has this object is read-only.
- The logout process is performed using the `logout` method.

Return value

This method returns a list of attribute names bound to the subject. If the alias of the attribute is specified in the `AttributeEntry` class, it returns that alias.

The getAttributes method

Description

Obtains all of the attributes bound to the subject. The requestor uses the `nextElement` method to obtain the object for `Enumeration`, and then casts the object to reference the value.

Syntax

```
public Enumeration getAttributes(String alias)
  throws IllegalStateException;
```

Parameter

`alias`:

Specifies the attribute name bound to the subject. If the alias of the attribute is specified in the `AttributeEntry` class, specify that alias.

Exception

`java.lang.IllegalStateException`:

If the object is invalid, `java.lang.IllegalStateException` occurs. Note that this exception inherits from `java.lang.RuntimeException` and can therefore be compiled without being defined in `catch` or `throws`.

This exception occurs when both of the following conditions coexist:

- The subject that has this object is read-only.
- The logout process is performed using the `logout` method.

Return value

This method returns the attribute value bound to the subject if that value is found. If the value is not found, it returns `null`.

The removeAttribute method

Description

Deletes one or more attributes bound to the subject. The deleted attributes are not reflected in the user management repository. If two or more attribute values are bound to the subject, all of these values are deleted.

Syntax

```
public void removeAttribute(String alias)
  throws IllegalStateException;
```

Parameter

`alias`:

Specifies the attribute name bound to the subject. If the alias of the attribute is specified in the `AttributeEntry` class, specify that alias.

Exception

`java.lang.IllegalStateException`:

If the object is invalid, `java.lang.IllegalStateException` occurs. Note that this exception inherits from `java.lang.RuntimeException` and can therefore be compiled without being defined in `catch` or `throws`.

This exception occurs when both of the following conditions coexist:

- The subject that has this object is read-only.
- The logout process is performed using the `logout` method.

Return value

None

The size method

Description

Obtains the total number of attributes bound to the subject.

Syntax

```
public int size()
  throws IllegalStateException;
```

Parameters

None

Exception

`java.lang.IllegalStateException`:

If the object is invalid, `java.lang.IllegalStateException` occurs. Note that this exception inherits from `java.lang.RuntimeException` and can therefore be compiled without being defined in `catch` or `throws`.

This exception occurs when both of the following conditions coexist:

- The subject that has this object is read-only.
- The logout process is performed using the `logout` method.

Return value

The total number of attributes associated with the subject

15.18 The UserData class

Description

Stores user information.

The package name of the `UserData` class is `com.cosminexus.admin.auth.api.repository.ldap`.

Syntax

```
class UserData
{
  public UserData();

  public void setPassword(String password);
  public Enumeration getAttributeNames();
  public Object getAttribute(String name);
  public Enumeration getAttributes(String name);
  public void addAttribute(String name,
                               Object attr);
  public void removeAttribute(String name);
  public int size();
}
```

Constructor and methods

Constructor/method name	Functionality
`UserData` constructor	Creates an instance of the `UserData` class.
`addAttribute` method	Adds one attribute value to this object.
`getAttribute` method	From the information stored in this object, obtains one attribute value associated with the specified attribute name.
`getAttributeNames` method	Obtains a list of attribute names stored in this object.
`getAttributes` method	From the information stored in this object, obtains all of the attribute values associated with the specified attribute name.
`removeAttribute` method	Removes an attribute from this object.
`setPassword` method	Stores a password into this object.
`size` method	Obtains the total number of attributes stored in this object.

The UserData constructor

Description

Creates an instance.

Syntax

```
public UserData();
```

Parameters

None

Exceptions

None

The addAttribute method

Description

Adds one attribute value to this object. Two or more attribute values can be associated with the same attribute.

Syntax

```
public void addAttribute(String name,
                         Object attr);
```

Parameters

`name`:

Specifies the name of the attribute. If the attribute has an alias, specify that alias.

`attr`:

Specifies the value of the attribute.

Exceptions

None

Return value

None

The getAttribute method

Description

From the information stored in this object, obtains one attribute value associated with the specified attribute name. If there are two or more attribute values, this method obtains one of these values.

Syntax

```
public Object getAttribute(String name);
```

Parameter

`name`:

Specifies the name of the attribute. If the attribute has an alias, specify that alias.

Exceptions

None

Return value

This method returns the attribute value if it is found. If it is not found, it returns `null`.

The getAttributeNames method

Description

Obtains a list of attribute names stored in this object.

Syntax

```
public Enumeration getAttributeNames();
```

Parameters

None

Exceptions

None

Return value

The `Enumeration` object that holds the list of attribute names

The getAttributes method

Description

From the information stored in this object, obtains all of the attribute values associated with the specified attribute name. The requestor uses the `nextElement` method to obtain the object for `Enumeration`, and casts it to reference the value.

Syntax

```
public Enumeration getAttributes(String name);
```

Parameter

`name`:

Specifies the name of the attribute. If the attribute has an alias, specify that alias.

Exceptions

None

Return value

If an `Enumeration` object that holds the attribute values is found, this method returns that object. If the object is not found, it returns `null`.

The removeAttribute method

Description

Deletes an attribute from this object. If two or more attribute values are associated with the specified attribute name, all of these values are deleted.

Syntax

```
public void removeAttribute(String name);
```

Parameter

`name`:

Specifies the name of the attribute. If the attribute has an alias, specify that alias.

Exceptions

None

Return value

None

The setPassword method

Description

Stores a password into this object. If a value already exists, it is overwritten.

Syntax

```
public void setPassword(String password);
```

Parameter

`password`:

Specifies the password.

Exceptions

None

Return value

None

The size method

Description

Obtains the total number of attributes stored in this object.

Syntax

```
public int size();
```

Parameters

None

Exceptions

None

Return value

The total number of attributes

15.19 The WebCertificateCallback class

Description

An implementation class that passes the results of Web server-based authentication from `CallbackHandler` to a login module.

The package name of the `WebCertificateCallback` class is `com.cosminexus.admin.auth.callback`.

Syntax

```
class WebCertificateCallback implements
javax.security.auth.callback.Callback
{
  public WebCertificateCallback(String attrName);

  public void setSubjectID(String name);
  public String getSubjectID();
  public void setRequest(HttpServletRequest req);
  public HttpServletRequest getRequest();
  public void setResponse(HttpServletResponse res);
  public HttpServletResponse getResponse();
  public void setAttributeEntries(AttributeEntry[] aliases);
  public AttributeEntry[] getAttributeEntries();
  public void setTagID(String tid);
  public String getTagID();
  public void setTagEntry(String entry);
  public String getTagEntry();
}
```

Constructor and methods

Constructor/method name	Functionality
`WebCertificateCallback` constructor	Creates an instance of the `WebCertifiateCallback` class.
`getAttributeEntries` method	Obtains a reference to the object that holds the list of attributes. The reference is specified by the `setAttributeEntries` method.
`getRequest` method	Obtains a reference to `HttpServletRequest`. The reference is specified by the `setRequest` method.
`getResponse` method	Obtains a reference to `HttpServletResponse`. The reference is specified by the `setResponse` method.
`getSubjectID` method	Obtains the DN name specified by the `setSubjectID` method.
`getTagEntry` method	Obtains the entry element specified by `setTagEntry`.

Constructor/method name	Functionality
`getTagID` method	Obtains TagID specified by `setTagID`.
`setAttributeEntries` method	Stores a reference to the object that holds the list of attributes into the object. The reference is specified by the parameter.
`setRequest` method	Stores a reference to `HttpServletRequest` into the object. The reference is specified by the parameter.
`setResponse` method	Stores a reference to `HttpServletResponse` into the object. The reference is specified by the parameter.
`setSubjectID` method	Stores the DN name specified by the parameter into the object.
`setTagEntry` method	Holds the login tag's `entry` element as specified by the parameter.
`setTagID` method	Holds the login tag's `id` element as specified by the parameter.

The WebCertificateCallback constructor

Description

Executes the `WebCertificateLoginModule` class's `login` method and then creates an instance of the `WebCertificateCallback` class.

Syntax

```
public WebCertificateCallback(String attrName);
```

Parameters

`attrName`:

Specifies the attribute name to be resolved from the DN name.

Exceptions

None

The getAttributeEntries method

Description

Obtains the value specified by the `setAttributeEntries` method. If that value does not exist, this method returns `null`.

Syntax

```
public AttributeEntry[] getAttributeEntries();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The getRequest method

Description

Obtains the value specified by the `setRequest` method. If that value does not exist, this method returns `null`.

Syntax

```
public HttpServletRequest getRequest();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The getResponse method

Description

Obtains the value specified by the `setResponse` method. If that value does not exist, this method returns `null`.

Syntax

```
public HttpServletResponse getResponse();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The getSubjectID method

Description

Obtains the value specified by the `setSubjectID` method. If that value does not exist, this method returns `null`.

Syntax

```
public String getSubjectID();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The getTagEntry method

Description

Obtains the value specified by the `setTagEntry` method. If an API is used for login, this method returns `null`.

Syntax

```
public String getTagEntry();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The getTagID method

Description

Obtains the value specified by the `setTagID` method. If an API is used for login, this method returns `null`.

Syntax

```
public String getTagID();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The setAttributeEntries method

Description

Stores the value specified by the parameter into this object. If a value already exists, it is overwritten.

Syntax

```
public void setAttributeEntries(AttributeEntry[] aliases);
```

Parameter

`aliases`:

Specifies the reference to the object (`AttributeEntry` array) that holds the list of attributes.

Exceptions

None

Return value

None

The setRequest method

Description

Stores the value specified by the parameter into this object. If a value already exists, it is overwritten.

Syntax

```
public void setRequest(HttpServletRequest req);
```

Parameter

`req`:

Specifies the reference to `HttpServletRequest`.

Exceptions

None

Return value

None

The setResponse method

Description

Stores the value specified by the parameter into this object. If a value already exists, it is overwritten.

Syntax

```
public void setResponse(HttpServletResponse res);
```

Parameters

`res`:

Specifies the reference to `HttpServletResponse`.

Exceptions

None

Return value

None

The setSubjectID method

Description

Stores the value specified by the parameter into this object. If a value already exists, it is overwritten.

Syntax

```
public void setSubjectID(String uid);
```

Parameter

`uid`:

Specifies the DN name.

Exceptions

None

Return value

None

The setTagEntry method

Description

Stores the value specified by the parameter into this object.

Syntax

```
public void setTagEntry(String tid);
```

Parameter

`tid`:

Specifies the `entry` element of the login tag.

Exceptions

None

Return value

None

The setTagID method

Description

Stores the value specified by the parameter into this object.

Syntax

```
public void setTagID(String tid);
```

Parameter

`tid`:

Specifies the `id` element of the login tag.

Exceptions

None

Return value

None

15.20 The WebCertificateHandler class

Description

An implementation class that obtains the results of SSL authentication performed by a Web server. This is `CallbackHandler` in the user authentication library.

The package name of the `WebCertificateHandler` class is `com.cosminexus.admin.auth.callback`.

Syntax

```
class WebCertificateHandler
{
  public WebCertificateHandler(HttpServletRequest request,
                               HttpServletResponse response,
                                   AttributeEntry[] aliases)
    throws ParameterError;
  public WebCertificateHandler(HttpServletRequest request,
                               HttpServletResponse response,
                                     String aliasesFile)
    throws ParameterError, FormatError,
FileNotFoundException,
           IOException, SecurityException;

  public void handle(Callback[] callbacks)
    throws IOException, UnsupportedCallbackException;
}
```

Constructor and methods

Constructor/method name	Functionality
`WebCertificateHandler` constructor	Creates an instance of the `WebCertificateHandler` class.
`handle` method	Obtains the results of SSL authentication.

The WebCertificateHandler constructor

Description

Creates an instance of the `WebCertificateHandler` class. The `request` and `response` parameters are mandatory. If `null` is specified, a `ParameterError` exception is called.

Syntax

```
public WebCertificateHandler(HttpServletRequest request,
                                   HttpServletResponse response,
                                    AttributeEntry[] aliases)
    throws ParameterError;

public WebCertificateHandler(HttpServletRequest request,
                                   HttpServletResponse response,
                                    String aliasesFile)
    throws ParameterError, FormatError, FileNotFoundException,
           IOException, SecurityException;
```

Parameters

`request`:

Specifies the JSP/Servlet activation parameter with no changes.

`response`:

Specifies the JSP/Servlet activation parameter with no changes.

`aliases`:

Specifies the information to be included in the Credential (`UserAttributes`) that is created when authentication succeeds. If there is no information to be obtained, specify `null`. In this case, no Credential is created (an empty `UserAttributes` object is created). For `aliases`, specify an array of `AttributeEntry` objects. If the specified parameter lacks necessary information, a `FormatError` exception occurs (because a mandatory value is not stored or because a value that violates the format is specified).

`aliasesFile`:

Specifies the information to be included in the Credential (`UserAttributes`) that is created when authentication succeeds. If there is no information to be obtained, specify `null`. In this case, no Credential is created (an empty UserAttributes object is created). For `aliasesFile`, specify a file name. If the specified parameter lacks necessary information, a `FormatError` exception occurs (because a mandatory value is not stored or because a value that violates the format is specified).

Exceptions

`java.io.FileNotFoundException`:

The specified file cannot be opened because it is missing or is a directory, or because of some other reason (when the exception occurs in the constructor of the `FileInputStream` class).

`java.lang.SecurityException`:

SecurityManager is present, and read access to the file using SecurityManager's `checkRead` method has been rejected.

`java.io.IOException`:

An attempt to read the file has failed.

`com.cosminexus.admin.common.ParameterError`:

A reference to `HttpServletRequest` or `HttpServletResponse` is not specified.

`com.cosminexus.admin.common.FormatError`:

Either or both of the `aliases` or `aliasesFile` parameters lack necessary information or contain extra information.

The handle method

Description

Obtains the results of SSL authentication performed by a Web server and assigns to the obtained information a reference to the `WebCertificateCallback` object (`Callback implementation class`).

Syntax

```
public void handle(Callback[] callbacks)
   throws IOException, UnsupportedCallbackException;
```

Parameter

`callbacks`:

If this parameter specifies a reference to the `WebSSOCallback` object, session information is set and returned. If it specifies a class other than this one, the `handle` method of `CallbackHandler` specified in the constructor is called.

Exceptions

`java.io.IOException`:

`HttpServletRequest` does not contain the results of Web server-based authentication.

`javax.security.auth.callback.UnsupportedCallbackException`:

An unsupported `callbacks` reference is specified.

Return value

Set a value for `callbacks` so that this value can be returned. This method does not return a value.

15.21 The WebCertificateLoginModule class

Description

An implementation class for a JAAS login module, which is a Cosminexus standard login module. This class obtains user attributes from certificates authenticated by a Web server.

The package name of the `WebCertificateLoginModule` class is `com.cosminexus.admin.auth.login`.

15.22 The WebLogoutCallback class

Description

An implementation class that passes user information from a Web application to a login module via `CallbackHandler`.

The package name of the `WebLogoutCallback` class is `com.cosminexus.admin.auth.callback`.

Syntax

```
class WebLogoutCallback implements
javax.security.auth.callback.Callback
{
  private HttpSession session = null;
  private String userID = null;

  public WebLogoutCallback();
  public void setSession(HttpSession session);
  public String getSession();
  public void setUserID(String userID);
  public String getUserID();
}
```

Constructor and methods

Constructor/method name	Functionality
`WebLogoutCallback` constructor	Creates an instance of the `WebLogoutCallback` class.
`getSession` method	Obtains a reference to the `HttpSession` object. The reference is specified by the `setSession` method.
`getUserID` method	Obtains the user ID specified by the `setUserID` method.
`setSession` method	Holds a reference to the `HttpSession` object. The reference is specified by the parameter.
`setUserID` method	Holds the user ID specified by the parameter.

The WebLogoutCallback constructor

Description

Creates an instance of the `WebLogoutCallback` class. This is done when WebLogoutHandler is specified in the Cosminexus standard login module and when the `logout` method is called.

Syntax

```
public WebLogoutCallback();
```

Parameters

None

Exceptions

None

The getSession method

Description

Obtains the value specified by the `setSession` method. If that value does not exist, this method returns `null`.

Syntax

```
public HttpSession getSession();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The getUserID method

Description

Obtains the value specified by the `setUserID` method. If that value does not exist, this method returns `null`.

Syntax

```
public String getUserID();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The setSession method

Description

Stores the value specified by the parameter into this object. If a value already exists, it is overwritten.

Syntax

```
public void setSession(HttpSession session);
```

Parameter

`session`:

Specifies the reference to the `HttpSession` object.

Exceptions

None

Return value

None

The setUserID method

Description

Stores the value specified by the parameter into this object. If a value already exists, it is overwritten.

Syntax

```
public void setUserID(String userID);
```

Parameter

`userID`:

Specifies the user ID.

Exceptions

None

Return value

None

15.23 The WebLogoutHandler class

Description

An implementation of the JAAS `CallbackHandler` class that obtains the user ID of a user who logs out.

The package name of the `WebLogoutHandler` class is `com.cosminexus.admin.auth.callback`.

Syntax

```
class WebLogoutHandler
{
  public WebLogoutHandler(HttpSession session , String
userID) throws ParameterError;
  public void handle(Callback[] callbacks)
  throws java.io.IOException, UnsupportedCallbackException;
}
```

Constructor and methods

Constructor/method name	Functionality
`WebLogoutHandler` constructor	Creates an instance of the `WebLogoutHandler` class.
`handle` method	Obtains a user ID.

The WebLogoutHandler constructor

Description

Creates an instance of the `WebLogoutHandler` class.

The session and userID parameters are mandatory. If `null` is specified, a `ParameterError` exception occurs.

Syntax

```
public WebLogoutHandler(HttpSession session, String userID);
```

Parameters

`session`:

Specifies the JSP/Servlet activation parameter with no changes.

`userID`:

Specifies the user ID of the user who logs out.

Exception

`com.cosminexus.admin.common.ParameterError`:

A reference to either `HttpSession` or `String` is not specified.

The handle method

Description

Obtains a user ID, assigns to it a reference to the `WebLogoutCallback` object (`Callback implementation class`), and passes it to the login module offered by the integrated user management framework.

Syntax

```
public void handle(Callback[] callbacks)
  throws UnsupportedCallbackException;
```

Parameter

`callbacks`:

If this parameter specifies a reference to the `WebLogoutCallback` object, user information is set and returned. If it specifies a reference to another object, `UnsupportedCallbackException` occurs.

Exceptions

`java.io.IOException`:

`HttpServletRequest` does not contain the results of Web server-based authentication.

`javax.security.auth.callback.UnsupportedCallbackException`:

An unsupported `callbacks` reference is specified.

Return value

Set a value for `callbacks` so that this value can be returned. This method does not return a value.

15.24 The WebPasswordCallback class

Description

An implementation class that passes authentication information from a Web application to a login module via `CallbackHandler`.

The package name of the `WebPasswordCallback` class is `com.cosminexus.admin.auth.callback`.

Syntax

```
class WebPasswordCallback implements
javax.security.auth.callback.Callback
{
  public static final int GETPW;
  public static final int NOPW;

  public WebPasswordCallback();

  public void setName(String name);
  public String getName();
  public void setPassword(String password);
  public String getPassword();
  public void setRequest(HttpServletRequest req);
  public HttpServletRequest getRequest();
  public void setResponse(HttpServletResponse res);
  public HttpServletResponse getResponse();
  public void setAttributeEntries(AttributeEntry[] aliases);
  public AttributeEntry[] getAttributeEntries();

  public void setOption(int option);
  public int getOption();
  public void setTagID(String tid);
  public String getTagID();
  public void setTagEntry(String entry);
  public String getTagEntry();
}
```

Member attributes

`GETPW`:

Requests that all information should be set for this `Callback` object.

`NOPW`:

Requests that all information except user IDs and passwords should be set for this `Callback` object. (In this case, the `forward`/`include` method is not executed for handling urls.)

Constructor and methods

Constructor/method name	Functionality
`WebPasswordCallback` constructor	Creates an instance of the `WebPasswordCallback` class.
`getAttributeEntries` method	Obtains a reference to the object that stores the list of attributes. The reference is specified by the `setAttributeEntries` method.
`getName` method	Obtains the user ID specified by the `setName` method.
`getOption` method	Obtains the configured option.
`getPassword` method	Obtains the password specified by the `setPassword` method.
`getRequest` method	Obtains a reference to `HttpServletRequest`. The reference is specified by the `setRequest` method.
`getResponse` method	Obtains a reference to `HttpServletResponse`. The reference is specified by the `setResponse` method.
`getTagEntry` method	Obtains the entry element specified by the `setTagEntry` method.
`getTagID` method	Obtains TagID specified by the `setTagID` method.
`setAttributeEntries` method	Stores into this object a reference to the object that holds the list of attributes. The reference is specified by the parameter.
`setName` method	Stores the user ID specified by the parameter into this object.
`setOption` method	Requests the settings configured with `CallbackHandler`.
`setPassword` method	Stores the password specified by the parameter into this object.
`setRequest` method	Stores a reference to `HttpServletRequest` into this object. The reference is specified by the parameter.
`setResponse` method	Stores a reference to `HttpServletResponse` into this object. The reference is specified by the parameter.
`setTagEntry` method	Requests the settings configured with the `CallbackHandler` method. Stores the login tag's entry element as specified by the parameter.
`setTagID` method	Requests the settings configured with the `CallbackHandler` method. Stores the login tag's id element as specified by the parameter.

The WebPasswordCallback constructor

Description

Executes the `WebPasswordLoginModule` class's `login` method and then creates an instance of the `WebPasswordCallback` class.

Syntax

```
public WebPasswordCallback();
```

Parameters

None

Exceptions

None

The getAttributeEntries method

Description

Obtains the value specified by the `setAttributeEntries` method. If that value does not exist, this method returns `null`.

Syntax

```
public AttributeEntry[] getAttributeEntries();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The getName method

Description

Obtains the value specified by the `setName` method. If that value does not exist, this method returns `null`.

Syntax

```
public String getName();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The getOption method

Description

Obtains the configured option. If the value does not exist, this method returns `GETPW` (by default, this method requests that all information should be configured).

Syntax

```
public int getOption();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The getPassword method

Description

Obtains the value specified by the `setPassword` method. If that value does not exist, this method returns `null`.

Syntax

```
public String getPassword();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The getRequest method

Description

Obtains the value specified by the `setRequest` method. If that value does not exist, this method returns `null`.

Syntax

```
public HttpServletRequest getRequest();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The getResponse method

Description

Obtains the value specified by the `setResponse` method. If that value does not exist, this method returns `null`.

Syntax

```
public HttpServletResponse getResponse();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The getTagEntry method

Description

Obtains the value specified by the `setTagEntry` method. If an API is used for login, this method returns `null`.

Syntax

```
public String getTagEntry();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The getTagID method

Description

Obtains the value specified by the `setTagID` method. If an API is used for login, this method returns `null`.

Syntax

```
public String getTagID();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The setAttributeEntries method

Description

Stores the value specified by the parameter into this object. If a value already exists, it is overwritten.

Syntax

```
public void setAttributeEntries(AttributeEntry[] aliases);
```

Parameter

`aliases`:

Specifies the reference to the object (`AttributeEntry` array) that holds the list of attributes.

Exceptions

None

Return value

None

The setName method

Description

Stores the value specified by the parameter into this object. If a value already exists, it is overwritten.

Syntax

```
public void setName(String uid);
```

Parameter

`uid`:

Specifies the user ID.

Exceptions

None

Return value

None

The setOption method

Description

Requests the settings configured with `CallbackHandler`. If `NOPW` is specified, this method requests `CallbackHandler` to set all information except user IDs and passwords (the `forward`/`include` method is not executed for handling urls). If `GETPW` is specified, this method requests `CallbackHandler` to set all information that can be stored into this `Callback` object.

If a value already exists, it is overwritten.

Syntax

```
public void setOption(int option);
```

Parameter

`option`:

Specifies `GETPW` or `NOPW`.

- `GETPW`

  Requests that all information be set for this `Callback` object.

- `NOPW`

  Requests that all information except user IDs and passwords be set for this `Callback` object (the `forward/include` method is not executed for handling urls).

Exceptions

None

Return value

None

The setPassword method

Description

Stores the value specified by the parameter into this object. If a value already exists, it is overwritten.

Syntax

```
public void setPassword(String password);
```

Parameter

`password`:

Specifies the password.

Exceptions

None

Return value

None

The setRequest method

Description

Stores the value specified by the parameter into this object. If a value already exists, it is overwritten.

Syntax

```
public void setRequest(HttpServletRequest req);
```

Parameter

`req`:

Specifies the reference to `HttpServletRequest`.

Exceptions

None

Return value

None

The setResponse method

Description

Stores the value specified by the parameter into this object. If a value already exists, it is overwritten.

Syntax

```
public void setResponse(HttpServletResponse res);
```

Parameter

`res`:

Specifies the reference to `HttpServletResponse`.

Exceptions

None

Return value

None

The setTagEntry method

Description

Stores the value specified by the parameter into this object.

Syntax

```
public void setTagEntry(String tid);
```

Parameter

`tid`:

Specifies the entry element of the login tag.

Exceptions

None

Return value

None

The setTagID method

Description

Stores the value specified by the parameter into this object.

Syntax

```
public void setTagID(String tid);
```

Parameter

`tid`:

Specifies the id element of the login tag.

Exceptions

None

Return value

None

15.25 The WebPasswordHandler class

Description

An implementation of the JAAS `CallbackHandler` class that obtains a user ID and password from a user via a Web browser.

Specify the user ID and password in the `com.cosminexus.admin.auth.name` and `com.cosminexus.admin.auth.password` parameters for HTTP requests, respectively.

The package name of the `WebPasswordHandler` class is `com.cosminexus.admin.auth.callback`.

Syntax

```
class WebPasswordHandler
{
  public WebPasswordHandler(HttpServletRequest request,
                            HttpServletResponse response,
                               AttributeEntry [] aliases,
                               String url,
                               boolean urlforward)
    throws FormatError, ParameterError;
  public WebPasswordHandler(HttpServletRequest request,
                            HttpServletResponse response,
                               String aliasesFile,
                               String url,
                               boolean urlforward)
    throws IOException, SecurityException, FormatError,
    ParameterError;

  public void handle(Callback[] callbacks)
    throws IOException, UnsupportedCallbackException;
}
```

Constructor and methods

Constructor/method name	Functionality
`WebPasswordHandler` constructor	Creates an instance of the `WebPasswordHandler` class.
`handle` method	Obtains authentication information.

The WebPasswordHandler constructor

Description

Creates an instance of the `WebPasswordHandler` class.

This constructor allows you to use either memory or a file to specify user information (attributes) to be stored in the Credential (`UserAttributes`). The `request` and `response` parameters are mandatory. If `null` is specified, a `ParameterError` exception occurs.

Syntax

```
public WebPasswordHandler(HttpServletRequest request,
                               HttpServletResponse response,
                               AttributeEntry[] aliases,
                               String url,
                               boolean urlforward)
  throws FormatError, ParameterError;

public WebPasswordHandler(HttpServletRequest request,
                               HttpServletResponse response,
                               String aliasesFile,
                               String url,
                               boolean urlforward)
  throws IOException, SecurityException, FormatError,
  ParameterError;
```

Parameters

`request`:

Specifies the JSP/Servlet activation parameter with no changes.

`response`:

Specifies the JSP/Servlet activation parameter with no changes.

`aliases`:

Specifies the information to be included in the Credential (`UserAttributes`) that is created when authentication succeeds. If there is no information to be obtained, specify `null`. In this case, no Credential is created (an empty `UserAttributes` object is created). For `aliases`, specify an array of `AttributeEntry` objects. If the specified parameter lacks necessary information, a `FormatError` exception occurs (because a mandatory value is not stored or because a value that violates the format is specified).

`aliasesFile`:

Specifies the information to be included in the Credential (`UserAttributes`) that is created when authentication succeeds. If there is no information to be obtained, specify `null`. In this case, no Credential is created (an empty `UserAttributes` object is created). For `aliasesFile`, specify a file name. If the specified parameter lacks necessary information, a `FormatError` exception occurs (because a mandatory value is not stored or because a value that violates the format is specified).

`url`:

Specifies the URL from which the authentication information (user ID or password) is to be obtained from the user. If the URL is specified, a Login Form is passed to the `RequestDispatcher` object according to the specified `urlforward` parameter (when input information is obtained from the user). If there is no need to specify the URL, specify `null`. In this case, the `urlforward` value is not referenced. If you specify `null` and authentication information cannot be obtained after execution of the `handle` method (because this information is not stored in `HttpServletRequest`), `LoginException` occurs after the `LoginContext` class's `login` method is executed.

`urlforward`:

Specifies how to display the URL. If this parameter is set to `true` for the specified URL, the `forward` method of the `RequestDispatcher` object is called. If it is set to `false`, the `include` method of that object is called.

Exceptions

`java.io.FileNotFoundException`:

The specified file cannot be opened because it is missing or is a directory, or because of some other reason (when the exception occurs in the constructor of the `FileInputStream` class).

`java.lang.SecurityException`:

SecurityManager is present, and read access to the file using SecurityManager's `checkRead` method has been rejected.

`java.io.IOException`:

An attempt to read the file has failed.

`com.cosminexus.admin.common.ParameterError`:

A reference to `HttpServletRequest` or `HttpServletResponse` is not specified.

`com.cosminexus.admin.common.FormatError`:

Either or both of the `aliases` or `aliasesFile` parameters lack necessary information or contain extra information.

The handle method

Description

Obtains authentication information, assigns to it a reference to the `WebPasswordCallback` object (`Callback implementation class`), and passes it to the login module in the user authentication library.

Syntax

```
public void handle(Callback[] callbacks)
  throws IOException, UnsupportedCallbackException;
```

Parameter

`callbacks`:

If this parameter specifies a reference to the `WebPasswordCallback` object, authentication information is set and returned. If it specifies a reference to the `WebSSOCallback` object, session information is set and returned. If it specifies a reference to another object, `UnsupportedCallbackException` occurs.

Exceptions

`java.io.IOException`:

`HttpServletRequest` does not contain the user ID or password. See *Remarks* for details about the parameters necessary for obtaining information.

`javax.security.auth.callback.UnsupportedCallbackException`:

An unsupported `callbacks` reference is specified.

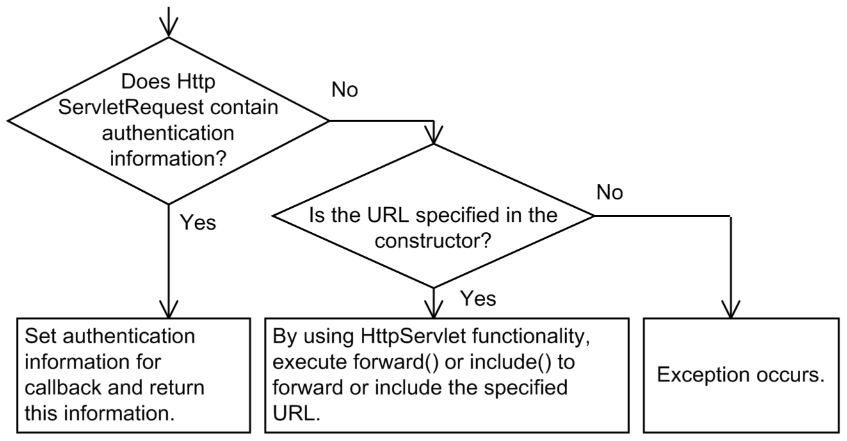
Return value

Set a value for `callbacks` so that this value can be returned. This method does not return a value.

Remarks

Authentication information is read in the order shown below.

Figure 15-1: Order of reading authentication information


Does Http ServletRequest contain authentication information?
No
Yes
Is the URL specified in the constructor?
No
Yes
Set authentication information for callback and return this information.
By using HttpServlet functionality, execute forward() or include() to forward or include the specified URL.
Exception occurs.

You can obtain authentication information from `HttpServletRequest` by using the following parameters:

- `com.cosminexus.admin.auth.name`

  Specifies the user ID defined by the user.

- `com.cosminexus.admin.auth.password`

  Specifies the password defined by the user.

15.26 The WebPasswordJDBCLoginModule class

Description

A JAAS login module implementation class that uses JDBC to access a database and authenticate a password.

The package name of the `WebPasswordJDBCLoginModule` class is `com.cosminexus.admin.auth.login`.

15.27 The WebPasswordLDAPLoginModule class

Description

A JAAS login module implementation class. Authentication is based on the results of binding to the LDAP directory server.

The package name of the `WebPasswordLDAPLoginModule` class is `com.cosminexus.admin.auth.login`.

15.28 The WebPasswordLoginModule class

Description

A JAAS login module implementation class. It authenticates passwords for Web applications.

The package name of the `WebPasswordLoginModule` class is `com.cosminexus.admin.auth.login`.

15.29 The WebSSOCallback class

Description

An implementation class that passes session information from a Web application to a login module via `CallbackHandler`.

The package name of the `WebSSOCallback` class is `com.cosminexus.admin.auth.sso.callback`.

Syntax

```
class WebSSOCallback implements
javax.security.auth.callback.Callback
{
  public WebSSOCallback();

  public void setRequest(HttpServletRequest req);
  public HttpServletRequest getRequest();
  public void setResponse(HttpServletResponse res);
  public HttpServletResponse getResponse();
  public void setTagID(String tid);
  public String getTagID();
  public void setTagEntry(String entry);
  public String getTagEntry();

}
```

Constructor and methods

Constructor/method name	Functionality
`WebSSOCallback` constructor	Creates an instance of the `WebSSOCallback` class.
`getRequest` method	Obtains a reference to `HttpServletRequest`. The reference is specified by the `setRequest` method.
`getResponse` method	Obtains a reference to `HttpServletResponse`. The reference is specified by the `setResponse` method.
`getTagEntry` method	Obtains the entry element specified by the `setTagEntry` method.
`getTagID` method	Obtains TagID specified by the `setTagID` method.
`setRequest` method	Stores a reference to `HttpServletRequest` into the object. The reference is specified by the parameter.
`setResponse` method	Stores a reference to `HttpServletResponse` into the object. The reference is specified by the parameter.
`setTagEntry` method	Holds the login tag's `entry` element as specified by the parameter.

Constructor/method name	Functionality
`setTagID` method	Holds the login tag's `id` element as specified by the parameter.

The WebSSOCallback constructor

Description

Executes the `WebSSOLoginModule` class's `login` method and then creates an instance of the `WebSSOCallback` class.

Syntax

```
public WebSSOCallback();
```

Parameters

None

Exceptions

None

The getRequest method

Description

Obtains the value specified by the `setRequest` method. If that value does not exist, this method returns `null`.

Syntax

```
public HttpServletRequest getRequest();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The getResponse method

Description

Obtains the value specified by the `setResponse` method. If that value does not exist, this method returns `null`.

Syntax

```
public HttpServletResponse getResponse();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The getTagEntry method

Description

Obtains the value specified by the `setTagEntry` method. If an API is used for login, this method returns `null`.

Syntax

```
public String getTagEntry();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The getTagID method

Description

Obtains the value specified by the `setTagID` method. If an API is used for login, this method returns `null`.

Syntax

```
public String getTagID();
```

Parameters

None

Exceptions

None

Return value

The value stored in the object

The setRequest method

Description

Stores the value specified by the parameter into this object. If a value already exists, it is overwritten.

Syntax

```
public void setRequest(HttpServletRequest req);
```

Parameter

`req`:

Specifies the reference to `HttpServletRequest`.

Exceptions

None

Return value

None

The setResponse method

Description

Stores the value specified by the parameter into this object. If a value already exists, it is overwritten.

Syntax

```
public void setResponse(HttpServletResponse res);
```

Parameter

`res`:

Specifies the reference to `HttpServletResponse`.

Exceptions

None

Return value

None

The setTagEntry method

Description

Stores the value specified by the parameter into this object.

Syntax

```
public void setTagEntry(String tid);
```

Parameter

`tid`:

Specifies the `entry` element of the login tag.

Exceptions

None

Return value

None

The setTagID method

Description

Stores the value specified by the parameter into this object.

Syntax

```
public void setTagID(String tid);
```

Parameter

`tid`:

Specifies the `id` element of the login tag.

Exceptions

None

Return value

None

15.30 The WebSSOHandler class

Description

An implementation of the JAAS `CallbackHandler` class that obtains session information via a Web browser. In the single sign-on library, this is `CallbackHandler`.

By using this class to specify references to `CallbackHandler` that are associated with the login modules for each system, you can achieve single sign-on functionality without changing the implementation of `CallbackHandler` for each system.

The package name of the `WebSSOHandler` class is `com.cosminexus.admin.auth.sso.callback`.

Syntax

```
class WebSSOHandler
{
  public WebSSOHandler(HttpServletRequest request,
                           HttpServletResponse response,
                           CallbackHandler ch)
    throws ParameterError;

  public void handle(Callback[] callbacks)
    throws IOException, UnsupportedCallbackException;
}
```

Constructor and methods

Constructor/method name	Functionality
`WebSSOHandler` constructor	Creates an instance of the `WebSSOHandler` class.
`handle` method	Obtains session information.

The WebSSOHandler constructor

Description

Creates an instance of the `WebSSOHandler` class. The `request` and `response` parameters are mandatory. If `null` is specified, a `ParameterError` exception occurs.

Syntax

```
public WebSSOHandler(HttpServletRequest request,
                         HttpServletResponse response,
                         CallbackHandler ch)
```

```
  throws ParameterError;
```

Parameters

`request`:

Specifies the JSP/Servlet activation parameter with no changes.

`response`:

Specifies the JSP/Servlet activation parameter with no changes.

`ch`:

Specifies the reference to `CallbackHandler` for each system. If this parameter is not necessary, specify `null`.

Exceptions

`com.cosminexus.admin.common.ParameterError`:

A reference to either `HttpServletRequest` or `HttpServletResponse` is not specified.

The handle method

Description

Obtains session information, assigns to it a reference to the `WebSSOCallback` object (`Callback implementation class`), and passes it to the login module in the single sign-on library.

Syntax

```
public void handle(Callback[] callbacks)
  throws IOException, UnsupportedCallbackException;
```

Parameters

`callbacks`:

If this parameter specifies a reference to the `WebSSOCallback` object, session information is set and returned. If it specifies a class other than this one, the `handle` method of `CallbackHandler` specified in the constructor is called.

Exceptions

`java.io.IOException`:

This exception occurs in the `handle` method of `CallbackHandler` for each system.

`javax.security.auth.callback.UnsupportedCallbackException`:

This exception occurs when both of the following conditions coexist:

- A reference to `CallbackHandler` for each system is not specified in the constructor (when `null` is specified).
- This exception occurred in the `handle` method of `CallbackHandler` for each system.

Return value

Set a value for `callbacks` so that this value can be returned. This method does not return a value.

15.31 The WebSSOLoginModule class

Description

A JAAS login module implementation class that calls another login module for single sign-on.

The package name of the `WebSSOLoginModule` class is `com.cosminexus.admin.auth.sso.login`.

15.32 Exception classes

This section describes the exception classes available to APIs for integrated user management. They include the exception classes for JAAS login modules and for APIs (non-JAAS APIs) from Hitachi.

(1) Exception classes for JAAS login modules

The table below lists the exception classes for JAAS login modules.

Table 15-3: List of exception classes for JAAS login modules

No.	Exception name	Description
1	`javax.security.auth.login.LoginException`	A parent class for exception classes 2 through 4. The constructor of this class has an `msg` parameter (`java.lang.String`).
2	`javax.security.auth.login.AccountExpiredException`	Reports that the user account has expired.
3	`javax.security.auth.login.CredentialExpiredException`	Reports that the credential has expired.
4	`javax.security.auth.login.FailedLoginException`	Reports that authentication has failed.

The login module in either the user authentication library or the single sign-on library assigns error message character strings to the exceptions and sends these strings. The error message character strings are listed in the table below.

Note that if the `LoginContext` class is instantiated when the JAAS configuration file contains an error,`java. lang.SecurityException` occurs. In this case, correct this configuration file by referring to the error message character string shown in the next table.

Table 15-4: Exceptions for the login module in the user authentication library or single sign-on library

Exception name	Error message character string	Reason for occurrence
`javax.security.auth.login.FailedLoginException`	`data not found`	Authentication information is not found in the passed parameter. The `HttpServletRequest` passed to the `WebPasswordHandler` class did not contain a user ID or password.

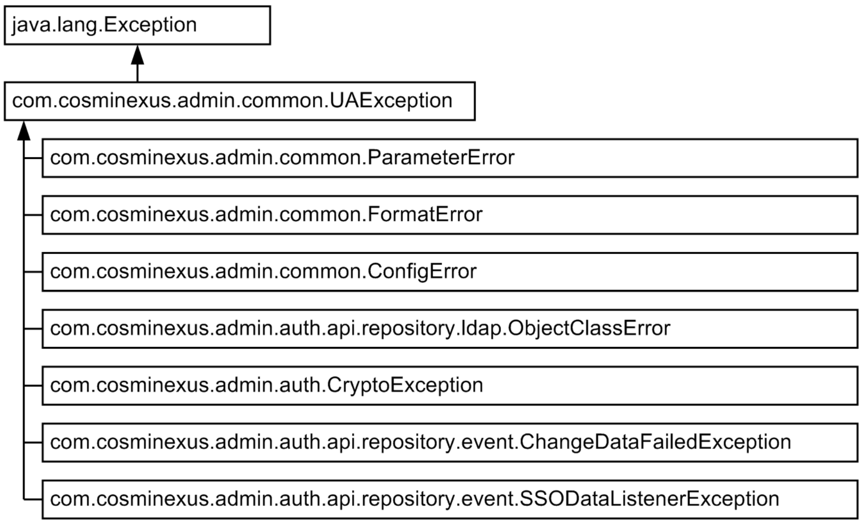
Exception name	Error message character string	Reason for occurrence
	`invalid data`	• Authentication is impossible because the user ID or password is wrong. • The entry associated with the user ID from the certificate was not found in the repository.
	`no data`	With user already authenticated within the session, necessary definitions do not include single sign-on authentication information that is associated with the realm to be called.
`javax.security.auth.login.LoginException`	`invalid parameter`	The list of attribute names and attributes for creating a credential contains errors indicating that: • An attribute name is not specified. • The same alias is specified more than once.
	SQL exception name	Access through JDBC has failed. If this exception occurs, take corrective action by referring to the error message character string.
	JNDI exception name	LDAP access has failed. • The LDAP server was not found (`CommunicationException`). • There is a bind DN specification error (`AuthenticationException`).
	`not supported`	An unsupported `CallbackHandler` is being used. • Information required for either `WebSSOLoginModule` or `WebPasswordLoginModule` cannot be obtained with `CallbackHandler`. • An exception has occurred when executing the `handle` method. This exception only occurs in `CallbackHandler` for user management when the condition described above exists.
	`no class for` *xxx*	The class called from `WebSSOLoginModule` cannot be used (*xxx* indicates the value specified by `com.cosminexus.admin.auth.sso.loginmodule`). • The class cannot be instantiated. The JAAS login module was not inherited. The access permission might be missing and the class path might not be configured.
	`config error`	• Processing cannot continue because the JAAS configuration file does not contain necessary information. • Processing cannot continue because the configuration file for user management with the Cosminexus standard login module does not contain necessary information.

Exception name	Error message character string	Reason for occurrence
	`invalid session`	When an attempt was made to bind the session to an `HttpSession` object, the `HttpSession` object became invalid.
	`crypto error`	Encryption or decryption has failed. • The shared single sign-on library to be called by JNI functionality was not found (there is a problem with the `java.library.path` settings). • Decryption has failed (different keys are used for encryption and decryption).
	`no sso data`	Information for single sign-on is not found. • Necessary information for single sign-on is missing.
	`no principal`	Because Principal was missing, the first authenticated user could not be identified.
	`class cast error`	There is a mismatch between the type fetched from the repository and that specified in the configuration file for integrated user management. Match these two types. See `com.cosminexus.admin.auth.ldap.password.encrypt` in `ua.conf` (which is the configuration file for integrated user management). For details about this `ua.conf` file, see *14.3 ua.conf (integrated user management configuration file)*.
	`not found driver`	JDBC is used. • The driver was not found in `WebPasswordJDBCLoginModule`. Store the driver in the correct position.
	Other	An error has occurred in one of the login modules for the systems. • An error has occurred in `WebSSOLoginModule`, which is a login module in a library other than the user authentication library.

(2) Exception classes for APIs offered by Hitachi

The figure below shows the hierarchy of the exception classes for APIs (non-JAAS APIs) offered by Hitachi.

Figure 15-2: Hierarchy of exception classes


java.lang.Exception
com.cosminexus.admin.common.UAException
com.cosminexus.admin.common.ParameterError
com.cosminexus.admin.common.FormatError
com.cosminexus.admin.common.ConfigError
com.cosminexus.admin.auth.api.repository.ldap.ObjectClassError
com.cosminexus.admin.auth.CryptoException
com.cosminexus.admin.auth.api.repository.event.ChangeDataFailedException
com.cosminexus.admin.auth.api.repository.event.SSODataListenerException

The following table lists these exception classes.

Table 15-5: Exception classes for APIs offered by Hitachi

No.	Exception name	Description
1	`com.cosminexus.admin.common.UAException`	A parent class for exception classes 2 through 8.
2	`com.cosminexus.admin.common.ParameterError`	There is a parameter error in one of the APIs.
3	`com.cosminexus.admin.common.FormatError`	There is a format error in one of the APIs.
4	`com.cosminexus.admin.common.ConfigError`	There is a configuration file error.
5	`com.cosminexus.admin.auth.api.repository.ldap.ObjectClassError`	There is an object class error.
6	`com.cosminexus.admin.auth.CryptoException`	Encryption or decryption has failed.
7	`com.cosminexus.admin.auth.api.repository.event.ChangeDataFailedException`	The listener class is called if an attempt to update authentication information for another system has failed.
8	`com.cosminexus.admin.auth.api.repository.event.SSODataListenerException`	The `LdapSSODataManager` class is called if an attempt to update authentication information for another system has failed.

Chapter

16. Tag Library Used with the Integrated User Management Framework

This chapter describes the JSP tag library that is used with the integrated user management framework.

16.1 List of the tags contained in the tag library

The integrated user management framework provides a JSP tag library for implementation of user authentication functionality in JSP.

To import this library into JSP, write the following code in JSP:

```
<%@ taglib uri="http://cosminexus.com/admin/auth/uatags"
prefix="ua" %>
```

The table below lists the tags contained in this library.

Table 16-1: List of the tags in the JSP tag library

Tag name	Overview
`<ua:attributeEntries>Entries</ua:attributeEntries>` tag	Used together with `<ua:attributeEntry/>` tags to specify a list of user attributes to be obtained at login.
`<ua:attributeEntry/>` tag	Specifies the user attribute to be obtained at login.
`<ua:chpw/>` tag	Changes the specified user's password.
`<ua:exception>Body</ua:exception>` tag	Specifies the exception processing.
`<ua:getPrincipalName/>` tag	Obtains or displays the Principal name (user ID) of a logged-in user.
`<ua:getAttribute/>` tag	Obtains or displays the user attribute value of a logged-in user.
`<ua:getAttributes/>` tag	Obtains or displays the user attribute values (Multi-Value) of a logged-in user.
`<ua:getAttributeNames/>` tag	Obtains or displays a list of the user attribute names of a logged-in user.
`<ua:login/>` tag	Logs a user in.
`<ua:logout/>` tag	Logs a user out.
`<ua:notLogin>Body</ua:notLogin>` tag	Specifies the processing required when there is no logged-in user. If this tag is entered at the beginning of each JSP page, you can check for a logged-in user before processing a JSP page.

This chapter describes the attributes of each tag in tabular form. The *Type* column indicates the type of script variable defined for each tag attribute. The *C/R* column includes `C` or `R`. `C` indicates that the tag attribute value is evaluated at JSP compile time. `R` indicates that it is evaluated at execution time.

16.2 The <ua:attributeEntries>Entries</ua:attributeEntries> tag

(1) Description

Used together with `<ua:attributeEntry/>` tags to specify a list of user attributes to be obtained at login. To specify this list, enter two or more `<ua:attributeEntry/>` tags for `Entries`.

```
<ua:attributeEntries id="ae">
  <ua:attributeEntry attrName="cn" />
  <ua:attributeEntry attrName="sn" />
  ...
</ua:attributeEntries>
```

(2) Tag attributes

The following table lists the tag attributes.

Table 16-2: Tag attributes (<ua:attributeEntries>Entries</ua:attributeEntries> tag)

Tag attribute	Description	Type	Required ?	C/R
`id="`*id*`"`	Specifies the identifier of an instance (array) of the `AttributeEntry` class. This tag attribute can also be used for the name of the script variable that references this instance. This instance has the scope that is specified with the `scope` tag attribute. The specified identifier should be unique for each scope.	`AttributeEntry[]`	R	C
`scope="`*scope*`"`	Specifies the scope of the script variable that is specified with the `id` tag attribute. The scope can be specified as `page`, `request`, `session`, or `application`. For the meanings of these values, see the description for the `<jsp:useBean/>` tag. If the `scope` tag attribute is omitted, `page` is assumed.	n/a	O	C

Legend:

R: Required

O: Optional

n/a: Not applicable

C: Indicates that the tag attribute value is evaluated at JSP compile time.

16.3 The <ua:attributeEntry/> tag

(1) Description

Specifies the user attribute to be obtained at login. For details, see *16.2 The <ua:attributeEntries>Entries</ua:attributeEntries> tag*.

(2) Tag attributes

The following table lists the tag attributes.

Table 16-3: Tag attributes (<ua:attributeEntry/> tag)

Tag attribute	Description	Type	Required ?	C/R
`attrName="`*attrName*`"`	Specifies the name of the user attribute to be obtained at login.	n/a	R	E
`alias="`*alias*`"`	Specifies the alias of the user attribute to be obtained.	n/a	O	E
`subCxt="`*subCxt*`"`	Specifies the subcontext from the user management context.	n/a	O	E

Legend:
R: Required
O: Optional
n/a: Not applicable
E: Indicates that the tag attribute value is evaluated at execution time.

16.4 The <ua:chpw/> tag

(1) Description

Changes the specified user's password.

(2) Tag attributes

The following table lists the tag attributes.

Table 16-4: Tag attributes (<ua:chpw/> tag)

Tag attribute	Description	Type	Required ?	C/R
entry="*JAAS-entry*"	Specifies the entry name for the JAAS configuration file.	n/a	R	E
userid="*userid*"	Specifies the user ID for the user whose password is to be changed.	n/a	R#	E
useridParam="*userid Param*"	Specifies the name of the HTTP request parameter that holds the user ID for the user whose password is to be changed.	n/a		E
oldpw="*oldpw*"	Specifies the current password in plain text.	n/a	R#	E
oldpwParam="*oldpwP aram*"	Specifies the name of the HTTP request parameter that holds the current password.	n/a		E
newpw="*newpw*"	Specifies the new password in plain text.	n/a	R#	E
newpwParam="*newpw Param*"	Specifies the name of the HTTP request parameter that holds the new password.	n/a		E
excepId="*excepId*"	Specifies the identifier of an instance of an exception that occurs while changing the password. This instance has the scope that is specified with the `excepScope` tag attribute. The specified identifier should be unique for each scope. If the `excepId` tag attribute is omitted, exceptions that occur while changing the password are propagated outside the `<ua:chpw/>` tag as `JspException`.	n/a	O	C

Tag attribute	Description	Type	Required ?	C/R
`excepScope="`*scope*`"`	Specifies the scope of the script variable that is specified with the `excepId` tag attribute. The scope can be specified as `page`, `request`, `session`, or `application`. For the meanings of these values, see the description for the `<jsp:useBean/>` tag. If the `excepScope` tag attribute is omitted, `page` is assumed.	n/a	O	C

Legend:

R: Required

O: Optional

n/a: Not applicable

C: Indicates that the tag attribute value is evaluated at JSP compile time.

E: Indicates that the tag attribute value is evaluated at execution time.

#

Specify either of these tag attributes.

16.5 The <ua:exception>Body</ua:exception> tag

(1) Description

Defines the process to be performed when the specified exception occurs.

(2) Tag attributes

The following table lists the tag attributes.

Table 16-5: Tag attributes (<ua:exception>Body</ua:exception> tag)

Tag attribute	Description	Type	Required ?	C/R
`name="`*name*`"`	Specifies the identifier (specified with the `excepId` tag attribute) of an exception object specified with the `<ua:login/>` or `<ua:chpw/>` tag. If an incorrect identifier is specified, nothing happens.	n/a	R	C
`type="`*type*`"`	Specifies the class name of the exception object to be caught. This name should be a complete name including the package name. If the exception object can be cast into a class that has the specified class name, `Body` is executed. If the `type` tag attribute is omitted, `java.lang.Throwable` is assumed.	n/a	O	C
`proceed="`*proceed*`"`	Specifies whether to process the remaining JSP pages after `Body` processing. If `true` (case-insensitive) is specified, these JSP pages are processed. If `true` is not specified, they are not processed. If the `proceed` tag attribute is omitted, `false` is assumed.	n/a	O	E

Legend:
R: Required
O: Optional
n/a: Not applicable
C: Indicates that the tag attribute value is evaluated at JSP compile time.
E: Indicates that the tag attribute value is evaluated at execution time.

16.6 The <ua:getPrincipalName/> tag

(1) Description

Obtains or displays the Principal name (user ID) of a logged-in user.

(2) Tag attributes

The following table lists the tag attributes.

Table 16-6: Tag attributes (<ua:getPrincipalName/> tag)

Tag attribute	Description	Type	Required ?	C/R
`id="`*id*`"`	Specifies the identifier of an instance that references the Principal name (user ID) of a logged-in user. This tag attribute can also be used for the name of the script variable that references this instance. This instance has a page scope, which means that it can be referenced within the JSP page in which it is created. Therefore, the specified identifier should be unique for each page scope. If the `id` tag attribute is omitted, the obtained Principal name is embedded into the JSP page.	String	O	C
`name="`*name*`"`	Specifies the identifier (specified with the `id` tag attribute) of the JAAS `LoginContext` object specified with the `<ua:login/>` tag. If an incorrect identifier is specified, the `id` script variable is set to `null`.	n/a	R	E

Legend:
R: Required
O: Optional
n/a: Not applicable
C: Indicates that the tag attribute value is evaluated at JSP compile time.
E: Indicates that the tag attribute value is evaluated at execution time.

16.7 The <ua:getAttribute/> tag

(1) Description

Obtains or displays the user attribute value of a logged-in user.

(2) Tag attributes

The following table lists the tag attributes.

Table 16-7: Tag attributes (<ua:getAttribute/> tag)

Tag attribute	Description	Type	Required ?	C/R
`id="`*id*`"`	Specifies the identifier of an instance that references the user attribute value of a logged-in user. This tag attribute can also be used for the name of the script variable that references this instance. This instance has a page scope, which means that it can be referenced within the JSP page in which it is created. Therefore, the specified identifier should be unique for each page scope. If the `id` tag attribute is omitted, the obtained user attribute value is embedded into a JSP page by using the `toString` method.	Value of the `type` tag attribute	O	C
`name="`*name*`"`	Specifies the identifier (specified with the `id` tag attribute) of the JAAS `LoginContext` object specified with the `<ua:login/>` tag. If an incorrect identifier is specified, the `id` script variable is set to `null`.	n/a	R	E
`attrName="`*attrName*`"`	Specifies the name of the user attribute to be obtained. If that user attribute does not exist, the `id` script variable is set to `null`.	n/a	R	E
`type="`*type*`"`	Specifies the class name of the attribute object to be obtained. This name should be a complete name including the package name. If the `type` tag attribute is omitted, `java.lang.String` is assumed.	n/a	O	C

Legend:

R: Required

O: Optional

n/a: Not applicable

C: Indicates that the tag attribute value is evaluated at JSP compile time.
E: Indicates that the tag attribute value is evaluated at execution time.

16.8 The <ua:getAttributes/> tag

(1) Description

Obtains or displays the user attribute values (Multi-Value) of a logged-in user.

(2) Tag attributes

The following table lists the tag attributes.

Table 16-8: Tag attributes (<ua:getAttributes/> tag)

Tag attribute	Description	Type	Required ?	C/R
`id="`*id*`"`	Specifies the identifier of an instance that references the user attribute values (Multi-Value) of a logged-in user. This tag attribute can also be used for the name of the script variable that references this instance. The type of this script variable is `java.util. Enumeration`. This instance has a page scope, which means that it can be referenced within the JSP page in which it is created. Therefore, the specified identifier should be unique for each page scope. If the `id` tag attribute is omitted, the obtained user attribute values are embedded one at a time into the JSP page by using the `toString` method.	`java.util.Enumeration`	O	C
`name="`*name*`"`	Specifies the identifier (specified with the `id` tag attribute) of the JAAS `LoginContext` object specified with the `<ua:login/>` tag. If an incorrect identifier is specified, the `id` script variable is set to `null`.	n/a	R	E
`attrName="`*attrName*`"`	Specifies the name of the user attribute to be obtained. If that user attribute does not exist, the `id` script variable is set to `null`.	n/a	R	E

Legend:

R: Required

O: Optional

n/a: Not applicable

C: Indicates that the tag attribute value is evaluated at JSP compile time.

E: Indicates that the tag attribute value is evaluated at execution time.

16.9 The <ua:getAttributeNames/> tag

(1) Description

Obtains or displays a list of the user attribute names of a logged-in user.

(2) Tag attributes

The following table lists the tag attributes.

Table 16-9: Tag attributes (<ua:getAttributeNames/> tag)

Tag attribute	Description	Type	Required ?	C/R
`id="`*id*`"`	Specifies the identifier of an instance that references the list of the user attribute names of a logged-in user. This tag attribute can also be used for the name of the script variable that references this instance. The type of this script variable is `java.util.Enumeration`. This instance has a page scope, which means that it can be referenced within the JSP page in which it is created. Therefore, the specified identifier should be unique for each page scope. If the `id` tag attribute is omitted, the obtained user attribute values are embedded one at a time into the JSP page by using the `toString` method.	`java.util.Enumeration`	O	C
`name="`*name*`"`	Specifies the identifier (specified with the `id` tag attribute) of the JAAS `LoginContext` object specified with the `<ua:login/>` tag. If an incorrect identifier is specified, the `id` script variable is set to `null`.	n/a	R	E

Legend:

R: Required

O: Optional

n/a: Not applicable

C: Indicates that the tag attribute value is evaluated at JSP compile time.

E: Indicates that the tag attribute value is evaluated at execution time.

16.10 The <ua:login/> tag

(1) Description

Uses the specified JAAS configuration entry to log a user in. When this is done, the `com.cosminexus.admin.auth.name` and `com.cosminexus.admin.auth.password` parameters should be specified for the HTTP request object. If the `<ua:logout/>` tag for the `<ua:login/>` tag is not defined, the user is implicitly logged out when the session is disconnected.

(2) Tag attributes

The following table lists the tag attributes.

Table 16-10: Tag attributes (<ua:login/> tag)

Tag attribute	Description	Type	Required?	C/R
`id="`*id*`"`	Specifies the identifier of an instance of the JAAS `LoginContext` class. This tag attribute can also be used for the name of the script variable that references this instance. This instance has a session scope, which means that it can be referenced from different JSP pages participating in the same session. The specified identifier should be unique for each scope.	`javax.security.auth.login.LoginContext`	R	C
`entry="`*JAAS-entry*`"`	Specifies the entry name of the JAAS configuration file.	n/a	R	E
`attrFile="`*attrFile*`"`	Specifies the name of a file that defines the attributes to be obtained from the user management repository. The file should be a CSV file that contains single sign-on authentication information. For details about this CSV file, see *14.4 CSV files containing single sign-on authentication information*.	n/a	O#	E
`attrEntName="`*attrEntName*`"`	Specifies the identifier specified with the `id` tag attribute of the `<ua:attributeEntries>` tag.	n/a		E

Tag attribute	Description	Type	Required ?	C/R
`excepId="`*excepId*`"`	Specifies the identifier of an instance of an exception that occurs during the login processing. This instance has the scope that is specified with the `excepScope` tag attribute. The specified identifier should be unique for each scope. If the `excepId` tag attribute is omitted, exceptions that occur during login processing are propagated outside the `<ua:login/>` tag as `JspException`.	n/a	O	C
`excepScope="`*scope*`"`	Specifies the scope of the script variable that is specified with the `excepId` tag attribute. The scope can be specified as `page`, `request`, `session`, or `application`. For the meanings of these values, see the description for the `<jsp:useBean/>` tag. If the `excepScope` tag attribute is omitted, `page` is assumed.	n/a	O	C

Legend:

R: Required

O: Optional

n/a: Not applicable

C: Indicates that the tag attribute value is evaluated at JSP compile time.

E: Indicates that the tag attribute value is evaluated at execution time.

#

Specify either of these tag attributes.

16.11 The <ua:logout/> tag

(1) Description

Logs a user out. If no user is logged in, nothing happens.

(2) Tag attributes

The following table lists the tag attributes.

Table 16-11: Tag attributes (<ua:logout/> tag)

Tag attribute	Description	Type	Required ?	C/R
`name="`*name*`"`	Specifies the identifier (specified with the `id` tag attribute) of the JAAS `LoginContext` object specified with the `<ua:login/>` tag. If an incorrect identifier is specified, nothing happens.	n/a	R	E

Legend:

R: Required

n/a: Not applicable

E: Indicates that the tag attribute value is evaluated at execution time.

16.12 The <ua:notLogin>Body</ua:notLogin> tag

(1) Description

Defines the processing to be performed if no user is logged in. By entering this tag at the beginning of a JSP page, you can check whether a user has logged in before the JSP page is processed.

(2) Tag attributes

The following table lists the tag attributes.

Table 16-12: Tag attributes (<ua:notLogin>Body</ua:notLogin> tag)

Tag attribute	Description	Type	Required ?	C/R
`realm="`*realm*`"`	Specifies the name of a realm. If a user has not already logged into that realm, `Body` is executed. If the `realm` tag attribute is omitted and a user has not already logged into any realms, `Body` is executed.	n/a	O	E
`proceed="`*proceed*`"`	Specifies whether to process the remaining JSP pages after `Body` processing. If `true` (case- insensitive) is specified, the remaining JSP pages are processed. If `true` is not specified, they are not processed. If the `proceed` tag attribute is omitted, `false` is assumed.	n/a	O	E

Legend:

O: Optional

n/a: Not applicable

E: Indicates that the tag attribute value is evaluated at execution time.

Chapter

17. APIs for Implementation of EJB Client Applications

Some APIs used for EJB client applications provide classes intended for security purposes. This chapter focuses on these security-related classes.

17.1 The LoginInfoManager class

Description

Provides security authentication by using the user name and password specified for a J2EE server.

The following describes the J2EE servers responsible for security authentication.

- When the `ejbserver.security.service.url` property is specified:

  J2EE servers connect to the CORBA naming service specified by the `ejbserver.security.service.url` property. If one of the servers has the same name as that specified by the `ejbserver.serverName` property, the J2EE server with that name is responsible for security authentication.

  Specify the `ejbserver.security.service.url` property when security authentication involves using a J2EE server that is not connected to the CORBA naming service specified by the `java.naming.provider.url` property.

- When the `ejbserver.security.service.url` property is not specified:

  J2EE servers connect to the CORBA naming service specified by the `java.naming.provider.url` property. If one of the servers has the same name as that specified by the `ejbserver.serverName` property, the J2EE server with that name is responsible for security authentication.

In a load balancing configuration that uses JNDI round-robin search or CTM linkage functionality, there will be two or more J2EE servers that can provide security authentication. When using this configuration, you need to configure the same user name and the same role for all of the J2EE servers and then select one J2EE server for security authentication.

For details about the properties, see *14. Files Used for Java Applications*, in the *uCosminexus Application Server Definition Reference Guide*. For details about how to implement security in an EJB client application, see *3.6 Implementing security in an EJB client application*, in the *uCosminexus Application Server EJB Container Functionality Guide*.

The package name of the `LoginInfoManager` class is `com.hitachi.software.ejb.security.base.authentication`.

Methods

Method name	Functionality
`getLoginInfoManager` method	Obtains the `LoginInfoManager` object.

Method name	Functionality
`login` method	Logs into a J2EE server.
`logout` method	Logs out of a J2EE server.

Remarks

When using the methods of the `LoginInfoManager` class, observe the following precautions:

- We recommend issuing the methods of the `LoginInfoManager` class from an EJB client application. If you issue them from within a JSP, servlet, or EJB, information configured with the RunAs functionality is deleted for each request.
- Make sure that you issue the `logout` method after calling a J2EE server by issuing the `login` method.
- Do not issue the `login` and `logout` methods as nested methods. If you repeatedly issue the `login` method without issuing the `logout` method, information specified with the first `login` method is overwritten by the following `login` method.

The getLoginInfoManager method

Description

Obtains the `LoginInfoManager` object.

Syntax

```
public static LoginInfoManager getLoginInfoManager();
```

Parameters

None

Exceptions

None

Return value

The `LoginInfoManger` object

The login method

Description

Logs into a J2EE server.

Syntax

```
public final boolean login(String username,
                                 String password)
  throws NotFoundServerException,InvalidUserNameException,
          InvalidPasswordException;
```

Parameters

`username`:

Specifies the user name (plain text).

`password`:

Specifies the password (plain text).

Exceptions

`com.hitachi.software.ejb.security.base.authentication.NotFoundServerException`:

The J2EE server has not been found.

`com.hitachi.software.ejb.security.base.authentication.InvalidUserNameException`:

The specified user name has not been found.

`com.hitachi.software.ejb.security.base.authentication.InvalidPasswordException`:

The specified password is invalid.

Return value

`true`:

Login has succeeded.

`false`:

Login has failed.

The logout method

Description

Logs out of a J2EE server.

Syntax

```
public final void logout();
```

Parameters

None

Exceptions

None

Return value

None

Chapter

18. Files Used to Control Load Balancers That Employ API-Based Direct Connections

This chapter describes the format, location, functionality, and specifiable keys for files used when using administration functionality to control load balancers that employ API-based direct connections.

18.1 List of files used to control load balancers that employ API-based direct connections

The following table shows the files used to control load balancers that employ API-based direct connections.

Table 18-1: List of files used to control load balancers that employ API-based direct connections

File name	Type	Description	Relevant information
`lb.properties`	Load balancer definition property file	Configures connection information necessary to access the load balancer.	18.2
LB-connection-information-distinguished-name`.properties`	Virtual server manager-side load balancer connection configuration property file	Configures connection information necessary to access the load balancer in a virtual server manager.	18.3
`tierlb.properties`	Tier-side load balancer connection configuration property file	Configures connection information necessary to access the load balancer within a tier.	18.4

18.2 lb.properties (load balancer definition property file)

(1) Format

Java property format

(2) File location

- In Windows

  Cosminexus-installation-directory`\manager\config`
- In UNIX

  `/opt/Cosminexus/manager/config`

(3) Functionality

This file configures connection information necessary to access the load balancer. It is used to control the load balancer from Cosminexus.

(4) Keys that can be specified

The following table shows the keys that can be specified for load balancers that employ API-based direct connections, along with their defaults.

Key name	Description	Default
`lb.list`	Specify the management IP address of the load balancer using dot notation (xxx.xxx.xxx.xxx), where *xxx* is an integer from `0` to `255`. If more than one load balancer is used, separate each of the management IP addresses by a comma (`,`).	None
`lb.connect_type.`*IP-address*#1#2	Specify the connection condition for the load balancer. Specify `API`. If this is omitted, `jp1_nc` (indicating compatibility with older VRs) is set.	`jp1_nc`
`lb.enable_passwd.`*IP-address*	Specify the password necessary for the Privileged EXEC level set for the load balancer. This setting is required for each load balancer. Specify this property if ACOS is used.	None
`lb.API.user.`*IP-address*	Specify the user name used to log into the load balancer via the API.	None
`lb.API.passwd.`*IP-address*	Specify the user password used to log into the load balancer via the API.	None
`lb.API.port.`*IP-address*	Specify the port number for the load balancer. The value that can be specified is an integer from `1` to `65534`. If the specified value is out of the range, the default value is set.	`443`

Key name	Description	Default
`lb.API.cookie_persistence_template.`*Web-system-name*`.`*IP-address*	Specify the cookie persistence template name created on the load balancer.	None
`lb.API.timeout.`*IP-address*	Specify the timeout period (in seconds) for API method execution. If an API method is not completed within the period specified in this key, Smart Composer functionality commands (such as `cmx_build_system`) will end abnormally due to a timeout error. The value that can be specified is an integer from `1` and `2147483`. If the specified value is out of the range, the default value is set.	`10`
`lb.API.protocol.`*IP-address*	Specify the protocol to be used for communication with the load balancer. • `http`: Uses the HTTP communication protocol. • `https`: Uses the HTTPS communication protocol.	`https`
`javax.net.ssl.trustStore`	If API is specified as the connection condition for the load balancer, specify the trust store where the server certificate for the load balancer is registered. Specify the trust store in accordance with the Java specifications.	cacerts (Default trust store for JDK)
`javax.net.ssl.trustStorePassword`	If API is specified as the connection condition for the load balancer, specify the password for the trust store where the server certificate for the load balancer is registered. Specify the trust store in accordance with the Java specifications.	cacerts (The default key store trusted by Java is used.)

#1: Specify the management IP address of the load balancer specified in `lb.list`.

#2: If the specified value is invalid and a `cmx_test_lb`, `cmx_build_system`, `cmx_delete_system`, `cmx_start_target`, or `cmx_stop_target` command is executed, an error will occur.

(5) Specification example

```
lb.list=192.168.10.100
lb.enable_passwd.192.168.10.100=adminpw

lb.connect_type.192.168.10.100=API
lb.API.user.192.168.10.100=user01
lb.API.passwd.192.168.10.100=user01pw
#lb.API.port.192.168.10.100=443
#lb.API.cookie_persistence_template.MyWebSystem.192.168.10.100
=SC_COOKIE_TEMPNAME
#lb.API.timeout.192.168.10.100=10
javax.net.ssl.trustStore=C:\\work\\ACOS.keystore
javax.net.ssl.trustStorePassword=keystore_pass
```

(6) Notes

- This file contains password and other information. Set appropriate access permissions for the file.
- If the configuration file is updated while Management Server is running, the updated information is incorporated the next time Management Server is restarted.
- If the file is updated, or if the connection configuration between the Management Server machine and load balancer is changed, use the `cmx_test_lb` command to check the connection to the load balancer. For details about the `cmx_test_lb` command, see *8. Commands for Smart Composer Functionality*, in the *uCosminexus Application Server Command Reference Guide*. If you are unable to connect to the load balancer, use the messages output by the `cmx_test_lb` command to check the configuration of the load balancer and the specified contents of the load balancer definition (`<load-balancer>` tag definition) in the Easy Setup definition file. For details about how to configure load balancers, see *8.5 Load balancer connection information setup with Management Server (Smart Composer functionality)*. For details about load balancer definitions (`<load-balancer>` tag definitions) in the Easy Setup definition file, see *4.7.5 Configuring an environment to connect to load balancers*, in the *uCosminexus Application Server System Setup and Operation Guide*.

18.3 LB-information-distinguished-name.properties (virtual server manager-side load balancer connection configuration property file)

(1) Format

J2SE property file format

(2) File location

- In Windows

  Cosminexus-installation-directory`\manager\vmi\config\lb\`

- In UNIX

  `/opt/Cosminexus/manager/vmi/config/lb/`

(3) Functionality

This file configures connection information necessary to access the load balancer in the virtual server manager.

(4) Keys that can be specified

The keys that can be specified are shown below. The *Default value* column gives the value assumed if the key is omitted. *VR* indicates the version of the application server on which the keys are introduced or changed.

Key name	Description	Specifiable value	Default value	VR
`lb.type`	Specify the load balancer type.	The following values can be specified: • `BIG-IPv9` • `BIG-IPv10.1` • `BIG-IPv10.2` • `BIG-IPv11` • `ACOS`	None	08-53
`lb.host`	Specify the management IP address of the load balancer to be connected.	IPv4 dot notation	None	08-53
`lb.protocol`	Specify the method of connecting to the load balancer. `API`: Uses the API to connect to the load balancer	The following value can be specified: • `API`	None	08-53

Key name	Description	Specifiable value	Default value	VR
`lb.port`	Specify the port number to be used by the load balancer.	The following values can be specified: • API `1-65534`	• API: `443`	08-53
`lb.user`	Specify the user name used to connect to the load balancer.	Any character string can be used.	None	08-53
`lb.password`	Specify the user password used to connect to the load balancer.	Any character string can be used.	Null	08-53
`lb.persistence.cookie-insert.templatename`	Specify the cookie persistence template name created on the load balancer. The specified key becomes effective if the following conditions are met: • `lb.type=ACOS` and `lb.protocol=API` are specified in *LB-connection-information-distinguished-name*`.properties` or `tierlb.properties` • `lb.persistence.method=cookie-insert` is specified in `tier.properties`	Specify a 1-13 character string consisting of alphanumeric characters and underscores (_)	None	08-70
`lb.timeout`	Specify the timeout period (in seconds) for login to the loader balancer or command transmission. If login to the load balancer or a CLI command issued to the load balancer is not complete within the time specified in this key, the `vmiunit` command will end abnormally due to a timeout error.	`1-2147483`	`10`	08-53
`lb.API.protocol`	Specify the protocol to be used for communication with the load balancer. `http`: Uses the HTTP communication protocol. `https`: Uses the HTTPS communication protocol.	The following values can be specified: • `http` • `https`	`https`	09-00
`lb.ACOS.privilegedexec.password`	Specify the password necessary for the Privileged EXEC level set on ACOS. The specified key is only valid when the load balancer type is ACOS.	Any character string can be used.	None	08-53

Key name	Description	Specifiable value	Default value	VR
`javax.net.ssl.trustStore`	Specify the trust store where the server certificate for the load balancer is registered. The specified key becomes effective if all of the following values are specified in *LB-connection-information-distinguished-name*`.properties` or `tierlb.properties`: • `lb.type=ACOS` • `lb.protocol=API` • `lb.API.protocol=https`	Follow the Java specifications.	cacerts (Default trust store for JDK)	08-70
`javax.net.ssl.trustStorePassword`	Specify the password for the trust store where the server certificate for the load balancer is registered. The specified key becomes effective if all of the following values are specified in *LB-connection-information-distinguished-name*`.properties` or `tierlb.properties`: • `lb.type=ACOS` • `lb.protocol=API` • `lb.API.protocol=https`	Follow the Java specifications.	cacerts (The default key store trusted by Java is used.)	08-70

(5) Notes

- This file contains password and other information. Set appropriate access permissions for the file.
- If the configuration file is updated while a virtual server manager is running, the updated information is incorporated the next time the virtual server manager is restarted.

18.4 tierlb.properties (tier-side load balancer connection configuration property file)

(1) Format

J2SE property file format

(2) File location

- In Windows

  definition-directory`\`*tier-specific-definition-directory*`\vmi\`

- In UNIX

  definition-directory`/`*tier-specific-definition-directory*`/vmi/`

Copy and use the following template file:

- In Windows

  Cosminexus-installation-directory`\manager\vmi\templates\tierlb.properties`

- In UNIX

  `/opt/Cosminexus/manager/vmi/templates/tierlb.properties`

(3) Functionality

This file configures connection information necessary to access the load balancer within a tier.

(4) Keys that can be specified

The keys that can be specified are shown below. The *Default value* column gives the value assumed if the key is omitted. *VR* indicates the version of the application server on which the keys are introduced or changed.

Key name	Description	Specifiable value	Default value	VR
`lb.type`	Specify the load balancer type.	The following values can be specified: • `BIG-IPv9` • `BIG-IPv10.1` • `BIG-IPv10.2` • `BIG-IPv11` • `ACOS`	None	08-53

Key name	Description	Specifiable value	Default value	VR
`lb.host`	Specify the management IP address of the load balancer to be connected.	IPv4 dot notation	None	08-53
`lb.protocol`	Specify the method of connecting to the load balancer. `API`: Uses the API to connect to the load balancer	The following value can be specified: • `API`	None	08-53
`lb.port`	Specify the port number to be used by the load balancer.	The following values can be specified: • API `1-65534`	• API: `443`	08-53
`lb.user`	Specify the user name used to connect to the load balancer.	Any character string can be used.	None	08-53
`lb.password`	Specify the user password used to connect to the load balancer.	Any character string can be used.	Null	08-53
`lb.persistence.cookie-insert.templatename`	Specify the cookie persistence template name created on the load balancer. The specified key becomes effective if the following conditions are met: • `lb.type=ACOS` and `lb.protocol=API` are specified in *LB-connection-information-distinguished-name*`.properties` or `tierlb.properties` • `lb.persistence.method=cookie-insert` is specified in `tier.properties`	A 1-13 character string consisting of alphanumeric characters and underscores (_)	None	08-70
`lb.timeout`	Specify the timeout period (in seconds) for login to the loader balancer or command transmission. If login to the load balancer or a CLI command issued to the load balancer is not complete within the time specified in this key, the `vmiunit` command will end abnormally due to a timeout error.	`1-2147483`	`10`	08-53

Key name	Description	Specifiable value	Default value	VR
`lb.API.protocol`	Specify the protocol to be used for communication with the load balancer. `http`: Uses the HTTP communication protocol. `https`: Uses the HTTPS communication protocol.	The following values can be specified: • `http` • `https`	`https`	09-00
`lb.ACOS.privilegedexec.password`	Specify the password necessary for the Privileged EXEC level set on ACOS. The specified key is only valid when the load balancer type is ACOS.	Any character string can be used.	None	08-53
`javax.net.ssl.trustStore`	Specify the trust store where the server certificate for the load balancer is registered. The specified key becomes effective if all of the following values are specified in *LB-connection-information-distinguished-name*`.properties` or `tierlb.properties`: • `lb.type=ACOS` • `lb.protocol=API` • `lb.API.protocol=https`	Follow the Java specifications.	cacerts (Default trust store for JDK)	08-70
`javax.net.ssl.trustStorePassword`	Specify the password for the trust store where the server certificate for the load balancer is registered. The specified key becomes effective if all of the following values are specified in *LB-connection-information-distinguished-name*`.properties` or `tierlb.properties`: • `lb.type=ACOS` • `lb.protocol=API` • `lb.API.protocol=https`	Follow the Java specifications.	cacerts (The default key store trusted by Java is used.)	08-70

Chapter

19. Messages Output by the Security Management Functionality

This chapter describes the messages output by the security management functionality.

19.1 Message description format

This chapter uses the following format to describe each message:

XXXXX*nnnn*-Y

`Message text`

(Meaning)

English message meaning

(Cause)

Cause for message output

(Action)

Action to be performed by the user

Each message consists of the following components:

`XXXXX`*nnnn*

Indicates the message ID.

A message ID consists of the following elements:

`XXXXX`

Indicates the ID (prefix) of the functionality that output the message. The message prefixes output by the Web service security functionality are as follows:

- `KDCGF`

  Indicates that an error occurred during reception of a SOAP message.

- `KDCGK`

  Indicates that processing by a secret key generation command ended normally or that an error occurred during execution of the command.

- `KDCGS`

  Indicates that an error occurred during reading of the Web service security functionality definition file.

- `KDCGW`

  Indicates that an error occurred during transmission of a SOAP message.

- `KEOS`

Indicates that an error occurred during setup, operation, or maintenance using Cosminexus Manager.

- `KEXS`

  Indicates that an error occurred in Cosminexus XML Security - Core.

nnnn

Indicates the message number managed by the program that output the message. Each message has a four-digit unique number.

Y

Indicates the message type. It is represented by a single letter.

The following message types might be output:

- `E`

  Indicates a message output when an error occurred. For details about how to handle this type of message, see *Action* for each message in *19.2* and succeeding sections.

- `I`

  Indicates a message informing that processing is complete. If this type of message is output, no action is required, because processing has ended normally.

- `W`

  Indicates a warning message. For details about how to handle this type of message, see *Action* for each message in *19.2* and succeeding sections.

Message text

Indicates the message text of a message output by the Web service security functionality. Web service security functionality messages are output in English.

(Meaning)

Describes the meaning of the English message.

(Cause)

Indicates the cause for the message that was output.

(Action)

Indicates an action to be taken by the user.

19.2 Messages starting with KDCGF

This section describes the messages between KDCGF0001 and KDCGF9999, which are output by the Web service security functionality.

Messages starting with KDCGF are output in SOAPFault format. A SOAPFault format message has the following four components:

(FaultCode)

In FaultCode, a FaultCode is output. The FaultCode consists of a name space URI and a local part. To the name space URI part of the FaultCode of a message starting with KDCGF, `{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd}` is output. To the local part, a character string indicating the cause of the error is output.

The value of FaultCode can be obtained in the following ways:

- Server side

  For SOAP 1.1, FaultCode can be obtained from the `faultcode` element of a SOAP Fault message.

  For SOAP 1.2, FaultCode can be obtained from the `soapenv12:Value` element included in the `soapenv12:Subcode` element (which is a child element of the `soapenv12:Code` element) of a SOAP Fault message. The value of the `soapenv12:Value` element of the `soapenv12:Code` element is soapenv12:Sender.

- Client side

  FaultCode can be obtained by using the C4Fault class provided by the SOAP Communication Infrastructure or the javax.xml.ws.soap.SOAPFaultException class provided by the JAX-WS functionality.

(FaultString)

In FaultString, a message ID or message text is output. For the meaning of message IDs, see *19.1 Message description format*.

FaultString can be obtained in the following ways:

- Server side

  For SOAP 1.1, FaultString can be obtained from the `faultString` element of a SOAP Fault message.

  For SOAP 1.2, FaultString can be obtained from the `soapenv12:Text` element of the `soapenv12:Reason` element of a SOAP Fault message.

- Client side

  FaultString can be obtained by using the C4Fault class provided by the SOAP Communication Infrastructure or the `javax.xml.ws.soap.SOAPFaultException` class provided by the JAX-WS functionality.

(FaultActor)

In FaultActor, the actor that generated the Fault is output.

FaultActor can be obtained in the following ways:

- Server side

  For SOAP 1.1, FaultActor can be obtained from the `faultactor` element of a SOAP Fault message.

  For SOAP 1.2, FaultActor can be obtained from the `soapenv12:Role` element of a SOAP Fault message.

- Client side

  FaultActor can be obtained by use of the C4Fault class provided by the SOAP Communication Infrastructure or the javax.xml.ws.soap.SOAPFaultException class provided by the JAX-WS functionality.

(FaultDetails)

In FaultDetails, details of the Fault are output.

FaultDetails can be obtained in the following ways:

- Server side

  For SOAP 1.1, FaultDetails can be obtained from the `detail` element of a SOAP Fault message.

  For SOAP 1.2, FaultDetails can be obtained from the `soapenv12:Detail` element of a SOAP Fault message.

- Client side

  FaultDetails can be obtained by use of the C4Fault class provided by the SOAP Communication Infrastructure or the javax.xml.ws.soap.SOAPFaultException class provided by the JAX-WS functionality.

KDCGF0001-E

```
FaultCode:
```

```
{http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-secext-1.0.xsd}UnsupportedSecurity
Token
FaultString: KDCGF0001-E An unsupported security token was
specified. (location = the-location-where-the-event-occurred)
FaultActor: None
FaultDetails: None
```

(Meaning)

An unsupported security token element has been used in *the-location-where-the-event-occurred*. The following information is output to *the-location-where-the-event-occurred*:

- `Server`: An error occurred in a message received at server side.
- `Client`: An error occurred in a message received on the client side.

(Cause)

The error might be due to one of the following causes:

- The `EncodingType` attribute of the `BinarySecurityToken` element is specified, but the attribute value is not `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary`.
- The `ValueType` attribute of the `BinarySecurityToken` element is specified, but the attribute value is not `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3`.
- The `EncodingType` attribute is specified in the `KeyIdentifier` element, but the attribute value is not "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary".
- The `ValueType` attribute is specified in the `KeyIdentifier` element, but the attribute value is not `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509SubjectKeyIdentifier`.
- The `ValueType` attribute is specified in the WS-Security `Reference` element, but the attribute value is not `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3`.
- A non-`Reference` or non-`KeyIdentifier` element is specified in a child element of the WS-Security `SecurityTokenReference` element.
- An XML encryption `Reference` element is specified in a child element of

the WS-Security `Security` element.

(Action)

Check with the sender of the message to see whether the sender has sent a SOAP message containing one of the errors indicated in *Cause*.

KDCGF0002-E

```
FaultCode: {http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-secext-1.0.xsd}UnsupportedAlgorith
m
FaultString: KDCGF0002-E An unsupported signature or encryption
algorithm was specified. (location =
the-location-where-the-event-occurred)
FaultActor: None
FaultDetails: None
```

(Meaning)

An unsupported signature or encryption algorithm has been used in *the-location-where-the-event-occurred*. The following information is output to *the-location-where-the-event-occurred*:

- `Server`: An error occurred in a message received at server side.
- `Client`: An error occurred in a message received on the client side.

(Cause)

The error might be due to one of the following causes:

- An unsupported algorithm is specified in the `Algorithm` attribute of the `Canonicalization` element.
- An unsupported algorithm is specified in the `Algorithm` attribute of the `SignatureMethod` element.
- An unsupported algorithm is specified in the `Algorithm` attribute of the `Transform` element.
- An algorithm that is not set in the Web service security policy definition file is specified in the `Algorithm` attribute of the `Canonicalization` element.
- An algorithm that is not set in the Web service security policy definition file is specified in the `Algorithm` attribute of the `SignatureMethod` element.
- An algorithm that is not set in the Web service security policy definition file is specified in the `Algorithm` attribute of the `Transform` element.
- An unsupported algorithm is specified in the `Algorithm` attribute of the

XML encryption `EncryptionMethod` element.

- An algorithm that is not set in the Web service security policy definition file is specified in the `Algorithm` attribute of the XML encryption `EncryptionMethod` element.

(Action)

Check with the sender of the message to see whether the sender has sent a SOAP message containing one of the errors indicated in *Cause*. Alternatively, check the Web service security policy definition file for incorrect settings.

KDCGF0003-E

```
FaultCode: {http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-secext-1.0.xsd}InvalidSecurity
FaultString: KDCGF0003-E An error occurred during security
header processing. (location = the-location-where-the-event-occurred)
FaultActor: None
FaultDetails: None
```

(Meaning)

An error occurred in the security header of *the-location-where-the-event-occurred*. The following information is output to *the-location-where-the-event-occurred*:

- `Server`: An error occurred in a message received at server side.
- `Client`: An error occurred in a message received on the client side.

(Cause)

The error might be due to one of the following causes:

- A `Created` element request is specified in the `Timestamp` element within the Web service security policy definition file, but the received SOAP message contains no `Created` element.
- An `Expires` element request is specified in the `Timestamp` element within the Web service security policy definition file, but the received SOAP message contains no `Expires` element.
- The `Created` and `Expires` elements in the received SOAP message have different xsd:dateTime values in the `ValueType` attribute.
- An element (`Created`, `Expires`, `BinarySecurityToken`, or `KeyIdentifier` element) that requires a value has no value.
- A `BinarySecurityToken` element request is specified in the Web service security policy definition file, but the received SOAP message contains no

`BinarySecurityToken` element.

- The `Reference` element has no `URI` attribute.
- No value is set in the `URI` attribute of the `Reference` element.
- A SOAP body signature request is specified in the Web service security policy definition file, but the SOAP body of the received SOAP message has no signature.
- An encrypted SOAP message has no `KeyInfo` element.
- A key specified in the `KeyName` element of an encrypted SOAP message is not defined in the Web service security functionality definition file.
- A SOAP body element encryption request is specified in the Web service security policy definition file, but the SOAP body of the received SOAP message has no encrypted element.
- A received SOAP message contains `ID` attributes that have the same attribute value.
- The `Name` and `My_role` attributes of the `ReceiverPortConfig` element in the Web service security policy definition file have no corresponding setting in the Web service security functionality definition file.

(Action)

Check with the sender of the message to see whether the sender has sent a SOAP message containing one of the errors indicated in *Cause*. Alternatively, check the Web service security policy definition file for incorrect settings.

KDCGF0004-E

```
FaultCode: {http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-secext-1.0.xsd}InvalidSecurityToke
n
FaultString: KDCGF0004-E An invalid security token was
specified. (location = the-location-where-the-event-occurred)
FaultActor: None
FaultDetails: None
```

(Meaning)

An invalid security token has been used for *the-location-where-the-event-occurred*. The following information is output to *the-location-where-the-event-occurred*:

- `Server:` An error occurred in a message received at server side.
- `Client:` An error occurred in a message received on the client side.

(Cause)

The error might be due to one of the following causes:

- The `BinarySecurityToken` element has no `ValueType` attribute.
- Verification of `BinarySecurityToken` elements in received SOAP messages using the certificate file defined in the Web service security policy definition file always fails.

(Action)

Check with the sender of the message to see whether the sender has sent a SOAP message containing one of the errors indicated in *Cause*. Alternatively, check the Web service security policy definition file for incorrect settings.

KDCGF0005-E

```
FaultCode: {http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-secext-1.0.xsd}FailedAuthenticatio
n
FaultString: KDCGF0005-E A security token could not be
authenticated or authorized. (location =
the-location-where-the-event-occurred)
FaultActor: None
FaultDetails: None
```

(Meaning)

The security token in *the-location-where-the-event-occurred* cannot be authenticated or authorized. The following information is output to *the-location-where-the-event-occurred*:

- `Server`: An error occurred in a message received at server side.
- `Client`: An error occurred in a message received on the client side.

(Cause)

See the *Cause* item for *KDCGJ0001-E* in the manual *uCosminexus Application Server Messages*.

(Action)

See the *Action* item for *KDCGJ0001-E* in the manual *uCosminexus Application Server Messages*.

KDCGF0006-E

```
FaultCode: {http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-secext-1.0.xsd}FailedCheck
```

```
FaultString: KDCGF0006-E A signature or decryption was invalid.
(location = the-location-where-the-event-occurred)
FaultActor: None
FaultDetails: None
```

(Meaning)

A signature or encryption in *the-location-where-the-event-occurred* is invalid. The following information is output to *the-location-where-the-event-occurred*:

- `Server`: An error occurred in a message received at server side.
- `Client`: An error occurred in a message received on the client side.

(Cause)

The error might be due to one of the following causes:

- A received SOAP message has an invalid signature.
- A received SOAP message is incorrectly encrypted.

(Action)

Check with the sender of the message to see whether the sender has sent a SOAP message containing one of the errors indicated in *Cause*.

KDCGF0007-E

```
FaultCode: {http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-secext-1.0.xsd}SecurityTokenUnavai
lable
FaultString: KDCGF0007-E A referenced security token cannot be
found. (location = the-location-where-the-event-occurred)
FaultActor: None
FaultDetails: None
```

(Meaning)

The referenced security token element could not be found in a SOAP message received at *the-location-where-the-event-occurred*. The following information is output to *the-location-where-the-event-occurred*:

- `Server`: An error occurred in a message received at server side.
- `Client`: An error occurred in a message received on the client side.

(Cause)

- A `BinarySecurityToken` element specified in the WS-Security `Reference` element was not found.
- An X.509 certificate with a subject key identifier specified in the

WS-Security `KeyIdentifier` element was not found in the key store file specified in the `VerificationKeyStore` element within the Web service security functionality definition file.

(Action)

Check with the sender of the message to see whether the sender has sent a SOAP message containing one of the errors indicated in *Cause*.

19.3 Messages starting with KDCGK

This section describes the messages between KDCGK0001 and KDCGK9999, which are output by the Web service security functionality.

KDCGK0001-I

`Generation of a secret key has finished. (file =` *file-name*`)`

(Meaning)

Generation of a secret key is complete.

The generated secret key file name is output to *file-name*.

KDCGK0010-E

`An argument is not specified. (argument =` *argument*`)`

(Meaning)

An argument is not specified.

The name of the argument to be specified is output to *argument*.

(Cause)

An argument that needs to be specified in the secret key generation command is not specified.

(Action)

Specify the argument and then retry the secret key generation command.

KDCGK0011-E

`An invalid argument is specified. (argument =` *argument*`)`

(Meaning)

An invalid argument is specified.

The name of the argument deemed invalid is output to *argument*.

(Cause)

An invalid argument that is not allowed to be used in the secret key generation command is specified.

(Action)

Delete the argument that was deemed invalid and retry the secret key generation command.

KDCGK0012-E

`An invalid argument value is specified. (argument =` *argument*`;`
`value =` *value-of-the-argument*`)`

(Meaning)

The value specified in the argument is invalid.

The following information is output to *argument* and *value-of-the-argument* respectively:

argument

The name of the argument is output.

value-of the-argument

The value specified in the argument is output.

(Cause)

An invalid value is specified in an argument for the secret key generation command.

(Action)

Check the secret key generation command to see whether the specified arguments and argument values are correct, and retry the secret key generation command.

KDCGK0013-E

`A specified file already exists. (file =` *file-name*`)`

(Meaning)

The specified secret key file already exists.

(Cause)

A file with the file name specified by the `-o` option of the secret key generation command already exists.

(Action)

Specify a different file name that is not used for an existing file, and retry the secret key generation command.

KDCGK0100-E

`An error occurred during output of the key to a file. (details = `*details*`)`

(Meaning)

An error occurred during file output.

Detailed error information is output to *details*.

(Cause)

An error occurred when outputting to file a secret key generated by the secret key generation command.

(Action)

Solve the cause of the error indicated in *details*, and retry the secret key generation command. If you are not sure of the cause of the error indicated in *details*, contact the system administrator.

KDCGK0101-E

`An error occurred during key creation. (details = `*details*`)`

(Meaning)

An error occurred during key generation.

Detailed error information is output to *details*.

(Cause)

An error occurred when using the secret key generation command to generate a secret key.

(Action)

Solve the cause of the error indicated in *details*, and retry the secret key generation command. If you are not sure of the cause of the error indicated in *details*, contact the system administrator.

KDCGK9000-E

`An unexpected error occurred during processing. (details = `*details*`)`

(Meaning)

An unexpected error occurred during processing.

Detailed error information is output to *details*.

(Cause)

An error of unknown cause occurred during execution of the secret key generation command.

(Action)

Contact the system administrator.

19.4 Messages starting with KDCGS

This section describes the messages between KDCGS0001 and KDCGS9999, which are output by the Web service security functionality.

KDCGS0005-E

`A URI or TargetId attribute value must start with #. (tag name =` *element-name*`; attribute name =` *attribute-name*`)`

(Meaning)

A `URI` or `TargetId` attribute must start with #.

The following information is output to *element-name* and *attribute-name* respectively:

element-name

The name of an element that has an attribute for which the specified value does not start with # is output.

attribute-name

The name of an attribute for which the specified value does not start with # is output.

(Cause)

The Web service security functionality definition file contains an attribute with a specified value that does not start with #.

(Action)

Correct the Web service security functionality definition file and add a hash mark (#) to the beginning of the value specified in the *attribute-name* attribute.

KDCGS0007-E

`The specified secret key file does not match the KeyType attribute. (secret key file =` *secret-key-file-name*`; KeyType =` *algorithm-identifier*`)`

(Meaning)

The secret key file does not match the information specified in the `keytype` attribute.

The following information is output to *secret-key-file-name* and *algorithm-identifier* respectively:

secret-key-file-name

The secret key file name is output.

algorithm-identifier

The algorithm identifier specified by the `keytype` attribute is output.

(Cause)

The algorithm identifier specified by the `keytype` attribute of the `SecretKeyFile` element within the Web service security functionality definition file does not match the contents of the secret key file.

(Action)

Correct the Web service security functionality definition file so that the information specified in the `keytype` attribute of the `SecretKeyFile` element matches the contents of the secret key file. The `keytype` attribute must contain the same algorithm identifier as that specified in the secret key generation command argument.

KDCGS0008-E

`An error occurred during creation of a secret key. (details =` *details*`)`

(Meaning)

An error occurred during secret key generation.

Detailed error information is output to *details*.

(Cause)

An error occurred during generation of a secret key from the secret key file specified by the `SecretKeyFile` element within the Web service security functionality definition file. The error might be due to one of the following causes:

- The `SecretKeyFile` element is specified incorrectly.
- The runtime environment is configured incorrectly.

(Action)

According to the contents of *details*, check the Web service security functionality definition file for incorrect settings, or check the runtime environment.

KDCGS0009-E

`The EmbedId attribute value is duplicated.`

(Meaning)

A duplicate `EmbedId` attribute value exists.

(Cause)

A duplicate `EmbedId` attribute value is specified in the Web service security functionality definition file.

(Action)

Correct the Web service security functionality definition file so that there are no duplicate `EmbedId` attribute values.

KDCGS0014-E

`An error occurred during reading of a KeyStore. (details =` *details*`)`

(Meaning)

An error occurred during reading of a key store. Detailed error information is output to *details*.

(Cause)

An error occurred when reading the key store file specified in the `File` attribute of the `KeyStore` element within the Web service security functionality definition file. The specified key store file might not exist, the user might not have access permission for the file, or the file format might be incorrect.

(Action)

According to the contents of *details*, check the specification of the `File` attribute of the `KeyStore` element in the Web service security functionality definition file.

KDCGS0015-E

`A definition is duplicated. (tag name =` *element-name*`)`

(Meaning)

A duplicate element exists. The duplicate element name is output to *element-name*.

(Cause)

An *element-name* that is allowed to be specified only once is specified more than once in the Web service security functionality definition file.

(Action)

Correct the Web service security functionality definition file so that the *element-name* is specified only once.

19.5 Messages starting with KDCGW

This section describes the messages between KDCGW0001 and KDCGW9999, which are output by the Web service security functionality.

KDCGW0002-E

```
FaultCode: {http://www.hitachi.co.jp/soft/xml/cosminexus/ws/
security/0760/FaultCode}<Server.SigningError or
Client.SigningError>
FaultString: KDCGW0002-E An error occurred during message
signature processing. (details = details)
FaultActor: None
FaultDetails: None
```

(Meaning)

An error occurred during message signature processing.

The following information is output to *details* and <*Server.SigningError or Client.SigningError*>:

<*Server.SigningError or Client.SigningError*>

This character string indicates whether the error occurred on the server side or the client side. If the error occurred on the server side, *Server.SigningError* is output. If the error occurred on the client side, *Client.SigningError* is output.

details

Detailed error information is output.

(Cause)

The error might be due to one of the following causes:

- The algorithm specified in the `CanonicalizationMethod`, `SignatureMethod`, or `Transform` element within the Web service security functionality definition file is incorrect.
- The signature target specified in the `SignatureTarget` element within the Web service security functionality definition file is incorrect.

(Action)

Solve the cause of the error indicated in *details*, and retry the processing. Before retrying the processing, it is necessary to re-deploy the SOAP application and Web service.

If you are not sure of the cause of the error indicated in *details*, contact the system administrator.

KDCGW0003-E

```
FaultCode: {http://www.hitachi.co.jp/soft/xml/cosminexus/ws/
security/0760/FaultCode}<Server.EncryptingError or
Client.EncryptingError>
FaultString: KDCGW0003-E An error occurred during message
encryption. (details = details)
FaultActor: None
FaultDetails: None
```

(Meaning)

An error occurred during message encryption.

The following information is output to *details* and <*Server.EncryptingError or Client.EncryptingError*>:

<*Server.EncryptingError or Client.EncryptingError*>

This character string indicates whether the error occurred the server side or the client side. If the error occurred on the server side, *Server.EncryptingError* is output. If the error occurred on the client side, *Client.EncryptingError* is output.

details

Detailed error information is output.

(Cause)

The error might be due to one of the following causes:

- The algorithm specified in the `EncryptionMethod` child element of the `ContentsEncryption` or `KeyEncryption` element within the Web service security functionality definition file is incorrect.
- The encryption target specified in the `EncryptionTarget` element within the Web service security functionality definition file is incorrect.

(Action)

Solve the cause of the error indicated in *details*, and retry the processing. Before retrying the processing, it is necessary to re-deploy the SOAP application and Web service.

If you are not sure of the cause of the error indicated in *details*, contact the system administrator.

19.6 Messages from KEOS02000 to KEOS09999

This section describes the messages between KEOS02000 and KEOS09999, which are output during setup, operation, or maintenance when using Cosminexus Manager.

KEOS02020-E (C)

`Loading of a shared library failed. Library name =` *aa....aa*

aa....aa: Library name

(Description)

The processing will be stopped.

(Action)

Check to see the following, and then retry:

- `java.library.path` is specified as a JavaVM start option.
- If the JavaVM start option is specified correctly, Keymate/Crypto is installed.

  If Keymate/Crypto is installed, available memory might be insufficient. Contact the administrator of the host that performed the processing to solve the insufficient memory problem.

If none of the above is the cause of the error, the installation might be incomplete. (A required file is missing or damaged.) Uninstall and then re-install User Management.

KEOS02102-E (C)

`Encryption of the specified SecretData failed.`

(Description)

Encryption of the specified SecretData failed or the encryption key file could not be accessed.

The processing will be stopped.

(Action)

Check the following, and then retry:

- An encryption key (`com.cosminexus.admin.auth.sso.keyfile`) is specified in the user management configuration file.
- The encryption key file exists.

- The user has access permissions for the key file.

KEOS02152-E (C)

`Decryption of the specified SecretData failed.`

(Description)

Decryption of the specified SecretData failed or the encryption key file could not be accessed.

The processing will be stopped.

(Action)

Check to see the following, and then retry:

- An encryption key (`com.cosminexus.admin.auth.sso.keyfile`) is specified in the user management configuration file.
- The encryption key file exists.
- The user has access permissions for the key file.
- The specified encryption key file is not the same as that used in registration.

KEOS02202-E (C)

`An encryption key file could not be accessed.`

(Description)

No encryption key file is specified in the user management configuration file or the user has no write permission for the directory to create the encryption key file.

The processing will be stopped.

(Action)

Check `com.cosminexus.admin.auth.sso.keyfile` within the user management configuration file to make sure the following is specified, and retry:

- An encryption key file name is specified.
- If a file name is specified, the user has access permissions for the directory.

KEOS02300-E (C/F)

`Password decryption failed. Details =` *aa....aa*

aa....aa: Details

(Description)

Password decryption failed. The pre-decryption character string will be used as the password.

The processing will be continued with the default settings.

(Action)

Check to see whether the password is scrambled.

KEOS13105-E (W/F)

`Creation of an encryption key file failed. Details =` *aa....aa*

aa....aa: Exception code

(Description)

The encryption key file could not be applied to the configuration information.

A **Back** link is displayed. Clicking the **Back** link will take the user to the window for configuring an encryption key file.

(Action)

Make sure an encryption key file exists in the specified directory. Also, make sure that the user has read permission for the encryption key file and the directory in which the key file resides.

KEOS13106-E (W/F)

`Loading of a shared library failed. Details =` *aa....aa*

aa....aa: Exception code

(Description)

Loading of a shared library failed.

A **Back** link is displayed.

Clicking the **Back** link will take the user to the Calling window.

(Action)

Check to see whether Keymate/Crypto is installed. If Keymate/Crypto is installed, available memory might be insufficient. Contact the administrator of the host that performed the processing to solve the insufficient memory problem.

If none of the above is the cause of the error, a file necessary for installation might be missing or damaged. Uninstall and then re-install the product.

KEOS13107-E (W/F)

`Encryption of SecretData failed. Details =` *aa....aa*

aa....aa: Exception code

(Description)

Encryption of SecretData failed.

A **Back** link is displayed.

Clicking the **Back** link will take the user to the Calling window.

(Action)

Make sure an encryption library is installed.

KEOS13119-I (W/F)

`The settings in the encryption key file were applied.`

(Description)

An encryption key file was created or applied.

A **Back** link is displayed.

(Action)

Click the **Back** link. Clicking the **Back** link will take you to the window for configuring an encryption key file.

KEOS13125-E (W/F)

`Input information contains an invalid character. Input information =` *aa....aa*

aa....aa: Input information

(Description)

Invalid input information.

A **Back** link is displayed.

Clicking the **Back** link will take the user to the Calling window.

(Action)

Check the input information for the following:

- If the input information is a realm name, it should be a character string consisting of alphabetic characters (A to Z and a to z) and numeric characters

(0 to 9) only. The reserved character string *mappings* cannot be used.

- If the input information is a user ID, it should be a character string consisting of alphabetic characters (A to Z and a to z) and numeric characters (0 to 9) only.
- If the input information is a password or SecretData, it should be a character string consisting of alphabetic characters (A to Z and a to z), numeric characters (0 to 9), and special characters. Special characters are the following symbols:

  (space) | ! | " | # | $ | % | & | ' | (|) | * | + | , | - | . | / | : | ; | < | = | > | ? | @ | [| \ |] | ^ | _ (underscore) | ` | { | } | | (vertical bar) | ~

KEOS13126-E (W/F)

```
SecretData does not match.
```

(Description)

The entered SecretData does not match the re-entered SecretData.

A **Back** link is displayed.

Clicking the **Back** link will take the user to the Calling window.

(Action)

Enter the SecretData again.

19.7 Messages starting with KEXS

This section describes the messages between KEXS10001 and KEXS99999, which are output by Cosminexus XML Security - Core.

KEXS20006-E

`The JCE provider was not found. Provider name =` *{0}*

(Cause)

A Java security provider is not set correctly.

{0}: Provider name

(Processing)

An exception is issued.

(Action)

Check the `java.security` file for incorrect settings.

KEXS20007-E

`A JCE algorithm was not found. Algorithm name =` *{0}*`; provider name =` *{1}*

(Cause)

The Java security provider is not set correctly.

{0}: Algorithm name

{1}: Provider name

(Processing)

An exception is issued.

(Action)

Check the `java.security` file for incorrect settings.

KEXS20008-E

`Invalid DSA ASN.1 format.`

(Cause)

The DSA ASN.1 format is invalid.

(Processing)

An exception is issued.

(Action)

Check the `java.security` file for incorrect settings.

KEXS20009-E

`The specified implementation class is invalid. Class name =` *{0}*

(Cause)

The specified implementation class is invalid.

{0}: Class name

(Processing)

An exception is issued.

(Action)

Specify a valid implementation class.

KEXS30010-E

`No parent node exists.`

(Cause)

An attempt was made to replace EncryptedData that does not have a parent node in decryption mode.

(Processing)

An exception is issued.

(Action)

Specify EncryptedData that has a parent node.

KEXS30011-E

`Invalid DOMFragment.`

(Cause)

The DOMFragment nodes have no parent or have no siblings.

(Processing)

An exception is issued.

(Action)

Specify a DOMFragment for which the nodes have a parent and siblings.

KEXS40001-E

`Invalid mode.`

(Cause)

The set mode is invalid for the attempted processing.

(Processing)

An exception is issued.

(Action)

Set a valid mode.

KEXS40002-E

`The algorithm was not found. Category =` *{0}*`; algorithm identifier =` *{1}*

(Cause)

The specified algorithm is invalid.

{0}: Category

{1}: Algorithm identifier

(Processing)

An exception is issued.

(Action)

Specify a valid algorithm.

KEXS40003-E

`No key resolver is set.`

(Cause)

No key resolver is set.

(Processing)

An exception is issued.

(Action)

Set a key resolver.

KEXS40004-E

`An invalid algorithm parameter is specified.`

(Cause)

An invalid algorithm parameter is specified.

(Processing)

An exception is issued.

(Action)

Specify a valid algorithm parameter.

KEXS40009-E

`Key resolution failed.`

(Cause)

The key could not be obtained during key resolution.

(Processing)

An exception is issued.

(Action)

Set an appropriate key resolver so that a key can be obtained.

KEXS40010-E

`An invalid object is included.`

(Cause)

List or Map contains an invalid object.

(Processing)

An exception is issued.

(Action)

Do not include any invalid objects.

KEXS40011-E

`The usage of the enveloped-signature transform or XPath function here() is not correct.`

(Cause)

No node is bound to the enveloped-signature transform or XPath function `here()`.

(Processing)

An exception is issued.

(Action)

Check to see whether the usage of the enveloped-signature transform or XPath function `here()` is correct.

KEXS40012-E

`No output destination is set.`

(Cause)

No output destination is set for XMLSerializer.

(Processing)

An exception is issued.

(Action)

Set an output destination.

KEXS40013-E

`The result is not well-formed. Mode =` *{0}*

(Cause)

A replacement that would produce the following results was attempted:

- Two document elements result.
- An element appears before a DocumentType node.
- The document node contains a non-comment and non-PI subnode.

{0}: Mode

(Processing)

An exception is issued.

(Action)

Correct the replacement so that the result becomes well-formed.

KEXS40014-E

`The specified key has an invalid key length. Key length =` *{0}*

(Cause)

The `Key` object has an invalid key length.

{0}: Key length

(Processing)

An exception is issued.

(Action)

Specify a key with a valid key length.

KEXS40015-E

`The specified key agreement context is invalid.`

(Cause)

The key agreement context is invalid.

(Processing)

An exception is issued.

(Action)

Specify a valid key agreement context.

KEXS40016-E

`The specified node is at an invalid location. Mode =` *{0}*

(Cause)

During encryption, an attempt was made to replace EncryptedData with an instance of DOMFragment that is its descendant. Or, during decryption, an attempt was made to replace DOMFragment with an instance of EncryptedData that is its descendant or that has a sibling relationship with DOMFragment.

{0}: Mode

(Processing)

An exception is issued.

(Action)

Specify instances of EncryptedData or DOMFragment that are in valid locations.

KEXS50001-E

`The specified element is invalid. Requested element =` *{0}*`;`
`specified element =` *{1}*

(Cause)

The specified element is invalid. The name space URI or local name contains an error.

{0}: Requested element

{1}: Specified element

(Processing)

An exception is issued.

(Action)

Specify a valid element.

KEXS50002-E

`No KeyInfo content can be created. Name space URI =` *{0}*`; local`
`name =` *{1}*

(Cause)

The specified KeyInfo content is invalid.

{0}: Name space URI

{1}: Local name

(Processing)

An exception is issued.

(Action)

Specify valid KeyInfo content.

KEXS50003-E

`No algorithm parameter is set. Algorithm identifier =` *{0}*

(Cause)

No algorithm parameter is set.

{0}: Algorithm identifier

(Processing)

An exception is issued.

(Action)

Set an algorithm parameter.

KEXS50014-E

`JCE algorithm processing failed. Class name =` *{0}*`; method name =` *{1}*`; provider name =` *{2}*

(Cause)

JCE algorithm processing failed or an invalid value might be set for the KeyInfo content.

{0}: Class name

{1}: Method name

{2}: Provider name

(Processing)

An exception is issued.

(Action)

Check the `java.security` file for incorrect settings.

KEXS50015-E

`Integer format error. Text =` *{0}*

(Cause)

The set text cannot be recognized as an integer.

{0}: Text

(Processing)

An exception is issued.

(Action)

Set text that is recognized as an integer.

KEXS50016-E

`Invalid DSA XML signature format.`

(Cause)

The `SignatureValue` element has an invalid DSA signature value.

(Processing)

An exception is issued.

(Action)

Set a valid DSA signature value.

KEXS50017-E

`No corresponding DOM node exists.`

(Cause)

An attempt was made to apply the decryptXML method to EncryptedData that was not created from an existing Element node, or a replacement attempt was made in decryption mode.

(Processing)

An exception is issued.

(Action)

Use EncryptedData created with the `newEncryptedData` method (`XMLSecurityContext` or `Element`) of the `XMLEncryptionFactory` class.

KEXS50018-E

`An invalid value is set in the KeySize element. KeySize =` *{0}*

(Cause)

An invalid value is set for the `KeySize` element.

{0}: `KeySize` element value

(Processing)

An exception is issued.

(Action)

Set a valid value.

KEXS50019-E

`Invalid data size. Data size =` *{0}*

(Cause)

Data input to the specified algorithm is too short or is not a multiple of the block length specified for the algorithm.

{0}: Data size

(Processing)

An exception is issued.

(Action)

Specify valid input data.

KEXS50020-E

`Integrity checking failed. Algorithm identifier =` *{0}*

(Cause)

The input data or key is invalid.

{0}: Algorithm identifier

(Processing)

An exception is issued.

(Action)

Specify valid input data or a valid key.

KEXS50021-E

`A parameter necessary for key generation is not set.`

(Cause)

A parameter necessary for key generation is not set.

(Processing)

An exception is issued.

(Action)

Set the necessary parameter.

KEXS50022-E

`The decryption result is null.`

(Cause)

The encrypted data is null. Processing with the `decryptXML` method is not possible.

(Processing)

An exception is issued.

(Action)

Specify EncryptedData containing valid data.

KEXS50023-E

`Invalid padding. Algorithm identifier =` *{0}*

(Cause)

The input data or key is invalid.

{0}: Algorithm identifier

(Processing)

An exception is issued.

(Action)

Specify valid input data or a valid key.

19.8 SSL-related messages

This section describes messages output by Cosminexus HTTP Server's SSL processing.

19.8.1 Message description format

This section uses the following format to describe each message:

`Message text`

Message description

Error level: The level of the error output to the error log

(S) System processing

(O) User action

Depending on the message, no *(O)* error level is given.

Each message consists of the following components:

Message text

There are two types of message. One type consists of a time and an error level followed by the message text. The other type consists of the message text only. The formats of these message types are shown below.

Format 1
`[`*time*`] [`*error-level*`]` *message-text*

Format 2
message-text

The *details* part of the message text description provides information about the error, including an (error code) character string.

Message description

Indicates additional information about the message, such as the cause of the message issuance.

Error level

Indicates an error level specified in the LogLevel directive.

The error levels that might be output are as follows:

- `emerg` level

- `alert` level
- `crit` level
- `error` level
- `warn` level
- `notice` level
- `info` level
- No error level

If a message has no error level, then no error level has been set for it, and the message consists only of message text.

(S)

Indicates the main processing performed by the system after the message output.

(O)

Indicates an action to be performed by the user when the message is output.

19.8.2 Notes

Notice-level messages are output regardless of the specification of the LogLevel directive.

Before analyzing the level specification, it is important to note that messages might be output regardless of the specification of the LogLevel directive. For example, messages might be output while Cosminexus HTTP Server is starting.

With some exceptions, the following messages are not described:

- Messages involving a config file syntax error, which are output while Cosminexus HTTP Server is starting
- Debug error level messages output after Cosminexus HTTP Server starts
- Messages without an error level that are output after Cosminexus HTTP Server starts

19.8.3 Message details

`allocate error`

Memory allocation necessary for SSL processing failed.

Error level: `error`

(S) Cannot connect via SSL.

(O) Check system resource usage.

`[client` *client-address*`] [port` *client-port-number*`] allocate error`

Memory allocation necessary for SSL processing failed.

Error level: `error`

(S) Cannot connect via SSL.

(O) Check system resource usage.

`Attempt to reinitialise SSL for server` *host-name*

An attempt was made to re-initialize the host setting.

Error level: `crit`

(S) Does not start the Web server.

(O) Check the SSL settings in the virtual host. (At least one SSL-related directive must be set.)

`Bad password for the private key`

No correct password could be read from the password file specified in the SSLCertificateKeyPassword directive.

Error level: `crit`

(S) Does not start the Web server.

(O) Set the correct password in the password file.

`Can't open certificate file` *Web-server-certificate-file*

No certificate can be read.

Error level: `crit`

(S) Does not start the Web server.

(O) Check the value set in the SSLCertificateFile directive.

`Could not get lastUpdate field in CRL:` *file-name*

The CRL issuance date could not be obtained.

Error level: `crit`

(S) Does not start the Web server.

(O) Check to see whether the CRL was created or downloaded correctly.

`Could not load the certificate file.`

The server certificate file failed to load.

Error level: `crit`

(S) Does not start the Web server.

(O) Check the value set in the SSLCertificateFile directive.

`Could not read the private key file.`

The server private key file could not be read.

Error level: None

(S) Stops the processing.

(O) Check the value set in the SSLCertificateKeyFile directive.

`Could not set up a new lock.`

Initialization of lock processing failed.

Error level: `crit`

(S) Does not start the Web server.

(O) Check system resource usage.

`CRL expired, but CRL check passed: issuer=`*issuer-name*`,`
`serial=`*serial-number*

Because the SSLCRLAuthoritative directive was set to Off, client access was permitted, even though the next CRL issuance date was passed and the CRL contained no serial number information for the client certificate.

Error level: `warn`

(S) Continues processing.

(O) Update the CRL.

`CRL expired, but serial number was found in CRL:`
`issuer=`*issuer-name*`, serial=`*serial-number*

Because the CRL contained serial number information for the client certificate, an SSL handshake failed even though the next CRL issuance date was passed.

Error level: `error`

(S) Denies access due to the SSL handshake failure.

(O) Update the CRL.

`CRL expired: issuer=`*issuer-name*

An SSL handshake failed because the next CRL issuance date was passed.

Error level: `error`

(S) Denies access due to the SSL handshake failure.

(O) Update the CRL.

`CRL expired:` *file-name*

A CRL for which the next issuance date had been passed was read.

Error level: `warn`

(S) Continues processing. When authenticating a client, the system performs processing according to the value set in the SSLCRLAuthoritative directive.

(O) Obtain a new CRL.

`CRL is a duplicate:` *file-name* `will not be used in server` *host-name:port-number*

Two or more CRLs that were issued by a single CA (so had the same subject) were read. The displayed CRLs will not be used.

Error level: `warn`

(S) Continues processing.

(O) If two or more CRLs with the same subject are issued, store only one of them in the directory.

`CRL is not a valid type:` *file-name*

An unexpected file was read from the directory specified in the SSLCRLDERPath or SSLCRLPEMPath directive.

Error level: `crit`

(S) Does not start the Web server.

(O) A non-CRL file cannot be stored in the directory. Check the CRL file format. If necessary, convert the file format and store the file in the appropriate directory.

`CRL is not valid: issuer=`*issuer-name*

Invalid CRL.

Error level: `error`

(S) Denies access due to the SSL handshake failure.

(O) Check to see whether the CRL is created correctly.

`CRL is not yet valid: issuer=`*issuer-name*

Authentication of the client certificate failed because the CRL issuance date was earlier than the current time.

Error level: `error`

(S) Denies access due to the SSL handshake failure.

(O) Check the time settings of the system.

`CRL is not yet valid:` *file-name*

The time set in the CRL issuance date is later than the current time.

Error level: `warn`

(S) Continues processing.

(O) Check the system time settings.

`CRL verify error: issuer=`*issuer-name*

Verification of the CRL signature failed.

Error level: `error`

(S) Fails to authenticate the client certificate.

(O) Check whether the correct CRL has been read.

`data set error`

SSL initialization failed.

Error level: `error`

(S) Cannot connect via SSL.

(O) Check system resource usage.

`[client` *client-address*`] [port` *client-port-number*`] data set error`

SSL initialization failed.

Error level: `error`

(S) Cannot connect via SSL.

(O) Check system resource usage.

`Depth of certificate chain (`*CA-certificate-depth*`) exceeded SSLExportCertChainDepth limit: subject=(`*CA-certificate-subject*`)`

Certificate verification succeeded, but setting and storing of the environment variable with the value set in the SSLExportCertChainDepth directive was canceled because the value set in the directive was 1 or higher and a certificate chain exceeding the set value was sent from the client.

This error message is output for each CA certificate exceeding the set value.

Example: If a certificate chain (exceeding a client certificate) has three levels and the value set in the SSLExportCertChainDepth directive is 1, the above error message will be output twice.

Error level: `warn`

(S) Continues processing.

`Error reading server certificate file` *file-name*

No certificate can be read.

Error level: `crit`

(S) Does not start the Web server.

(O) Check to see whether the certificate format is correct.

`error setting verify locations`

The path name specified in the SSLCACertificateFile or SSLCACertificatePath directive cannot be set.

Error level: `crit`

(S) Does not start the Web server.

(O) Check the setting of the SSLCACertificateFile or SSLCACertificatePath directive.

`Failed to stack CRL in ReadCRL()`

Failed to store data.

Error level: `crit`

(S) Does not start the Web server.

(O) Restart the Web server.

`Malloc error in GetCertificateAndKey()`

Memory allocation failed.

Error level: `crit`

(S) Does not start the Web server.

(O) Check system resource usage.

`Malloc error in GetPrivateKey()`

Memory allocation failed.

Error level: `crit`

(S) Does not start the Web server.

(O) Check system resource usage.

`Malloc error in SetupLock()`

Memory allocation failed.

Error level: `crit`

(S) Does not start the Web server.

(O) Check system resource usage.

`malloc failed in CRLCheck()`

Memory allocation necessary for processing failed.

Error level: `error`

(S) Fails to authenticate the client certificate.

(O) Check the amount of memory used by the system.

`malloc failed in GetCRL()`

Memory allocation necessary for processing failed.

Error level: `crit`

(S) Does not start the Web server.

(O) Check the amount of memory used by the system.

`malloc failed in ReadCRL()`

Memory allocation necessary for processing failed.

Error level: `crit`

(S) Does not start the Web server.

(O) Check the amount of memory used by the system.

`No client certificate`

No client certificate was sent.

Error level: `error`

(S) Stops the SSL request processing.

`[client` *client-address*`] [port` *client-port-number*`] No client certificate`

No client certificate was sent.

Error level: `error`

(S) Stops the SSL request processing.

`No SSL Certificate set for server` *host-name*`:`*port*

No Web server certificate is set.

Error level: `crit`

(S) Does not start the Web server.

(O) Set the SSLCertificateFile directive.

`Required SSLCacheServerPath missing. gcache will not be started.`

No gcache server can be started, because no SSLCacheServerPath directive is specified.

Error level: `error`

(S) Does not start a gcache server.

(O) To use the session management functionality, specify a SSLCacheServerPath directive.

`Required SSLCacheServerPort missing. gcache will not be started.`

The gcache server cannot be started because no SSLCacheServerPort directive is specified.

Error level: `error`

(S) Does not start the gcache server.

(O) To use the session management functionality, specify an SSLCacheServerPort directive.

`Required SSLCertificateKeyPassword missing.`

No SSLCertificateKeyPassword directive is set.

Error level: `crit`

(S) Does not start the Web server.

(O) Set the SSLCertificateKeyPassword directive.

`Serial number was found in CRL: issuer=`*issuer-name*`,`
`serial=`*serial-number*

An SSL handshake failed because the CRL contained serial number information for a client certificate.

Error level: `error`

(S) Denies access due to the SSL handshake failure.

`Set error in GetCertificateAndKey()`

SSL initialization failed.

Error level: `crit`

(S) Does not start the Web server.

(O) Check system resource usage.

`SSLExportCertChainDepth is outside the appropriate range`

The SSLExportCertChainDepth directive has an invalid value that is not allowed to be specified.

Error level: None

(S) Does not start the Web server.

(O) Check the value specified in the directive.

`[client `*client-address*`]` *details*`: SSL handshake interrupted by system:`
`client port `*port-number*

SSL handshake processing did not end normally.

Error level: `info`

(S) Stops the SSL request processing.

`[client` *client-address*`]` *details*`: SSL handshake interrupted by system: client port` *port-number* `(`*SSL-handshaking-time*`) (`*error-number*`) (`*server-process-ID*`):`*SSL-handshake-status*

SSL handshake processing did not end normally.

Error level: `info`

(S) Stops the SSL request processing.

`[client` *client-address*`] SSL library error` *error-number* `in handshake`

SSL handshake processing did not end normally.

Error level: `error`

(S) Stops the SSL request processing.

`[client` *client-address*`] [port` *client-port-number*`] SSL library error` *error-number* `in handshake (`*SSL-handshaking-time*`) (`*error-number*`) (`*server-process-ID*`):`*SSL-handshake-status*

SSL handshake processing did not end normally.

Error level: `error`

(S) Stops the SSL request processing.

`SSL Library Error:` *details*

An error occurred in the SSL library.

Error level: `crit` or `error`

(S) If the Web server is starting, the system stops the start processing. If an SSL request is being processed, the system stops the SSL request processing.

(O) Check the details.

`SSLSessionCacheSize is outside the appropriate range`

The SSLSessionCacheSize directive has an invalid value that is not allowed to be specified.

Error level: None

(S) Does not start the Web server.

(O) Check the value specified in the directive.

`SSLSessionCacheSizePerChild is outside the appropriate range`

The SSLSessionCacheSizePerChild directive has an invalid value that is not allowed to be specified.

Error level: None

(S) Does not start the Web server.

(O) Check the value specified in the directive.

`SSLSessionCacheTimeout not set`

No value is set in the SSLSessionCacheTimeout directive.

Error level: `crit`

(S) Does not start the Web server.

(O) Set the SSLSessionCacheTimeout directive.

`The private key doesn't match the public key`

The Web server private key and Web server certificate specified in the SSLCertificateFile and SSLCertificateKeyFile directives respectively do not match correctly.

Error level: `crit`

(S) Does not start the Web server.

(O) Check to see whether the set private key and certificate form a correct pair.

`unable to set certificate`

The Web server certificate specified in the SSLCertificateFile directive cannot be set correctly.

Error level: `crit`

(S) Does not start the server.

(O) Check to see whether the Web server certificate specified in the SSLCertificateFile directive has the correct format.

`unable to set ciphers`

The cipher type specified in the SSLRequiredCiphers directive cannot be set.

Error level: `crit`

(S) Does not start the Web server.

(O) Check the value specified in the SSLRequiredCiphers directive.

`unable to set private key`

The Web server private key specified in the SSLCertificateKeyFile or SSLCertificateFile directive cannot be set correctly.

Error level: `crit`

(S) Does not start the Web server.

(O) Check to see whether the format of the private key is correct and whether the set private key and certificate form a correct pair.

`Verify depth exceeded`

Verification failed because the sent client certificate was at a lower hierarchical level than the value set in the SSLVerifyDepth directive.

Error level: error

(S) Stops the SSL request processing.

(O) Check the value set in the SSLVerifyDepth directive. If the client certificate is not accepted, no action is required.

`[client` *client-address*`] [port` *client-port-number*`] Verify depth exceeded`

Verification failed, because the sent client certificate was at a lower hierarchical level than the value set in the SSLVerifyDepth directive.

Error level: `error`

(S) Stops the SSL request processing.

(O) Check the value set in the SSLVerifyDepth directive. If the client certificate is not accepted, no action is required.

`verify error`

Verification of a client certificate failed and the issuer of the certificate cannot be obtained.

Error level: `error`

(S) Stops the SSL request processing.

(O) Check the CA certificate set in the SSLCertificateFile and SSLCertificatePath directives. If the client certificate is not accepted, no action is required.

`[client` *client-address*`] [port` *client-port number*`] verify error`

Verification of a client certificate failed and the issuer of the certificate cannot be obtained.

Error level: `error`

(S) Stops the SSL request processing.

(O) Check the CA certificate set in the SSLCertificateFile and SSLCertificatePath directives. If the client certificate is not accepted, no action is required.

`verify error:num=`*value*`:`*error-message*

Signature verification failed.

Error level: `error`

(S) Stops the SSL request processing.

(O) Read the corresponding CA certificate. If the client certificate is not accepted, no action is required.

`[client` *client-address*`] [port` *client-port-number*`] verify error:num=`*value*`:`*error-message*

Signature verification failed.

Error level: `error`

(S) Stops the SSL request processing.

(O) Read the corresponding CA certificate. If the client certificate is not accepted, no action is required.

details`: Can't open directory` *directory-name*

The directory specified in the SSLCACertificatePath directive cannot be opened.

Error level: error

(S) Does not start the Web server.

(O) Check to see whether the cause indicated in *details* exists, such as existence of the directory or sufficient permissions.

details`: Can't open key file` *file-name*

The private key file cannot be read.

Error level: `crit`

(S) Does not start the Web server.

(O) Check the value specified in the SSLCertificateKeyFile directive according to the cause indicated in *details*.

`access to` *file-name* `failed for` *host-name*`, reason: Cipher` *cipher-type* `is not on the permitted list`

The cipher type used for access is not specified in the SSLRequireCipher directive.

Error level: `error`

(S) Returns the status code "403 Forbidden" and stops processing the request.

`access to` *file-name* `failed for` *host-name*`, reason: Cipher` *cipher-type* `is forbidden`

The *cipher-type* used for access is specified in the SSLBanCipher directive.

Error level: `error`

(S) Returns the status code "403 Forbidden" and stops processing the request.

details`: Could not create a new mutex`

Initialization of lock processing failed.

Error level: `crit`

(S) Does not start the Web server.

(O) Take appropriate action according to the information in *details*.

details`: Could not open CRL directory for DER format:` *directory-name*

The directory specified in the SSLCRLDERPath directive cannot be opened.

Error level: `crit`

(S) Does not start the Web server.

(O) Take an appropriate action to resolve the cause of the issue indicated in *details*.

details`: Could not open CRL directory for PEM format:` *directory-name*

The directory specified in the SSLCRLPEMPath directive cannot be opened.

Error level: `crit`

(S) Does not start the Web server.

(O) Take an appropriate action to resolve the cause of the issue indicated in *details*.

details`: Could not open CRL file:` *file-name*

The CRL file cannot be opened.

Error level: `crit`

(S) Does not start the Web server.

(O) Take an appropriate action to resolve the cause of the issue indicated in *details*.

details`: Could not Read password file.`

The password file specified in the SSLCertificateKeyPassword directive cannot be loaded.

Error level: `crit`

(S) Does not start the Web server.

(O) Take appropriate action according to the information in *details*.

details`: couldn't change working directory:` *directory*

Setting of *directory* running a gcache server failed.

Error level: None

(S) Does not start the gcache server, but Continues processing to start the Web server.

(O) Check the setting of the SSLCacheServerRunDir directive according to *details* returned by the `chdir()` function.

details`: Error reading private key file` *file-name*`:`

The private key cannot be read.

Error level: `crit`

(S) Does not start the Web server.

(O) Check the value specified in the SSLCertificateKeyFile directive according to the cause indicated in *details*.

`access to` *file-name* `failed for` *host-name*`, reason: SSL denied`

A directory specified in the SSLDenySSL directive was accessed via SSL.

Error level: `error`

(S) Returns the status code "403 Forbidden" and stops processing the request.

`access to` *file-name* `failed for` *host-name*`, reason: SSL required`

A directory specified in the SSLRequireSSL directive was accessed without using SSL.

Error level: `error`

(S) Returns the status code "403 Forbidden" and stops processing the request.

details`: unable to exec gcache:` *file-name*

Failed to start the gcache server specified by *file-name*.

Error level: None

(S) Does not start the gcache server, but continues processing to start the Web server.

(O) Check the setting of the SSLCacheServerPath directive according to *details* returned by the `execl()` function.

details`: unable to spawn gcache process`

Faile to start the gcache server.

Error level: `crit`

(S) Does not start the gcache server, but continues processing to start the Web server.

(O) Check *details* returned by the `fork()` function.

`Could not get the certificate issuer name`

The name of the certificate's issuer could not be obtained.

Error level: `error`

(S) Fails to authenticate the client certificate.

`Could not get the issuer in CRLCheck()`

The CRL's issuer could not be obtained.

Error level: `error`

(S) Fails to authenticate the client certificate.

(O) Check to see whether the CRL was created correctly.

`Could not get the issuer name from the CRL`

The name of the CRL's issuer could not be obtained.

Error level: `error`

(S) Fails to authenticate the client certificate.

(O) Check to see whether the CRL was created correctly.

`Unable to create a time object in CRLCheck()`

A time object could not be created.

Error level: `error`

(S) Fails to authenticate the client certificate.

(O) Check system resource usage.

`Could not get the current time in CRLCheck()`

The current time could not be obtained.

Error level: `error`

(S) Fails to authenticate the client certificate.

(O) Check system resource usage.

`Unable to compare the times in CRLCheck()`

The time comparison could not be performed.

Error level: `error`

(S) Fails to authenticate the client certificate.

(O) Check system resource usage.

`Could not get the serial number from the certificate`

No serial number could be obtained from the certificate.

Error level: `error`

(S) Fails to authenticate the client certificate.

`Unable to create a certificate store object`

No certificate store object could be created.

Error level: `error`

(S) Fails to authenticate the client certificate.

(O) Check system resource usage.

`Error while trying to find certificate in CRL`

An error occurred while searching for a certificate.

Error level: `error`

(S) Fails to authenticate the client certificate.

(O) Check system resource usage.

`Unable to create a library context`

A library context could not be created.

Error level: `crit`

(S) Does not start the Web server.

(O) Check system resource usage.

`Unable to create a key context`

A key context could not be created.

Error level: `crit`

(S) Does not start the Web server.

(O) Check system resource usage.

`Unable to create a time context`

A time context could not be created.

Error level: `crit`

(S) Does not start the Web server.

(O) Check system resource usage.

`Unable to create a time object in ReadCRL()`

A time object could not be created.

Error level: `crit`

(S) Does not start the Web server.

(O) Check system resource usage.

`Unable to create a time object in GetCRL()`

A time object could not be created.

Error level: `crit`

(S) Does not start the Web server.

(O) Check system resource usage.

`Could not get the issuer in GetCRL()`

The CRL's issuer could not be obtained.

Error level: `crit`

(S) Does not start the Web server.

(O) Check to see whether the CRL was created correctly.

`Unable to compare the times in GetCRL()`

The time comparison could not be performed.

Error level: `crit`

(S) Does not start the Web server.

(O) Check system resource usage.

`Unable to create a time object in CheckCRLs()`

A time object could not be created.

Error level: `crit`

(S) Does not start the Web server.

(O) Check system resource usage.

`Could not get lastUpdate field in CheckCRLs()`

The issuance date of the CRL could not be obtained.

Error level: `crit`

(S) Does not start the Web server.

(O) Check to see whether the CRL was created correctly.

`Could not get the current time in CheckCRLs()`

The current time could not be obtained.

Error level: `crit`

(S) Does not start the Web server.

(O) Check system resource usage.

`Unable to compare the times in CheckCRLs()`

The time comparison could not be performed.

Error level: `crit`

(S) Does not start the Web server.

(O) Check system resource usage.

`Unable to create a certificate context`

No certificate context could be obtained.

Error level: `crit`

(S) Does not start the Web server.

(O) Check system resource usage.

19.9 Messages output by the Web server (Cosminexus HTTP Server) sslpasswd command

This section describes the messages output by the Cosminexus HTTP Server's `sslpasswd` command.

`Could not create the password file.`

The password file could not be created.

Error level: None

(S) Stops the processing.

(O) The read server private key file might be invalid, or the password might be too long. Check to see whether any of these errors exists.

details`: Could not open private key file.`

The server private key file could not be opened.

Error level: None

(S) Stops the processing.

(O) Take appropriate action according to the information in *details*.

`Could not read the appropriate private key file.`

The server private key file was not valid.

Error level: None

(S) Stops the processing.

(O) Specify a server private key with a password.

details`: Could not open the password file.`

The password file could not be opened.

Error level: None

(S) Stops the processing.

(O) Take appropriate action according to the information in *details*.

details`: Could not write the password file.`

The password file could not be created.

Error level: None

(S) Stops the processing.

(O) Take appropriate action according to the information in *details*.

`Unable to create a library context`

A library context could not be created.

Error level: None

(S) Stops the processing.

(O) Check system resource usage.

`Unable to create a key context`

A key context could not be created.

Error level: None

(S) Stops the processing.

(O) Check system resource usage.

Appendixes

A. Major Functional Changes in Application Server Versions

This appendix outlines the major functional changes that occurred between versions of Application Server prior to version 09-50. These changes are grouped by purpose. For details about the major functional changes made in version 09-50, see *1.4 Major functional changes in Application Server 09-50*.

This appendix contains the tables described below.

- The tables provided for each earlier version present an overview of the major functional changes in that version. For details about functionality, refer to the information shown in the columns *Reference manual* and *Relevant information*. The columns *Reference manual* and *Relevant information* indicate where to find relevant information about the functionalities in the manuals for version 09-50.
- The phrase *uCosminexus Application Server* is omitted from the manual titles listed in the column *Reference manual*.

A.1 Major functional changes in 09-00

(1) Facilitating system implementation and creation

The following table outlines the changes made to facilitate system implementation and creation.

Table A-1: Changes made to facilitate system implementation and creation

Item	Overview of changes	Reference manual	Relevant information
Canceling restrictions on the types of environments that can be created when the setup wizard is used	Restrictions on the types of environments that can be created when the setup wizard is used were removed. The setup wizard can now be used even for creation and unsetup of environments built using other methods.	*System Setup and Operation Guide*	*2.2.7*
Simplifying the procedure for deleting created environments	The deletion procedure was simplified by adding a functionality (the `mngunsetup` command) that deletes system environments created using Management Server.	*System Setup and Operation Guide*	*4.1.37*
		Command Reference Guide	*mngunsetup (deletes environments created using Management Server)*

(2) Implementing standard and existing functionalities

The following table outlines the changes made to enable implementation of standard and existing functionalities.

Table A-2: Changes made to enable implementation of standard and existing functionalities

Item	Overview of changes	Reference manual	Relevant information
Support for Servlet 3.0	Servlet 3.0 is now supported.	*Web Container Functionality Guide*	Chapter *6*
Support for EJB 3.1	EJB 3.1 is now supported.	*EJB Container Functionality Guide*	Chapter *2*
Support for JSF 2.1	JSF 2.1 is now supported.	*Web Container Functionality Guide*	Chapter *3*
Support for JSTL 1.2	JSTL 1.2 is now supported.	*Web Container Functionality Guide*	Chapter *3*
Support for CDI 1.0	CDI 1.0 is now supported.	*Common Container Functionality Guide*	Chapter *9*
Use of portable global JNDI names	Objects can now be looked up by their portable global JNDI names.	*Common Container Functionality Guide*	*2.4*
Support for 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*
Support for 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 enhancing reliability

The following table outlines the changes made to maintain and enhance reliability.

Table A-3: Changes made to maintain and enhance reliability

Item	Overview of changes	Reference manual	Relevant information
Use of TLSv1.2 for SSL/TLS communication	By using RSA BSAFE SSL-J, SSL/TLS communication can now be provided according to TLSv1.2 or any other security protocol.	This manual	*7.3*

(4) Maintaining and enhancing availability

The following table outlines the changes made to maintain and enhance availability.

Table A-4: Changes made to maintain and enhance availability

Item	Overview of changes	Reference manual	Relevant information
Monitoring the total number of pending queues for the entire Web container	The total number of pending queues for the entire Web container can now be output as operation information for monitoring.	*Operation, Monitoring, and Linkage Guide*	Chapter *3*
Output of application performance analysis traces (user-extended traces)	Performance analysis traces for analyzing the performance of user-developed applications can now be output without changing the application.	*Maintenance and Migration Guide*	Chapter *7*
Adding a restart feature for the operation management functionality	Automatic restart can now be configured for the operation management functionality (Management Server and Administration Agent). This capability allows operation to continue even if a fault occurs during operation management. In addition, the automatic-start configuration method was changed.	*Operation, Monitoring, and Linkage Guide*	*2.4.1, 2.4.2, 2.6.3, 2.6.4*
		Command Reference Guide	*mngautorun (configures and cancels the configuration of automatic start and automatic restart)*

(5) Other purposes

The following table outlines changes made for other purposes.

Table A-5: Changes made for other purposes

Item	Overview of changes	Reference manual	Relevant information
Switching between output log files	When a log is output, it is now possible to switch between destination files that have different dates.	*Maintenance and Migration Guide*	*3.2.1*
Changing the Web server name	The Web server included in the application server was renamed to *HTTP Server*.	*HTTP Server*	--
Support for direct access involving BIG-IP API (SOAP architecture)	Direct access to BIG-IP (load balancer) through API (SOAP architecture) is now supported. In addition, the method for configuring the load balancer access environment for direct access via an API was changed.	*System Setup and Operation Guide*	*4.7.3*, Appendix *K*
		This manual	*8.2*, *8.4*, *8.5*, *8.6*, *18.2*, *18.3*, *18.4*

Legend: --: Entire manual

A.2 Major functional changes in version 08-70

(1) Facilitating system implementation and creation

The table below outlines the changes made to facilitate system implementation and creation.

Table A-6: Changes made to facilitate system implementation and creation

Item	Overview of changes	Reference manual	Relevant information
Improving the management portal	The management portal window now allows you to configure the resource adapter attribute definition property (Connector attribute file settings). It also now allows you to perform connection tests and upload J2EE applications (ear and zip files) to Management Server.	*First Step Guide*	*3.5*
Adding functionality to implicitly import the import attribute of the page/tag directive	Functionality to implicitly import the import attribute of the page/tag directive is now available.	*Web Container Functionality Guide*	*2.3.7*

Item	Overview of changes	Reference manual	Relevant information
Improving the integrated user management functionality	When a database is used as the user information repository, you can now connect to the database by using a JDBC driver, which is a database product. Using Cosminexus DABroker Library's JDBC driver to connect to a database is not supported. You can now use the Easy Setup definition file and management portal window to configure the settings for the integrated user management functionality. DN for Active Directory can now include double-byte characters such as Japanese.	This manual	Chapter *5*
		This manual	*14.3*
Adding HTTP Server settings	You can now use the Easy Setup definition file and management portal window to directly configure directives (settings in `httpsd.conf`) for defining the HTTP Server operation environment.	*System Setup and Operation Guide*	*4.1.21*
		Definition Reference Guide	*4.13*

(2) Implementing standard and existing functionality

The table below outlines the changes made to enable implementation of standard and existing functionality.

Table A-7: Changes made to enable implementation of standard and existing functionality

Item	Overview of changes	Reference manual	Relevant information
Adding specification items to `ejb-jar.xml`	Class-level and method-level interceptors can now be specified in `ejb-jar.xml`.	*EJB Container Functionality Guide*	2.15
Support for parallel copy garbage collection	Parallel copy garbage collection can now be selected.	*Definition Reference Guide*	16.5

Item	Overview of changes	Reference manual	Relevant information
Support for global transactions through an inbound resource adapter that complies with Connector 1.5 specifications	Transacted delivery is now possible with resource adapters that comply with the Connector 1.5 specifications. This allows EISs that call message-driven beans to participate in global transactions.	*Common Container Functionality Guide*	3.16.3
Adapting the TP1 inbound adapter to MHP	MHP is now available as an OpenTP1 client that calls the application server by using the TP1 inbound adapter.	*Common Container Functionality Guide*	Chapter 4
Adapting the `cjrarupdate` command to the FTP inbound adapter	FTP inbound adapters are now included among the resource adapters that can be upgraded with the `cjrarupdate` command.	*Command Reference Guide*	2.2

(3) Maintaining and enhancing reliability

The table below outlines the changes made to maintain and enhance reliability.

Table A-8: Changes made to maintain and enhance reliability

Item	Overview of changes	Reference manual	Relevant information
Improving the database session failover functionality	Modes that do not obtain a lock on databases that hold global session information can now be selected for performance-focused systems. In addition, reference-only requests that do not update the database can now be defined.	*Expansion Guide*	Chapter *6*
Adding processing to be handled by the OutOfMemory handling functionality	Processing that is to be handled by the OutOfMemory handling functionality was added.	*Maintenance and Migration Guide*	*2.5.7*
		Definition Reference Guide	*16.2*
Adding memory-saving functionality for explicit heaps used in an HTTP session	Functionality to restrict the amount of memory allocated to explicit heaps in HTTP sessions was added.	*Expansion Guide*	*8.11*

(4) Other purposes

The table below outlines the changes made for other purposes.

Table A-9: Changes made for other purposes

Item	Overview of changes	Reference manual	Relevant information
Support for direct access to load balancers through an API (REST architecture)	Direct access to load balancers through an API (REST architecture) is now supported. In addition, ACOS (AX2500) is now included among the available load balancers.	*System Setup and Operation Guide*	*4.7.2, 4.7.3*
		Definition Reference Guide	*4.5*
Adding memory-saving functionality for explicit heaps used in an HTTP session	Functionality to restrict the amount of memory allocated to explicit heaps in HTTP sessions was added.	*Expansion Guide*	Appendix *A*

A.3 Major functional changes in version 08-53

(1) Implementing standard and existing functionality

The table below outlines the changes made to enable implementation of standard and existing functionality.

Table A-10: Changes made to enable implementation of standard and existing functionality

Item	Overview of changes	Reference manual	Relevant information
Calls from OpenTP1 for transaction linkage	Transaction linkage is now possible when OpenTP1 calls a message-driven bean running on the application server.	*Common Container Functionality Guide*	Chapter 4
JavaMail	The email reception functionality, which requires a JavaMail 1.3-compliant API, is now available through linkage with a POP3 email server.	*Common Container Functionality Guide*	Chapter 8

(2) Maintaining and enhancing reliability

The table below outlines the changes made to maintain and enhance reliability.

Table A-11: Changes made to maintain and enhance reliability

Item	Overview of changes	Reference manual	Relevant information
Enhancing the JavaVM troubleshooting functionality	To enhance the JavaVM troubleshooting functionality, the following functionality is now available: • Changing the operation if an `OutOfMemoryError` occurs • Setting the maximum amount of C-heap memory at JIT compilation time • Setting the maximum number of threads • Adding output items of expanded verbosegc information	*Maintenance and Migration Guide*	Chapters 4, 5, and 9

(3) Other purposes

The table below outlines the changes made for other purposes.

Table A-12: Changes made for other purposes

Item	Overview of changes	Reference manual	Relevant information
Support for Microsoft IIS 7.0 and Microsoft IIS 7.5	Microsoft IIS 7.0 and Microsoft IIS 7.5 are now supported as Web servers.	--	--
Support for HiRDB Version 9 and SQL Server 2008	The following products are now supported as databases: • HiRDB Server Version 9 • HiRDB/Developer's Kit Version 9 • HiRDB/Run Time Version 9 • SQL Server 2008 In addition, SQL Server JDBC Driver is now supported as a JDBC driver for SQL Server 2008.	*Common Container Functionality Guide*	Chapter 3

Legend:

--: Not applicable.

A.4 Major functional changes in version 08-50

(1) Facilitating system implementation and creation

The table below outlines the changes made to facilitate system implementation and creation.

Table A-13: Changes made to facilitate system implementation and creation

Item	Overview of changes	Reference manual	Relevant information
Changing the tags in `web.xml` for Web service providers	The listener, servlet, and servlet-mapping tags in `web.xml` for Web service providers were changed from mandatory to optional.	*Definition Reference Guide*	*2.4*
Using network resources on logical servers	Functionality was added to provide access from J2EE applications to network resources and network drives on other hosts.	*Operation, Monitoring, and Linkage Guide*	*1.2.3, 5.2, 5.7*
Simplifying the procedure for executing sample programs	The procedure for executing some sample programs was simplified by packaging them into EAR files.	*First Step Guide*	*3.5*
		System Setup and Operation Guide	Appendix *M*
Improving the setup wizard's completion window	The Easy Setup definition file and Connector attribute file used for setup can now be displayed in the setup wizard's completion window.	*System Setup and Operation Guide*	*2.2.6*

(2) Implementing standard and existing functionality

The table below outlines the changes made to enable implementation of standard and existing functionality.

Table A-14: Changes made to enable implementation of standard and existing functionality

Item	Overview of changes	Reference manual	Relevant information
Support for calls from OpenTP1	Message-driven beans running on Application Server can now be called from OpenTP1.	*Common Container Functionality Guide*	Chapter *4*
Support for JMS	CJMS provider functionality that complies with the JMS 1.1 specifications is now available.	*Common Container Functionality Guide*	Chapter *7*

Item	Overview of changes	Reference manual	Relevant information
Support for Java SE 6	Java SE 6 functionality is now available.	*Maintenance and Migration Guide*	*5.5, 5.8.1*
Support for generics	Generics are now available to EJB.	*EJB Container Functionality Guide*	*4.2.19*

(3) Maintaining and enhancing reliability

The table below outlines the changes made to maintain and enhance reliability.

Table A-15: Changes made to maintain and enhance reliability

Item	Overview of changes	Reference manual	Relevant information
Facilitating the use of the explicit management heap functionality	Explicit management heap functionality can now be used through the automatic arrangement configuration file.	*System Design Guide*	*7.1.1, 7.6.3, 7.10.5, 7.11.1*
		Expansion Guide	Chapter *8*
Suppressing the database session failover functionality for each URI	When the database session failover functionality is used, requests that are not to be processed by this functionality can now be specified for each URI.	*Expansion Guide*	*5.6.1*

(4) Maintaining and enhancing availability

The table below outlines the changes made to maintain and enhance availability.

Table A-16: Changes made to maintain and enhance availability

Item	Overview of changes	Reference manual	Relevant information
Omitting the management user account	The user's login ID and password can now be omitted when using the management portal, Management Server command, or Smart Composer functionality command.	*System Setup and Operation Guide*	*4.1.15*

Item	Overview of changes	Reference manual	Relevant information
		Command Reference Guide	*1.4, mngsvrctl (for Starting, Stopping, or Setting up Management Server) and mngsvrutil (Management Commands for Management Server), 8.3, cmx_admin_passwd (Configuring the Management User Account for Management Server)*

(5) Other purposes

The table below outlines the changes made for other purposes.

Table A-17: Changes made for other purposes

Item	Overview of changes	Reference manual	Relevant information
Statistical functionality to identify unnecessary objects in the tenured area	It is now possible to identify only those objects in the tenured area that are unnecessary.	*Maintenance and Migration Guide*	*9.8*
Functionality to output a list of reference objects to enable identification of unnecessary objects in the tenured area	It is now possible to output a list of objects that can be used as reference objects to identify unnecessary objects in the tenured area using the abovementioned statistical functionality.		*9.9*
Per-class statistics analysis functionality	Per-class statistics can now be output in CSV format.		*9.10*

Item	Overview of changes	Reference manual	Relevant information
Cluster node switching due to detecting that a logical server has automatically restarted too many times	In a cluster configuration where Management Server is monitored for node switching, you can now set up node switching to take place when a logical server is abnormally inactive (that is, when it has automatically restarted too many times or if a fault is detected when the automatic restart count is 0).	*Operation, Monitoring, and Linkage Guide*	*18.4.3*, *18.5.3*, *20.2.2*, *20.3.3*, *20.3.4*
Node switching in per-host management models	Node switching in per-host management models is now possible during the operation of a system linked with cluster software.		Chapter *20*
Support for ACOS (AX2000 or BS320)	ACOS (AX2000 or BS320) is now available as a load balancer.	*System Setup and Operation Guide*	*4.7*, *4.7.3*, *4.7.5*, *4.7.6*, Appendix *K*, *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 for the stateful session bean (with session synchronization interface) for CMT transaction management	The transaction attributes `Supports`, `NotSupported`, and `Never` can now be specified for stateful session beans (with session synchronization interface) for CMT transaction management.	*EJB Container Functionality Guide*	*2.7.3*
Forcibly terminating Administration Agent on `OutOfMemoryError`	If an `OutOfMemoryError` occurs in JavaVM, Administration Agent is now forced to terminate.	*Maintenance and Migration Guide*	*2.5.8*
Asynchronous parallel processing of threads	Asynchronous timer processing and asynchronous thread processing are now possible with TimerManager and WorkManager.	*Expansion Guide*	Chapter *10*

A.5 Major functional changes in version 08-00

(1) Improving development productivity

The table below outlines the changes made to improve development productivity.

Table A-18: Changes made to improve development productivity

Item	Overview of changes	Reference manual	Relevant information
Facilitating migration from other application server products	To facilitate migration from other application server products, the following functionality was made available: • Determining the upper limit for HTTP sessions, based on exceptions • Preventing translation errors when there are duplicate JavaBeans IDs or when custom tag attribute names and TLD definitions are case-insensitive	*Web Container Functionality Guide*	*2.3*, *2.7.5*
Offering `cosminexus.xml`	After a J2EE application is imported into the J2EE server, that application can now be started by entering the Cosminexus application server's unique attributes into `cosminexus.xml`, without configuring the properties.	*Common Container Functionality Guide*	*11.3*

(2) Implementing standard functionality

The table below outlines the changes made to enable implementation of standard functionality.

Table A-19: Changes made to enable implementation of standard functionality

Item	Overview of changes	Reference manual	Relevant information
Support for Servlet 2.5	Servlet 2.5 is now supported.	*Web Container Functionality Guide*	*2.2*, *2.5.4*, *2.6*, Chapter *6*
Support for JSP 2.1	JSP 2.1 is now supported.	*Web Container Functionality Guide*	*2.3.1*, *2.3.3*, *2.5*, *2.6*, Chapter *6*
JSP debugging	JSP debugging is now possible in development environments that use MyEclipse.#	*Web Container Functionality Guide*	*2.4*

Item	Overview of changes	Reference manual	Relevant information
Storing the tag library into a library JAR and providing TLD mappings	When the tag library is inside a library JAR, you can now use a Web container at Web application startup to search the library JAR for a TLD file and provide TLD mappings automatically.	*Web Container Functionality Guide*	*2.3.4*
Omitting `application.xml`	The file `application.xml` can now be omitted when using J2EE applications.	*Common Container Functionality Guide*	*11.4*
Using both annotations and DD	Both annotations and DD can now be used. This allows the information specified in the annotation to be updated with DD.	*Common Container Functionality Guide*	*12.5*
Annotations complying with the Java EE 5 standard (default interceptor)	The default interceptor can now be stored into a library JAR. In addition, DI processing from the default interceptor is now possible.	*Common Container Functionality Guide*	*11.4*
Resolving a reference with @Resource	Resource reference can now be resolved with @Resource.	*Common Container Functionality Guide*	*12.4*
Support for JPA	The JPA specifications are now supported.	*Common Container Functionality Guide*	Chapters *5* and *6*

#: The JSP debug functionality of version 09-00 or later is available in development environment which uses WTP.

(3) Maintaining and enhancing reliability

The table below outlines the changes made to maintain and enhance reliability.

Table A-20: Changes made to maintain and enhance reliability

Item	Overview of changes	Reference manual	Relevant information
Inheriting session information	HTTP session information is now stored in a database so the information can be inherited.	*Expansion Guide*	Chapters *5* and *6*

Item	Overview of changes	Reference manual	Relevant information
Suppressing full garbage collection	The object that causes full garbage collection is now stored in memory outside the Java heap. This means you can suppress full garbage collection.	*Expansion Guide*	Chapter *8*
Monitoring client performance	The time taken for client processing can now be checked and analyzed.	--	--

Legend:

--: Functionality eliminated from version 09-00

(4) Other purposes

The table below outlines the changes made for other purposes.

Table A-21: Changes made for other purposes

Item	Overview of changes	Reference manual	Relevant information
Deleting invalid HTTP cookies	Any invalid HTTP cookies can now be deleted.	*Web Container Functionality Guide*	*2.7.4*
Detecting Naming Service errors	If a Naming Service error occurs, the EJB client can now detect it earlier than before.	*Common Container Functionality Guide*	*2.9*
Connection error detection timeout	The period for connection error detection timeouts can now be specified.	*Common Container Functionality Guide*	*3.15.1*
Support for Oracle 11g	Oracle 11g can now be used as a database.	*Common Container Functionality Guide*	Chapter *3*
Scheduling batch processing	Execution of batch applications can now be scheduled using CTM.	*Expansion Guide*	Chapter *4*
Batch processing log	The size and area count for the log file for batch processing commands can now be specified. In addition, the retry count and retry interval for accessing this file when it is exclusive-locked can now be specified.	*Definition Reference Guide*	*3.6*
Snapshot log	The contents of the snapshot log were changed.	*Maintenance and Migration Guide*	*A.1* and *A.2*

Item	Overview of changes	Reference manual	Relevant information
Disclosing the protected areas for method cancel	A list of protected areas not subject to method cancel was disclosed.	*Operation, Monitoring, and Linkage Guide*	Appendix *C*
Functionality to choose whether to perform garbage collection before statistics output	You can now choose whether to perform garbage collection before output of statistics per class.	*Maintenance and Migration Guide*	*9.7*
Functionality to output age distribution information for the survivor area	Age distribution information about the Java object in the survivor area can now be output to the JavaVM log file.	*Maintenance and Migration Guide*	*9.11*
Functionality to eliminate accumulated finalization processes	Accumulated JavaVM finalization processes can now be monitored and eliminated.	--	--
Changing the maximum heap size for server management commands	The maximum size of the heap available for server management commands was changed.	*Definition Reference Guide*	*5.2*, *5.3*
Action taken when a non-recommended display name is specified	A message is now output when a non-recommended display name is specified for a J2EE application.	*Messages*	KDJE42374-W

Legend:

--: Functionality that was dropped in 09-00.

B. Registration of Exception Lists (Windows)

If you enable Windows Firewall, you must register the component software programs to the firewall's exception list. Which component software programs you register to the exception list depends on which component software programs are installed. This applies to the following OSs:

- Windows Server 2012
- Windows Server 2008
- Windows 8
- Windows 7
- Windows Vista
- Windows XP

If you enable the firewall, execute the appropriate command at the command prompt to register to the exception list the component software programs listed in the table below that have been installed. The table below shows the exception list registration command to execute for each component software program. Programs created by using the application server and BPM/ESB infrastructure products must also be added to the exception list. You can use the exception list registration command to register these programs to the exception list as well.

Table B-1: Exception list registration command to be executed for component software

Installed component software	Requirement for registration to the exception list	Exception list registration command to be executed
Component Container	Required	`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CC\server\bin\cjstartsv.exe" name="Cosminexus Component Container" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CC\web\bin\cjstartweb.exe" name="Cosminexus Component Container" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CC\client\bin\cjclstartap.exe" name="Cosminexus Component Container" mode=ENABLE`

Installed component software	Requirement for registration to the exception list	Exception list registration command to be executed
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\manager\bin\adminagent.exe" name="Cosminexus Component Container" mode=ENABLE`
	Required if server management commands are executed	`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\bin\vbj.exe" name="Cosminexus Component Container" mode=ENABLE`#1
	Required if the scheduling functionality is used by batch commands	`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CC\batch\bin\cjexecjob.exe" name="Cosminexus Component Container" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CC\batch\bin\cjkilljob.exe" name="Cosminexus Component Container" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CC\batch\bin\cjlistjob.exe" name="Cosminexus Component Container" mode=ENABLE`
	Required if server communication agents of virtual servers are used (32-bit version of Windows)	`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\sinagent\bin\sinaviagent.exe" name="uCosminexus SI Navigation System Agent" mode=ENABLE`
Component Transaction Monitor	Required	`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CTM\bin\ctmchpara.exe" name="Cosminexus Component Transaction Monitor" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CTM\bin\ctmd.exe" name="Cosminexus Component Transaction Monitor" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CTM\bin\ctmdmd.exe" name="Cosminexus Component Transaction Monitor" mode=ENABLE`

Installed component software	Requirement for registration to the exception list	Exception list registration command to be executed
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CTM\bin\ctmdmstart.exe" name="Cosminexus Component Transaction Monitor" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CTM\bin\ctmdmstop.exe" name="Cosminexus Component Transaction Monitor" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CTM\bin\ctmgetior.exe" name="Cosminexus Component Transaction Monitor" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CTM\bin\ctmholdque.exe" name="Cosminexus Component Transaction Monitor" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CTM\bin\ctmidl2cpp.exe" name="Cosminexus Component Transaction Monitor" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CTM\bin\ctmidl2j.exe" name="Cosminexus Component Transaction Monitor" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CTM\bin\ctmlsque.exe" name="Cosminexus Component Transaction Monitor" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CTM\bin\ctmnaminfo.exe" name="Cosminexus Component Transaction Monitor" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CTM\bin\ctmregltd.exe" name="Cosminexus Component Transaction Monitor" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CTM\bin\ctmridinfo.exe" name="Cosminexus Component Transaction Monitor" mode=ENABLE`

Installed component software	Requirement for registration to the exception list	Exception list registration command to be executed
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CTM\bin\ctmrlesque.exe" name="Cosminexus Component Transaction Monitor" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CTM\bin\ctmstart.exe" name="Cosminexus Component Transaction Monitor" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CTM\bin\ctmstop.exe" name="Cosminexus Component Transaction Monitor" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CTM\bin\ctmstartgw.exe" name="Cosminexus Component Transaction Monitor" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CTM\bin\ctmstopgw.exe" name="Cosminexus Component Transaction Monitor" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\CTM\bin\ctmtscgwd.exe" name="Cosminexus Component Transaction Monitor" mode=ENABLE`
HTTP Server	Required	`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\httpsd\httpsd.exe" name="Cosminexus HTTP Server"mode=ENABLE`
TPBroker#2	Required	`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\bin\events.exe" name="Cosminexus TPBroker" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\bin\gatekeeper.exe" name="Cosminexus TPBroker" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\bin\irep.exe" name="Cosminexus TPBroker" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\bin\nameserv.exe" name="Cosminexus TPBroker" mode=ENABLE`

Installed component software	Requirement for registration to the exception list	Exception list registration command to be executed
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\bin\oad.exe" name="Cosminexus TPBroker" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\bin\osagent.exe" name="Cosminexus TPBroker" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\bin\osfind.exe" name="Cosminexus TPBroker" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\bin\admd.exe" name="Cosminexus TPBroker" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\bin\otsd.exe" name="Cosminexus TPBroker" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\bin\trnctxsv.exe" name="Cosminexus TPBroker" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\bin\tsstoptrnctxsv.exe" name="Cosminexus TPBroker" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\otsspool\bin\complete.exe" name="Cosminexus TPBroker" mode=ENABLE`#3
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\otsspool\bin\rcvd.exe" name="Cosminexus TPBroker" mode=ENABLE`#3
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\bin\tscommit.exe" name="Cosminexus TPBroker" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\bin\tslstrn.exe" name="Cosminexus TPBroker" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\bin\tsrollback.exe" name="Cosminexus TPBroker" mode=ENABLE`

Installed component software	Requirement for registration to the exception list	Exception list registration command to be executed
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\bin\tsstat.exe" name="Cosminexus TPBroker" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\bin\tsstop.exe" name="Cosminexus TPBroker" mode=ENABLE`
		`netsh firewall add allowedprogram program="`*Application-Server-installation-directory*`\TPB\bin\tstrnsts.exe" name="Cosminexus TPBroker" mode=ENABLE`
HiRDB Embedded Server Version 8	Required if embedded databases are used in Developer or Service Architect	`for %%p in (`*Developer-or-Service-Architect-installation-directory*`\DB\bin\*.exe) do netsh firewall set allowedprogram %%p "Cosminexus Developer(DB)"`
		`for %%p in (`*Developer-or-Service-Architect-installation-directory*`\DB\lib\servers\*.exe) do netsh firewall set allowedprogram %%p "Cosminexus Developer(DB)"`
		`for %%p in (`*Developer-or-Service-Architect-installation-directory*`\DB\SAMPLE\sampleconf\*.exe) do netsh firewall set allowedprogram %%p "Cosminexus Developer(DB)"`
		`for %%p in (`*Developer-or-Service-Architect-installation-directory*`\DB\SAMPLE\tools\*.exe) do netsh firewall set allowedprogram %%p "Cosminexus Developer(DB)"`
		`for %%p in (`*Developer-or-Service-Architect-installation-directory*`\DB\HiRDEF\*.exe)do netsh firewall set allowedprogram %%p "Cosminexus Developer(DB)"`
		`for %%p in (`*Developer-or-Service-Architect-installation-directory*`\DB\PDISTUP\bin\*.exe) do netsh firewall set allowedprogram %%p "Cosminexus Developer(DB)"`

#1: EJB clients that use the `vbj` command are also excluded from filtering for the firewall.

#2: You can also use the `tssetfw` command to register exception lists. For details about the `tssetfw` command, see the *TPBroker Additional Features* manual.

#3: If the OTS functionality is used, the task depends on whether the `TPSPOOL` environment variable has been set:

- When the `TPSPOOL` environment variable has been set

  Use the `tssetup` command to initialize the TPBroker environment, and then register the following exception lists manually:

  ```
  %TPSPOOL%\bin\rcvd.exe
  ```

  ```
  %TPSPOOL%\bin\complete.exe
  ```

- When thee `TPSPOOL` environment variable has not been set

  Use the `tssetup` command to initialize the TPBroker environment, and then use the commands to register the exception lists.

C. Terminology used in this manual

Terminology used in this manual

For the terms used in the manual, see the *uCosminexus Application Server and BPM/ ESB Platform Terminology Guide*.

Index