HITACHI
Inspire the Next

Hitachi Navigation Platform Development Guide

3021-3-025-30(E)

Notices

■ Relevant program products

P-2943-4PA4 Hitachi Navigation Platform 10-50 (For Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2, Windows Server 2016)

P-2943-4VA4 Hitachi Navigation Platform for Developers 10-50 (For Windows 7 x64, Windows 8 x64, Windows 8.1 x64, Windows 10 x64)

P-292C-4PBL JP1/Navigation Platform 11-50 (For Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2, Windows Server 2016)

P-292C-4VBL JP1/Navigation Platform for Developers 11-50 (For Windows 7 x64, Windows 8 x64, Windows 8.1 x64, Windows 10 x64)

■ Trademarks

This product includes software developed by the University of California, Berkeley and its contributors.

This software contains code derived from the RSA Data Security Inc. MD5 Message-Digest Algorithm, including various modifications by Spyglass Inc., Carnegie Mellon University, and Bell Communications Research, Inc (Bellcore).

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

This product includes RSA BSAFE(R) Cryptographic software of EMC Corporation.

■ Microsoft product name abbreviations

This manual uses the following abbreviations for Microsoft product names.

Abbreviation			Full name or meaning
Excel			Microsoft(R) Office Excel
Internet Explorer	Internet Explorer 8		Windows(R) Internet Explorer(R) 8
	Internet Explorer 9		Windows(R) Internet Explorer(R) 9
	Internet Explorer 10		Windows(R) Internet Explorer(R) 10
	Internet Explorer 11		Windows(R) Internet Explorer(R) 11
Windows	Windows 7	Windows 7 x64	Microsoft(R) Windows(R) 7 Enterprise (64-bit Edition)
			Microsoft(R) Windows(R) 7 Professional (64-bit Edition)
			Microsoft(R) Windows(R) 7 Ultimate (64-bit Edition)
	Windows 8	Windows 8 x64	Windows(R) 8 Enterprise (64-bit Edition)
			Windows(R) 8 Pro (64-bit Edition)
	Windows 8.1	Windows 8.1 x64	Windows(R) 8.1 Enterprise (64-bit Edition)
			Windows(R) 8.1 Pro (64-bit Edition)
	Windows 10	Windows 10 x64	Windows(R) 10 Enterprise (64-bit Edition)

Abbreviation			Full name or meaning
Windows	Windows 10	Windows 10 x64	Windows(R) 10 Pro (64-bit Edition)
	Windows Server 2008 R2		Microsoft(R) Windows Server(R) 2008 R2 Datacenter
			Microsoft(R) Windows Server(R) 2008 R2 Enterprise
			Microsoft(R) Windows Server(R) 2008 R2 Standard
	Windows Server 2012		Microsoft(R) Windows Server(R) 2012 Datacenter
			Microsoft(R) Windows Server(R) 2012 Standard
	Windows Server 2012 R2		Microsoft(R) Windows Server(R) 2012 R2 Datacenter
			Microsoft(R) Windows Server(R) 2012 R2 Standard
	Windows Server 2016		Microsoft(R) Windows Server(R) 2016 Datacenter
			Microsoft(R) Windows Server(R) 2016 Standard

■ Restrictions

■ Issued

Nov. 2017: 3021-3-025-30(E)

■ Copyright

Summary of amendments

The following table lists changes in this manual (3021-3-025-30(E)) and product changes related to this manual.

Changes	Location
The version of JDK, which is a prerequisite for setting up Eclipse, was changed to 8.	*3.1*
The value to be entered in the **VM arguments** field by clicking the **JRE** tab when an attempt to build plugins has failed was changed to `-XX:MaxMetaspaceSize=256m`.	*4.6.2*

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

Preface

This manual describes how to develop plugins and custom windows that are used in the following products, and describes about API references:

- Hitachi Navigation Platform
- Hitachi Navigation Platform for Developers
- JP1/Navigation Platform
- JP1/Navigation Platform for Developers

■ Abbreviations for product names

This manual uses the following abbreviations for the above product names:

Abbreviation		Full name
Navigation Platform	Navigation Platform	Hitachi Navigation Platform
		JP1/Navigation Platform
	Navigation Platform for Developers	Hitachi Navigation Platform for Developers
		JP1/Navigation Platform for Developers

■ Intended readers

This manual is intended for users who develop plugins and custom windows by using APIs provided by Navigation Platform

Note that readers of this manual must have:

- Basic knowledge of Windows operations
- Basic knowledge of Java program development
- Basic knowledge of Eclipse
- Basic knowledge of XML

■ Conventions: Fonts and symbols

The following table explains the text formatting conventions used in this manual:

Text formatting	Convention
Bold	Bold characters indicate text in a window, other than the window title. Such text includes menus, menu options, buttons, radio box options, or explanatory labels. For example: • From the **File** menu, choose **Open**. • Click the **Cancel** button. • In the **Enter name** entry box, type your name.

Text formatting	Convention
Italic	Italic characters indicate a placeholder for some actual text to be provided by the user or system. For example: • Write the command as follows: `copy` *source-file target-file* • The following message appears: `A file was not found. (file =` *file-name*`)` Italic characters are also used for emphasis. For example: • Do *not* delete the configuration file.
`Monospace`	Monospace characters indicate text that the user enters without change, or text (such as messages) output by the system. For example: • At the prompt, enter `dir`. • Use the `send` command to send mail. • The following message is displayed: `The password is incorrect.`

The following table explains the symbols used in this manual:

Symbol	Convention
`|`	In syntax explanations, a vertical bar separates multiple items, and has the meaning of OR. For example: `A|B|C` means `A`, or `B`, or `C`.
`{ }`	In syntax explanations, curly brackets indicate that only one of the enclosed items is to be selected. For example: `{A|B|C}` means only one of `A`, or `B`, or `C`.
`[ ]`	In syntax explanations, square brackets indicate that the enclosed item or items are optional. For example: `[A]` means that you can specify `A` or nothing. `[B|C]` means that you can specify `B`, or `C`, or nothing.
`...`	In coding, an ellipsis (...) indicates that one or more lines of coding have been omitted. In syntax explanations, an ellipsis indicates that the immediately preceding item can be repeated as many times as necessary. For example: `A, B, B, ...` means that, after you specify `A, B,` you can specify `B` as many times as necessary.

Contents

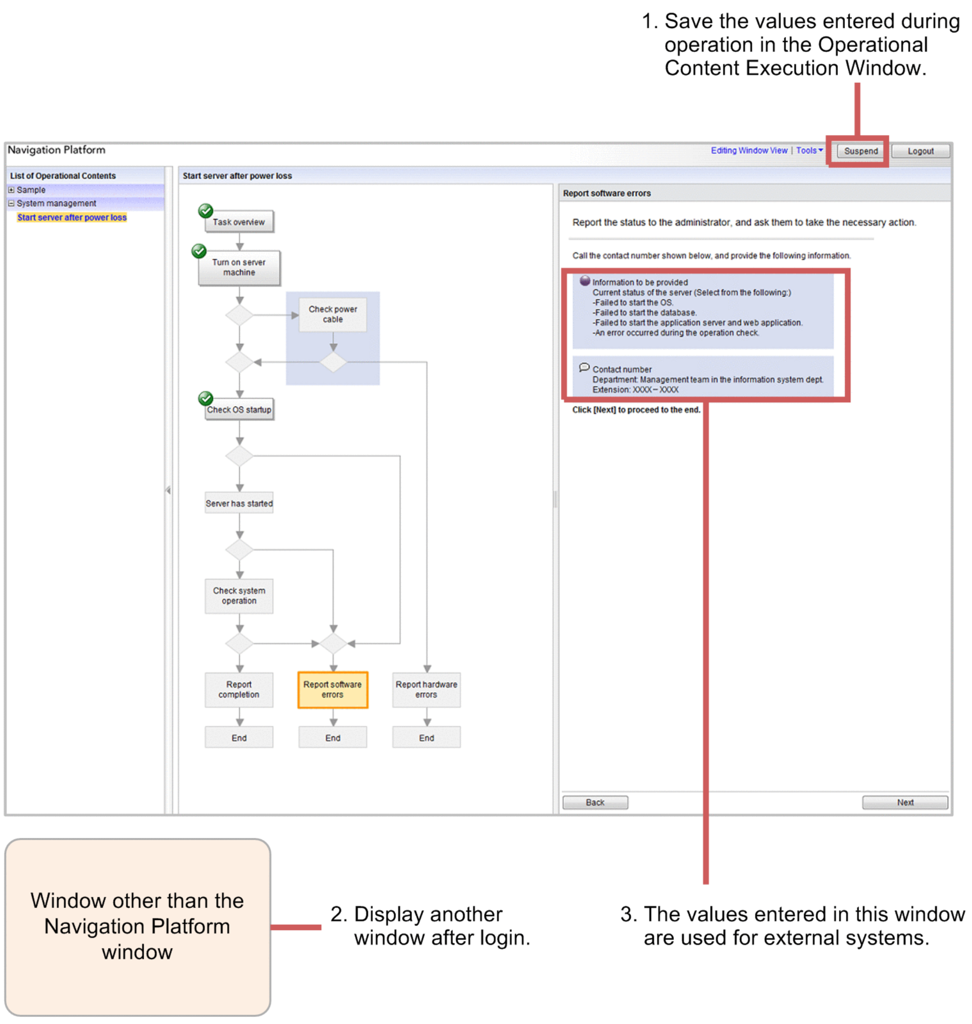
1 Overview of Development

This chapter describes the range of plugins that can be used in Navigation Platform and the overview of development operation.

1.1 Range of development

By developing plugins in accordance with the contents of operation, use of Navigation Platform becomes more convenient. The following shows an example of customizing a Navigation Platform window by using developed plugins

Figure 1–1: Example of customizing a Navigation Platform window


1. Save the values entered during operation in the Operational Content Execution Window.
Navigation Platform
Editing Window View | Tools
Suspend
Logout
List of Operational Contents
Sample
System management
Start server after power loss
Start server after power loss
Task overview
Turn on server machine
Check power cable
Check OS startup
Server has started
Check system operation
Report completion
Report software errors
Report hardware errors
End
End
End
Report software errors
Back
Next
Window other than the Navigation Platform window
2. Display another window after login.
3. The values entered in this window are used for external systems.

Report the status to the administrator, and ask them to take the necessary action.

Call the contact number shown below, and provide the following information.

Information to be provided
Current status of the server (Select from the following:)
-Failed to start the OS.
-Failed to start the database.
-Failed to start the application server and web application.
-An error occurred during the operation check.

Contact number
Department: Management team in the information system dept.
Extension: XXXX−XXXX

Click [Next] to proceed to the end.

To customize the window shown in this figure, you must first develop the following plugins:

1. Suspend/Resume Plugin
2. Custom Window Plugin
3. I/O Plugin

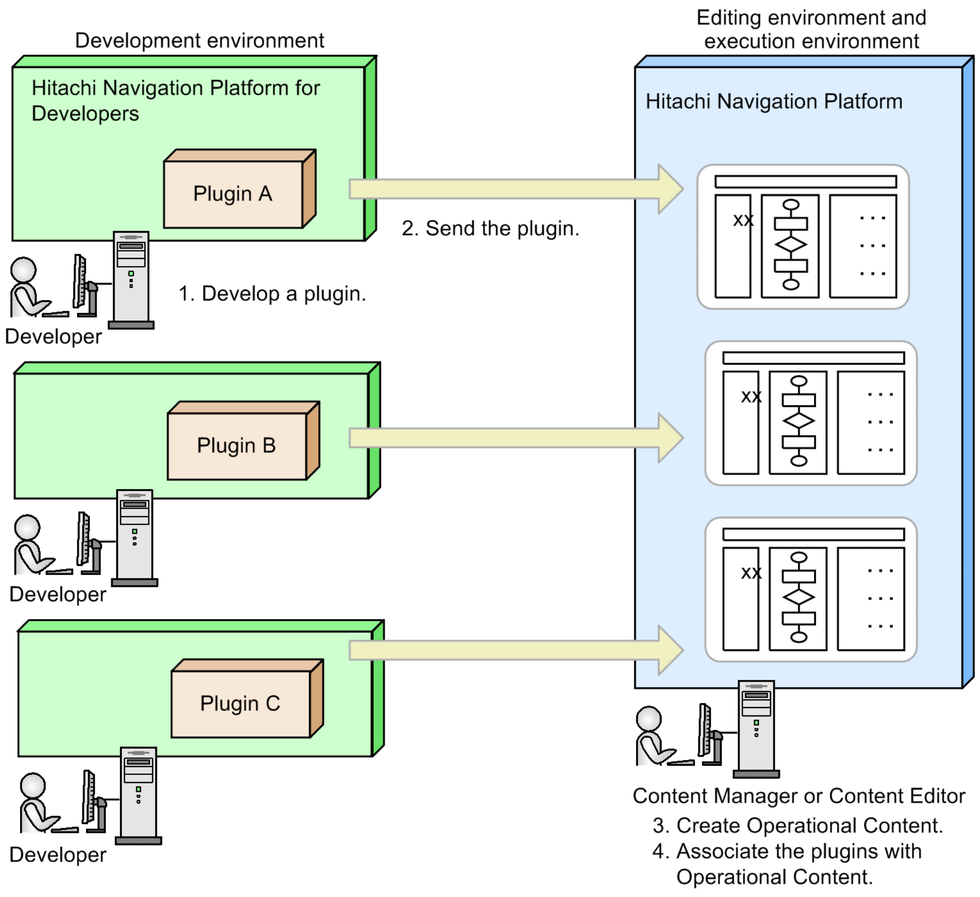
This manual describes how to customize Navigation Platform by using these plugins.

For details about how to customize Navigation Platform without developing plugins, see the description of customization and user property file (`ucnp_user.properties`) in the *Hitachi Navigation Platform Setup and Operations Guide*.

1.2 Flow of plugin development

The following figure shows an overview and flow of plugin development.

Figure 1–2: Overview of plugin development


Development environment
Hitachi Navigation Platform for Developers
Plugin A
Plugin B
Plugin C
Developer
Developer
Developer
1. Develop a plugin.
2. Send the plugin.
Editing environment and execution environment
Hitachi Navigation Platform
xx
Content Manager or Content Editor
3. Create Operational Content.
4. Associate the plugins with Operational Content.

The numbers in the figure correspond to the following numbers:

1. A developer develops a plugin in a development environment.
   Eclipse, which is provided by Navigation Platform, is used for plugin development.
   To use Eclipse, setup is required in advance.
2. The developer sends the developed plugin to the editing environment and execution environment.
   The system administrator applies the plugin to the editing environment and execution environment.
3. After plugins are applied, the Content Manager or Content Editor creates Operational Content in the editing environment.
4. For I/O Plugins, the Content Manager or Content Editor associates the plugins with Operational Content by drawing mapping lines between Plugin Parts and Guide Parts.
   For Suspend/Resume Plugins and Custom Window Plugins, association is not necessary.

Reference note

If the system configuration does not use an editing environment, a developer might also perform the tasks in 3 and 4 in a development environment. For details about system configurations, see the *Hitachi Navigation Platform Setup and Operations Guide*. For details about how to create Operational Content, see the *Hitachi Navigation Platform Content Editing Guide*.

The following provides details about plugin development tasks and includes references.

Before starting development

You must construct a development environment before starting the tasks described here. Note that you must have Windows administrator roles to perform any task described here.

Upon completion of the development

After you have completed development of plugins, send J2EE applications (*plugin-name*`.ear`) to an editing environment and execution environment. In the case of developed I/O Plugins, export Operational Content associated with the plugins, and then import them to the editing environment and execution environment.

For details about each task, see the *Hitachi Navigation Platform Setup and Operations Guide*.

Table 1–1: Plugin development tasks

Order	Task	I/O Plugin	Suspend/Resume Plugin	Custom Window Plugin	See:
1	Preparing a development environment (by setting up Eclipse and adding libraries)	R	R	R	*Chapter 3*
2	Creating template plugins	R	R	R	*4.1*
3	Importing a template plugin project to Eclipse	R	R	R	*4.2*
4	Customizing (editing) the template plugin by using Eclipse	R	R	R	*4.3.1*, *4.4*
5	Adding (and implementing) necessary processing to the plugin	S	S	S	*4.4*, *4.5*
6	Building a customized template plugin project by using Ant	R	R	R	*4.6*
7	Deploying the plugin J2EE application (*plugin-name*`.ear`) by using Ant	R	R	R	*4.7*
8	Placing Plugin Parts in the Guide area and then associating plugins with Operational Content by drawing mapping lines to connect Plugin Parts and Guide Parts	R	N	N	*4.8*
9	Checking the user property file	R	R	S	*4.9*
10	Debugging plugins by using the Eclipse debugger function	R	R	R	*4.10*

Legend:

R: The task is required.

N: The task is not required

S: Perform the task if necessary.

Note:

You can change J2EE server setting items if necessary. For details, see *4.13 Changing J2EE server settings*.

2 Plugin Overview

This chapter describes the types of plugins that can be developed in Navigation Platform, and provides an overview of each type.

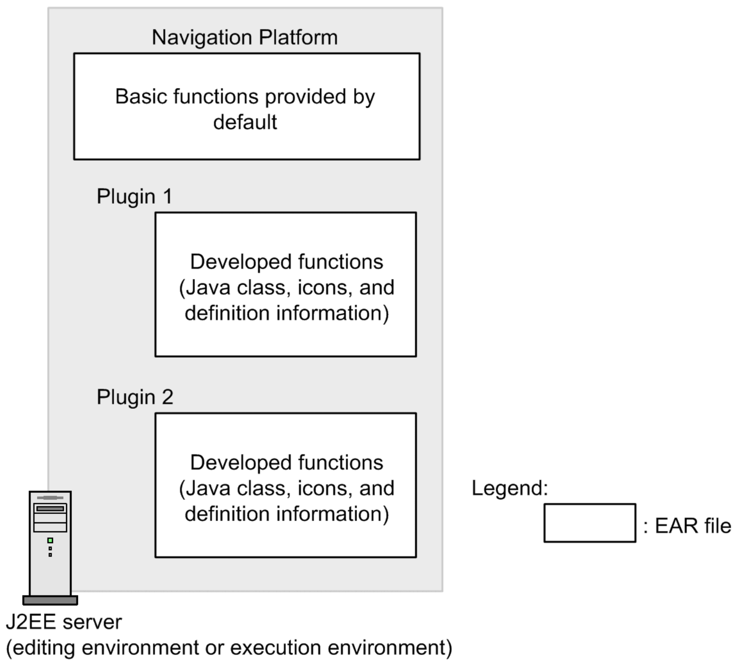
2.1 Overview of Navigation Platform plugins

Navigation Platform provides two types of plugins: System Plugins that do not need to be developed and User Plugins that need to be developed. This manual describes User Plugins (referred to as *plugins* hereafter).

The entity of a plugin is an EAR file. Developers use Eclipse in a development environment to create plugin EAR files of Navigation Platform. By deploying EAR files created in the development environment into Navigation Platform (J2EE server) in an editing environment or execution environment, Content Editors and Content users are able to use plugins.

Plugins deployed in Navigation Platform (J2EE server) might be called *J2EE applications*. The following indicates the relationship between plugin EAR files and the Navigation Platform EAR file.

Figure 2–1: Image of plugins incorporated in Navigation Platform


Navigation Platform
Basic functions provided by default
Plugin 1
Developed functions (Java class, icons, and definition information)
Plugin 2
Developed functions (Java class, icons, and definition information)
Legend:
: EAR file
J2EE server
(editing environment or execution environment)

Plugins developed by using uCosminexus Navigation Developer 09-00 or earlier are incorporated in Navigation Platform in a different way, and are therefore not available. Plugins developed by using uCosminexus Navigation Developer 09-50 or later are available, although tasks such as copying definition information to a new format template plugin must be performed.

2.1.1 About initialization and termination processing of plugins

You must use the `IPluginInitializer` interface to implement initialization and termination processing in plugins in the following cases:

- A connection is established from the plugin to a database.
- Preprocessing is required for executing a plugin.
- Postprocessing is required when a plugin stops.

When Navigation Platform starts, plugins also start

(1) Creating instances of initialization and termination processing classes

Instances are created when a plugin starts.

(2) Discarding instances of initialization and termination processing classes

Instances are discarded when a servlet stops.

2.1.2 About plugin sessions

When you execute the `setAttribute` method for the `HttpSession` object acquired in a plugin, do not specify a name beginning with any of the character strings listed below for the `name` parameter of the `setAttribute` method.

Character strings prohibited at the beginning of the `name` parameter:

```
ucnp
java.
javax.
javax.portlet.
hptl
com.cosminexus
jp.co.hitachi.soft.portal
```

2.2 Types of plugins

Plugins that can be developed in Navigation Platform are classified by function as follows:

- I/O Plugin
- Suspend/Resume Plugin
- Custom Window Plugin

Operation of I/O Plugins and Suspend/Resume Plugins uses the values (cache) entered or selected in windows by users. Custom Window Plugins are used to display separate windows independent of Navigation Platform.

2.3 Overview of I/O Plugins

I/O Plugins are executed during node transition in Operational Content. To use, in the next node, the values input by users in Operational Content windows or to pass such values to external programs, you need to develop I/O Plugins.

2.3.1 I/O Plugin execution timing

The following table describes the order in which I/O Plugins are executed during node transition in Operational Content.

Table 2–1: I/O Plugins and methods executed during node transition

Execution order	I/O Plugin	I/O Plugin method
1	I/O Plugin associated with the transfer source node	`inputFromNode` method (`IIoPluginController` (server processing implementation interface))
2	I/O Plugin associated with the transfer destination node#	`outputToNode` method (`IIoPluginController` (server processing implementation interface))

#
Including an I/O Plugin associated with the start process node that is not connected to the Terminal Node (start)

Note that I/O Plugins are also executed when node transition occurs in the Operational Content Execution Window (preview). If you do not want to execute I/O Plugins in a preview window, you need to implement accordingly. For details about the implementation, see *5.2.3 Plugin processing during preview*.

The following operations are possible by specifying error information for return values of the methods of I/O Plugins shown in this table:

- Display an alert when a user executes the I/O Plugin
- Highlight the item for which an invalid value is entered, and then suppress node transition

For details about the information you can specify for return values of methods, see the return values of *inputFromNode method* or *outputToNode method* in *5.2 IIoPluginController (server processing implementation interface)*.

2.3.2 Execution order of I/O Plugins

When you develop multiple I/O Plugins, we recommend you make sure that they are independent of any other plugin processing.

However, in case multiple I/O Plugins are placed in the same node, you can specify the execution order in an I/O Plugin XML file so that they are executed in the specified ascending order. You can specify the execution order for each plugin, but cannot specify the execution order of methods within a plugin.

2.3.3 Data that can be passed by I/O Plugins

Data that can be received by plugins

When executing an I/O Plugin, you can receive data such as information about the transfer source or transfer destination node and the cache of Guide Parts mapped in the Plugin Part parameters. The data is passed as a parameter of the `inputFromNode` method or `outputToNode` method of `IIoPluginController` (server processing implementation interface).

Data returned by plugins

The following operations are possible by specifying the updated cache of Guide Parts mapped in the Plugin Part parameters and error information for the data to be returned by plugins:

- Display alerts for users
- Highlight the item for which an invalid value is entered, and then suppress node transition

Make sure that the data is returned as a return value of the `inputFromNode` method or `outputToNode` method of `IIoPluginController` (server processing implementation interface).

For details about data that can be passed with I/O Plugins and, see the following in *5.2 IIoPluginController (server processing implementation interface)*:

- Descriptions of `param` and return values in *inputFromNode method*
- Descriptions of `param` and return values in *outputToNode method*

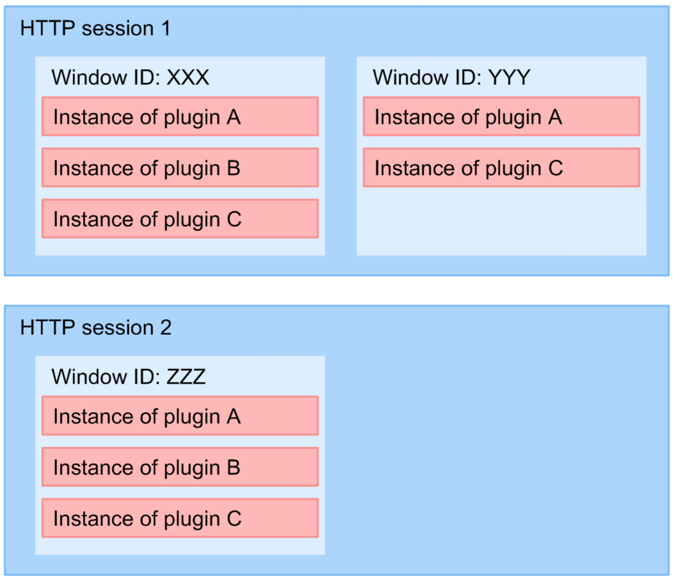
2.3.4 Lifecycle of I/O Plugin instances

The following describes the lifecycle of plugin instances.

(1) Creating and retaining I/O Plugin instances

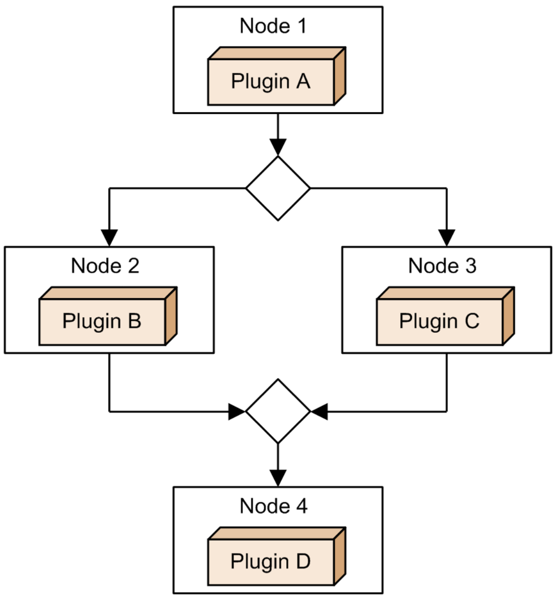
For I/O Plugins having the same plugin ID, one instance is retained in one window. The instances are retained in HTTP sessions based on window IDs as keys. The following figure shows instances of I/O Plugins.

Figure 2–2: I/O Plugin instances



When you execute I/O Plugins, instances of the I/O Plugins to be executed are acquired from HTTP sessions by using window IDs and plugin IDs as keys. If an instance cannot be acquired from HTTP sessions, new instances are created and then added to the HTTP sessions. At this time, only the instances to be executed are created. The figure below shows an example of Operational Content containing branch nodes. If transition occurs from node 1 to node 2, and to node 4 during execution of this Operational Content, plugins A, B, and D are executed but plugin C is not. Therefore, if operation of Operational Content containing this transition is executed, an instance of plugin C is not created.

Figure 2–3: Operational Content containing branches



(2) Discarding I/O Plugin instances

The table below indicates when I/O Plugin instances are discarded. This applies regardless of whether the operation is performed in the Operational Content Execution Window or Operational Content Execution Window (preview).

Table 2–2: When I/O Plugin instance are discarded

No.	Discarded when:	Plugin instance to be discarded
1	Operational Content is displayed in the Operational Content Execution Window or Operational Content Execution Window (preview)[#1]	Instances retained by the HTTP session (when the operation is performed) based on the window ID of the window in which the operation is performed
2	The **Editing Window View** menu is clicked	
3	The **Close** button is clicked[#2]	
4	The **Done** button is clicked[#3]	
5	The **Logout** button is clicked	All instances retained by the HTTP session to be discarded
6	A session times out	

#1

Instances are discarded when you select Operational Content in the menu area. If you specify a parameter for the basic URL, and then open a specific Operational Content, instances are not discarded.

#2

If `logout_close` is specified for the `ucnp.base.server.close.button.setting` property in the user property file (`ucnp_user.properties`), clicking the **Close** button discards the same instance as those when the **Logout** button is clicked in No. 5. For details about the property, see the description about the `ucnp.base.server.close.button.setting` property (whether to display the **Close** button in the header area) in the *Hitachi Navigation Platform Setup and Operations Guide*.

#3

Instances are discarded only when `true` is specified for the `ucnp.base.client.complete.button.window.close` property in the user property file (`ucnp_user.properties`). For details about the property, see the description of the `ucnp.base.client.complete.button.window.close` property (Web browser operation when the **Done** button is clicked in the Operational Content Execution Window) in the *Hitachi Navigation Platform Setup and Operations Guide*.

2.4 Overview of Suspend/Resume Plugins

Suspend/Resume Plugins are plugins that temporarily save information entered by users in the Operational Content Execution Window, and restore the saved information when operations are resumed. If you want to perform operation that requires several days to complete, develop Suspend/Resume Plugins. The entered values are saved as suspend information. Even if you reference information about other operations or log out in the middle of operation, you can resume the operation from the temporarily saved status.

2.4.1 Suspend/Resume Plugin execution timing

Suspend/Resume Plugins are executed at the timing shown below.

For suspending operation:

When a user clicks the **Suspend** button

For resuming operation:

- When a user selects Operational Content in the Operational Content Execution Window
- When a user opens a specific Operational Content by specifying a parameter for the basic URL

2.4.2 Lifecycle of Suspend/Resume Plugins

A Suspend/Resume Plugin terminates by deleting Suspend information when:

- A user clicks the **Done** button in the Operational Content Execution Window.
- An error occurs during suspend information check when the user resumes operation in the Operational Content Execution Window.
- The contents of Operational Content displayed in the Operational Content Execution Window are changed.
- A Content Manager deletes Operational Content in the Operational Content Editing Window.

2.5 Overview of Custom Window Plugins

Custom Window Plugins are plugins that develop new windows. If you want to display your original windows besides the windows of Navigation Platform after you logged in, develop Custom Window Plugins.

2.5.1 Custom Window Plugin execution timing

Custom Window Plugins are executed when:

- You access and log in to Navigation Platform by the URL with the `ucnpUserPageId` parameter specified.
- When you access and log in to Navigation Platform by the URL for which the value acquired by using the `getCustomWindowUrl` method of `CustomWindowUrlUtil` is specified for the `ucnpUserPageId` parameter.

2.5.2 Data that can be received by Custom Window Plugins

To send data to a custom window, you must use the `ucnpUserData` parameter. If you want to receive multiple pieces of data in a new window, combine them, and then specify this combination for the `ucnpUserData` parameter.

You can use the `GET` or `POST` method to send the `ucnpUserData` parameter. You must note the following if you use the `GET` method to send data as part of the URL:

- Specify a URL encoded value in UTF-8.
- Specify a value within the maximum number of characters that can be used for the URL of the Web browser you want to use.

If these conditions are not satisfied, operation is unpredictable.

Note that the `ucnpUserData` parameter takes effect only on the custom window that first appears when you log in to Navigation Platform. This parameter is disabled if you switch to another window from the custom window after the login.

2.5.3 Processing if an error occurs in a Custom Window Plugin

If a Custom Window Plugin is not found or is not running when you log in, a KDCZ10375-E message is displayed in the error window.

3 Preparation of Development

This chapter describes operation, such as setup and specifying environment variable settings, required for developing plugins and custom windows.

Before starting preparation of development, you must set up a development environment, and then create Operational Content (or import it from an editing environment). For details about these tasks, see the *Hitachi Navigation Platform Setup and Operations Guide*.

3.1 Setting up Eclipse

The following shows the procedure for setting up Eclipse.

Important note

Use JDK version 8. However, you can use JDK version 5, 6, or 7 to create libraries that are to be referenced from plugins and custom windows.

By setting up Eclipse according to the following procedure, you can use JDK version 8, which is included in Hitachi Navigation Platform for Developers.

1. Copy the archive of Eclipse to any folder from the CD-ROM of Hitachi Navigation Platform for Developers or JP1/ Navigation Platform for Developers.
2. Extract the copied archive to a folder other than the Navigation Platform for Developers installation directory.
   This folder is referred to as the *Eclipse installation directory* hereafter.
3. Add the following directory to the `Path` system environment variable:
   Eclipse-installation-directory`\plugins\`*directory-beginning-with-org.apache.ant*`_\bin;`
4. Edit the `eclipse.ini` file directly under the Eclipse installation directory.
   Information to add 1
      Add the following setting on a line before `-vmargs`:
      `-vm`
      Navigation-Platform-for-Developers-installation-directory`\PP\uCPSB\jdk\bin\javaw.exe`
   Information to add 2
      Add the following setting on a line after `-vmargs`:
      `-XX:MaxMetaspaceSize=`*128-MB-or-greater-value*
      `-Djava.endorsed.dirs=`*Navigation-Platform-for-Developers-installation-directory*`\PP\uCPSB\jaxp\lib`
   Example:
   ```
   -vm
   C:\Program Files\Hitachi\HNP\PP\uCPSB\jdk\bin\javaw.exe
   -vmargs
   -XX:MaxMetaspaceSize=128m
   -Djava.endorsed.dirs=C:\Program Files\Hitachi\HNP\PP\uCPSB\jaxp\lib
   ```
5. Execute `eclipse.exe` and then make sure that Eclipse starts.
   At this time, make sure that a path under the Navigation Platform for Developers installation directory is not specified for the workspace.

3.2 Importing a pluginSDK project

You can use a pluginSDK project provided by Navigation Platform to prepare the environment required for developing plugins.

To import a pluginSDK project to Eclipse:

1. Start Eclipse.
2. In Eclipse, select **File** and then **Import**.
   The Import window appears.
3. Select **General**, and then **Existing Projects into Workspace**.
4. Click the **Next** button.
5. In the **Select root directory** text box, specify *Navigation-Platform-for-Developers-installation-directory* `\pluginSDK`.
6. Click the **Finish** button.
   The pluginSDK project is added to Eclipse.

Note:
   Make sure that the **Copy projects into workspace** check box is cleared.

3.3 Adding libraries

This section describes how to add Java libraries (JAR files) for use with plugins.

Important note

A user who adds a library must have Windows administrator roles. If a user without Windows administrator roles adds a library to a directory such as the *OS-installation-drive*`:\Program Files` directory (by adding or copying a file), the file might be redirected to a user folder.

3.3.1 Location to place libraries

The following describes the directory that stores a library for use with plugins. The directory that stores a library varies depending on whether the library is shared by the whole J2EE server.

To use the library with plugins only

The library must be placed in the Application Class Loader layer. Store the library in the following directory:

Navigation-Platform-installation-directory`\pluginSDK\plugin\`*plugin-ID*`\WEB-INF\lib`

To share the library on the whole J2EE server

The library must be placed in the System Class Loader layer. Store the library in the directory shown below. In this case, you must place the library in both the development environment and execution environment.

Navigation-Platform-installation-directory`\usrlib\sys`

3.3.2 Creating and configuring libraries

The following describes a note on creating libraries, and how to configure a library you created. Also described is how to use Eclipse to perform reference resolution of the configured library.

(1) Note on creating libraries

To create a library that references APIs provided by Navigation Platform, specify the following library in the class path, and then build the library.

Navigation-Platform-installation-directory`\syslib\ucnpsys.jar`

(2) Configuring a library

To configure a library:

1. Store the created library in the location indicated in *3.3.1 Location to place libraries*.
   You do not need to perform the following steps if you want to use the library only with plugins.
   To share the library in the whole J2EE server, proceed with the next step.
2. Add the absolute path of the library to the class path specification for the J2EE server.
3. Restart Navigation Platform.

To share the library in the whole J2EE server, perform steps 1 to 3 on the J2EE server in an execution environment, in addition to the J2EE server in a development environment.

(3) Project reference resolution

To perform reference resolution for an Eclipse project, use the following procedure to specify the added library JAR file:

1. In the Eclipse Project Explorer view, right-click the project.
   A menu opens.
2. Click **Properties**.
3. In the left pane, click **Java Build Path**.
4. In the right pane, click the **Libraries** tab.
5. Click the **Add JARs** button, and then specify the JAR file you want to add.

3.3.3 Notes on adding libraries

The following are notes on adding libraries:

- When you configure a library in the Application Class Loader layer, do not specify a file name beginning with `ucnp` for the JAR file name of the library.
  Because file names are not case sensitive, you cannot use a file name beginning with `UCNP` or `uCNP`
- Make sure that only the libraries required for development are stored in the location to place libraries. Storing unnecessary files or directories might cause an error during a build process.
- A user who adds a library must have Windows administrator roles. If a user without Windows administrator roles adds a library, the library file is redirected to a user folder and does not work properly.

4 Developing Plugins

This chapter describes how to develop plugins.

4.1 Creating template plugins

To create a template plugin:

1. Edit the plugin information property file.
2. Execute the template plugin creation command.

This section describes details about each step.

> **Reference note**
>
> Perform the tasks described here only once for a plugin. If you modify a created plugin, there is no need to perform these tasks again.

4.1.1 Editing the plugin information property file

A plugin information property file is a property file that defines the information required for creating template plugins. When you execute the template plugin creation command, files and folders are created based on the information defined in the plugin information property file.

Create a plugin information property file by editing the following sample file:

```
Navigation-Platform-for-Developers-installation-directory\pluginSDK
\plugin.properties.sam
```

The file name after editing must be *any-character-string*`.properties`.

(1) Notes on creating a property file

The following describes the notes on creating a plugin information property file by editing the sample file:

- Characters in a property file are encoded in ISO 8859-1 (Latin1). Characters other than those of ISO 8859-1 cannot be used.
- You cannot use Windows reserved device names (`CON`, `AUX`, `COM`*n* (*n*: 1-9), `LPT`*n* (*n*: 1-9), `PRN`, `NUL`, and `CLOCK$`).
- A line that begins with a hash mark (`#`) or exclamation mark (`!`) is assumed to be a comment.
- Empty characters (single-byte spaces, tabs, or line feeds) at the beginning of a line are ignored.

(2) Property file description format

The following shows an example of property file description format. In the description below, *empty character* indicates a single-byte space, tab, or line feed.

```
  property-key=value
```

- Enter a colon (`:`) or equal sign (=) between the property key and the value. Empty characters entered between the property key, colon (or equal sign), and value are ignored.

- If the property key is followed by `:` or = (excluding empty characters), the string from the character (excluding empty characters) just after `:` or = to the end of the line is assumed to be the value.
- If the property key is followed by a character (excluding empty characters) other than `:` or =, the string from that character to the end of the line is assumed to be the value.
- Empty characters added to the end of the value are assumed to be part of the value.
- Colons (`:`), equal signs (=), hash marks (#), and exclamation marks (`!`) contained in the value are assumed to be part of the value.

(3) Details about the property keys used with all plugins

The following describes details about the property keys (specified in the plugin information property file) that are required for all plugins.

`userplugin.id`

Specify a plugin ID. Make sure that the plugin ID is unique within the system. For a Suspend/Resume Plugin, you must always specify the fixed value `ucnp.plugin.suspend`.

Characters that can be used

Single-byte alphanumeric characters

Single-byte periods (`.`)

Single-byte underscores (_)

Important note

The following describes the restrictions on the combination of characters that can be used:

- Only one Suspend/Resume Plugin can be specified within the system. Therefore, you must always specify the fixed value `ucnp.plugin.suspend`.
- For I/O Plugins and Custom Window Plugins, you cannot specify a value ending with a period or beginning with `ucnp`. In addition, because plugin IDs are not case sensitive, you cannot specify `UCNP` or `uCnp`.

String length that can be specified

1 to 64 bytes

Specification example

`userplugin`

`userplugin.name`

Specify a plugin name. Make sure that the plugin name is unique within the system.

Characters that can be used

Single-byte alphanumeric characters

Prohibited plugin names

Plugin names beginning with `ucnp` (not case sensitive)

`env`

`AppName`

String length that can be specified

1 to 31 bytes

Specification example

`userplugin`

`userplugin.type`

Specify the plugin type.

Characters that can be used

For I/O Plugins: `TYPE_IO`

For Suspend/Resume Plugin: `TYPE_SUSPEND`

For Custom Window Plugins: `TYPE_WINDOW`

`userplugin.version`

Specify the plugin version.

Characters that can be used

Single-byte alphanumeric characters

Single-byte periods (`.`)

Single-byte underscores (_)

Single-byte hyphens (`-`)

String length that can be specified

1 to 32 bytes

Specification example

`00.01`

`userplugin.java.package`

Specify a Java package name.

Characters that can be used

Single-byte alphanumeric characters

Single-byte periods (`.`)

Character strings that are valid as Java package names

Character string not used for creating a directory that has the same name as a Windows reserved device name

Prohibited package names

Package names beginning with `jp.co.hitachi.soft.ucnp`

String length that can be specified

1 or more bytes

However, make sure that the sum of the values specified for *Java-package-name*, *plugin-ID*, and *I/O-action-controller-class-name*, or *Java-package-name*, *plugin-ID*, and *suspend/resume-action-controller-class-name* is no more than 128 bytes.

Specification example

`sample.userplugin`

(4) Details about the property keys used with I/O Plugins

The following describes the property keys that must be set if you specify `TYPE_IO` for the `userplugin.type` property key.

`userplugin.server.controller.ioaction`

Specify the I/O action controller class name. In *4.1.2 Executing the template plugin creation command*, a controller class is created based on this class name.

Characters that can be used

Single-byte alphanumeric characters

Single-byte underscores (_)

Character strings that are valid as Java class names

Character strings that do not contain Windows reserved device names

Character strings that contain a unique value combined with a Java package name

String length that can be specified

1 or more bytes

However, make sure that the sum of the values specified for *Java-package-name*, *plugin-ID*, and *I/O-action-controller-class-name* is no more than 128 bytes.

Specification example

```
IoPluginController
```

`userplugin.server.controller.ioaction.type`

Specify the character string that identifies the I/O action controller class. Specify a unique value within the system. In *4.1.2 Executing the template plugin creation command*, the I/O plugin XML file (`ioaction.xml`) is created based on this character string. Note that the I/O plugin XML file defines the information displayed in the **Plugins** palette and Plugin Parts in the Operational Content Editing Window. Generally, specify the same value as the ID of the template plugin to be created. However, you can specify a different value.

Characters that can be used

Single-byte alphanumeric characters

Single-byte periods (`.`)

Single-byte underscores (_)

Single-byte hyphens (`-`)

String length that can be specified

1 to 64 bytes

Specification example

```
userplugin
```

(5) Details about the property key used with Suspend/Resume Plugins

The following describes the property key that must be set if you specify `TYPE_SUSPEND` for the `userplugin.type` property key.

`userplugin.server.controller.suspend`

Specify the suspend/resume action controller class name.

Characters that can be used

Single-byte alphanumeric characters

Single-byte underscores (_)

Character strings that are valid as Java class names

Character strings that do not contain Windows reserved device names

Character string containing a unique value combined with a Java package name

String length that can be specified

1 or more bytes

However, make sure that the sum of the values specified for *Java-package-name*, *plugin-ID*, and *suspend/resume-action-controller-class-name* is no more than 128 bytes.

Specification example

`SuspendActionController`

(6) Property file coding example

The following shows examples of coding plugin information property files.

For an I/O Plugin

```
userplugin.id = example.inputdata
userplugin.name = InputData
userplugin.type = TYPE_IO
userplugin.version = 01.00
userplugin.java.package = com.example.inputdata
userplugin.server.controller.ioaction = InputDataController
userplugin.server.controller.ioaction.type = example.inputdata
```

For a Suspend/Resume Plugin

```
userplugin.id = ucnp.plugin.suspend
userplugin.name = Suspend
userplugin.type = TYPE_SUSPEND
userplugin.version = 01.00
userplugin.java.package = com.example.suspend
userplugin.server.controller.suspend = SuspendActionController
```

For a Custom Window Plugin

```
userplugin.id = example.contentslist
userplugin.name = ContentsList
userplugin.type = TYPE_WINDOW
userplugin.version = 01.00
userplugin.java.package = com.example.customwindow
```

4.1.2 Executing the template plugin creation command

You can execute the template plugin creation command to create template plugins. Template plugins are created in the format of an Eclipse Java project.

(1) Format of the template plugin creation command

The following shows the format of the template plugin creation command:

```
Navigation-Platform-for-Developers-installation-directory\pluginSDK\bin
\npcreateplg.batΔpath-to-the-plugin-information-property-file
```

When you execute the template plugin creation command, a directory is created under *Navigation-Platform-for-Developers-installation-directory*`\pluginSDK\plugin` according to the contents of the plugin information property file.

To create multiple plugins, repeat the process of editing the plugin information property file and create a template plugin for the number of plugins.

(2) Execution results of the template plugin creation command (for I/O Plugins)

The following shows the directory structure for I/O Plugins. You need to edit only the underlined files. Bold text indicates the values specified in the plugin information property file. For details about how to edit the plugin information property file, see *4.1.1 Editing the plugin information property file*.

```
Navigation-Platform-for-Developers-installation-directory\pluginSDK\plugin
 |- plugin-ID
    |-.project
    |-.classpath
    |-build.xml
    |-plugin.properties
    |-ucnpsdkversion.properties
    |--js
    |--dd
    |  |--META-INF
    |      |-application.xml
    |      |-cosminexus.xml
    |--images
    |    |-sample_icon.gif
    |--WEB-INF
        |  |-plugin.xml
        |  |-web.xml
        |--lib
        |--conf
        |  |-ioaction.xml
        |  |-ucnp_label_plugin-ID.properties
        |  |-ucnp_message_plugin-ID.properties
        |  |-ucnp_plugin-ID.properties
        |
        |--src
            |--Java-package-name
                |--controller
                    |-I/O-action-controller-lass-name.java
                    |-PluginInitializer.java
```

(3) Execution results of the template plugin creation command (for Suspend/Resume Plugins)

The following shows the directory structure for Suspend/Resume Plugins. You need to edit only the underlined files. Bold text indicates the values specified in the plugin information property file. For details about how to edit the plugin information property file, see *4.1.1 Editing the plugin information property file*.

```
Navigation-Platform-for-Developers-installation-directory\pluginSDK\plugin
|- ucnp.plugin.suspend
    |-.project
```

```
      |-.classpath
      |-build.xml
      |-plugin.properties
      |-ucnpsdkversion.properties
      |--js
      |--dd
      |   |--META-INF
      |       |-application.xml
      |       |-cosminexus.xml
      |--WEB-INF
          |   |-plugin.xml
          |   |-web.xml
          |--lib
          |--conf
          |   |-ucnp_label_ucnp.plugin.suspend.properties
          |   |-ucnp_message_ucnp.plugin.suspend.properties
          |   |-ucnp_ucnp.plugin.suspend.properties
          |--src
              |--Java-package-name
                  |--controller
                      |-suspend/resume-action-controller-class-name.java
                      |-PluginInitializer.java
```

(4) Execution results of the template plugin creation command (for Custom Window Plugins)

The following shows the directory structure for Custom Window Plugins. You need to edit only the underlined files. Bold text indicates the values specified in the plugin information property file. For details about how to edit the plugin information property file, see *4.1.1 Editing the plugin information property file*.

```
Navigation-Platform-for-Developers-installation-directory\pluginSDK\plugin
|-plugin-ID
      |-.project
      |-.classpath
      |-build.xml
      |-plugin.properties
      |-ucnpsdkversion.properties
      |--js
      |--jsp
      |   |--sys
      |   |   |-ucnpCustom.jsp
      |   |-plugin-name.jsp
      |--dd
      |   |--META-INF
      |       |-application.xml
      |       |-cosminexus.xml
      |--WEB-INF
          |   |-plugin.xml
          |   |-web.xml
          |--lib
          |--conf
          |   |-ucnp_label_plugin-ID.properties
          |   |-ucnp_message_plugin-ID.properties
          |   |-ucnp_plugin-ID.properties
          |--lib
          |--src
```

```
              |--Java-package-name
                  |--controller
                      |-PluginInitializer.java
```

(5) Notes on executing the template plugin creation command

The following provides the notes on executing the command:

- If multiple commands are executed at the same time, operation is unpredictable.
- If a directory having the same name as the plugin ID specified in the plugin information property file exists under *Navigation-Platform-for-Developers-installation-directory*`\pluginSDK\plugin`, an overwrite confirmation message appears. Specify as follows in response to the confirmation message.
    - To overwrite the existing directory:
      Specify `Y` or `y`.
    - Not to overwrite the existing directory:
      Specify `N` or `n`. Processing is canceled.

      In this case, change the plugin ID specified in the plugin information property file, and then re-execute the command.
- To execute the template plugin creation command, you must first open the command prompt by selecting **Run as administrator**. If this condition is not satisfied, operation is unpredictable.
- Encoding of Java source files created by the template plugin creation command differs depending on the version of Navigation Platform for Developers. Navigation Platform for Developers 10-00 or earlier uses Shift_JIS, and 10-10 or later uses UTF-8. Therefore, if you use a Java source file created in 10-00 or earlier, you must be careful about the difference in the encoding type.

4.2 Importing a template plugin project

The following shows the procedure for importing a template plugin project to Eclipse.

Reference note

Perform this task only once for one plugin. This task is not needed when you modify a created plugin.

1. Start Eclipse.
2. In Eclipse, select **File** and then **Import**.
   The Import window appears.
3. Select **General**, and then **Existing Projects into Workspace**.
4. Click the **Next** button.
5. In the **Select root directory** text box, specify the template plugin directory created in *4.1 Creating template plugins*.
6. Click the **Finish** button.
   The template plugin project is added to Eclipse.

Notes:

- Do not select the **Copy projects into workspace** check box.
- To start Eclipse, right-click the `eclipse.exe` file, and then select **Run as administrator**. If this condition is not satisfied, operation is unpredictable.
- Depending on the Eclipse version, an error message indicating that the workspace is being refreshed might appear during import. If this error message appears, delete the imported project, and then import it again.
  To prevent such errors, you need to close or refresh the pluginSDK project before you start the import. Right-click the pluginSDK project displayed in the Project Explorer view, and then select **Close Project** or **Update**.

4.3 Customizing template plugins

You can customize template plugins by using Eclipse.

4.3.1 Customizing I/O Plugins

This subsection describes the procedure for customizing I/O Plugins.

To customize a template plugin created as an I/O Plugin:

1. Create the icon of the **Plugins** palette and Plugin Parts.
2. Edit the I/O plugin XML file.
3. Set tool tips for parameter descriptions.
4. Specify the plugin execution order.
5. Specify whether to display a confirmation dialog box.
6. Specify whether to execute plugins in the preview window.
7. Specify the button type for suppressing execution.
8. Implement processing to be performed by the plugin.

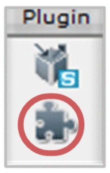
The following describes details about each step.

(1) Creating the icon of the [Plugins] palette and Plugin Parts

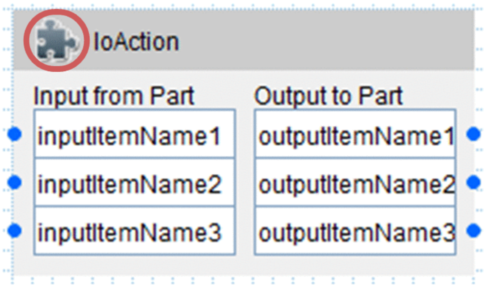
If necessary, change the icon of the **Plugins** palette and Plugin Parts displayed in the Operational Content Editing Window. The following figure shows the location of the icon of the **Plugins** palette and Plugin Part.

Figure 4–1: Location of the icon of the Plugins palette and Plugin Part

■ Icon in the **Plugins** palette



■ Icon for the Plugin Part



When you execute the template plugin creation command, the icons provided by Navigation Platform by default are displayed in the **Plugins** palette and Plugin Parts.

To develop multiple plugins, you can create a different icon for each plugin to identify plugins by displayed icons. Store the created icons in the following directory:

```
Navigation-Platform-for-Developers-installation-directory\pluginSDK\plugin
\plugin-ID\images
```

Create icons in the format shown in the following table.

Table 4–1: Format for creating icons

No.	Item	Description
1	File name	See *(2) Editing I/O plugin XML files*, and then specify the file name. Note that the file name of the default icon provided by Navigation Platform is `sample_icon.gif`.
2	File format	You can use any format that can be displayed in the Web browser. Note that the file format of the default icon provided by Navigation Platform is GIF format.
3	Size	Create an icon in 24 **x** 24 pixels.

Note:

If a user without Windows administrator roles creates an icon in a directory such as the *OS-installation-drive*: `\Program Files` directory (by adding or copying a file), the file might be redirected to a user folder. Therefore, the user that adds the file must have Windows administrator roles.

(2) Editing I/O plugin XML files

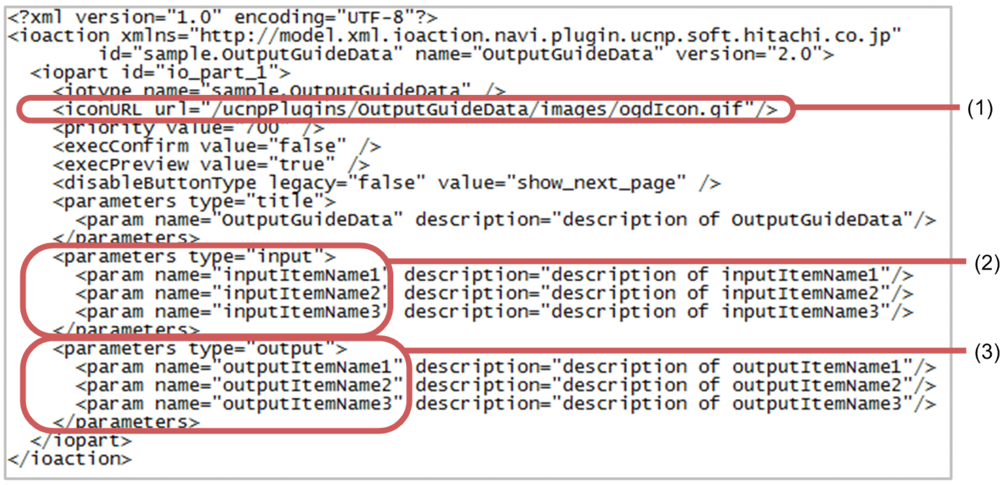
The I/O plugin XML file (`ioaction.xml`) is created when a template plugin is created. Edit the I/O plugin XML file to change the following information:

- I/O Plugin execution order
- I/O parameter definition
- Information displayed in the **Plugins** palette, for Plugin Parts, and for tool tips in the Operational Content Editing Window

The following figure shows the correspondence between the information specified in the I/O plugin XML file and the information displayed in the Operational Content Editing Window.

Figure 4–2: Correspondence between the I/O plugin XML file and the information displayed in the Operational Content Editing Window

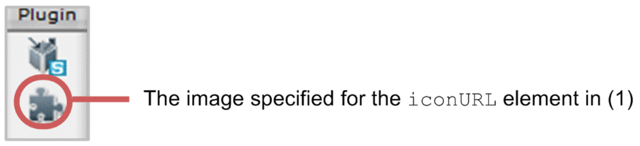
■ Contents of the I/O plugin XML file



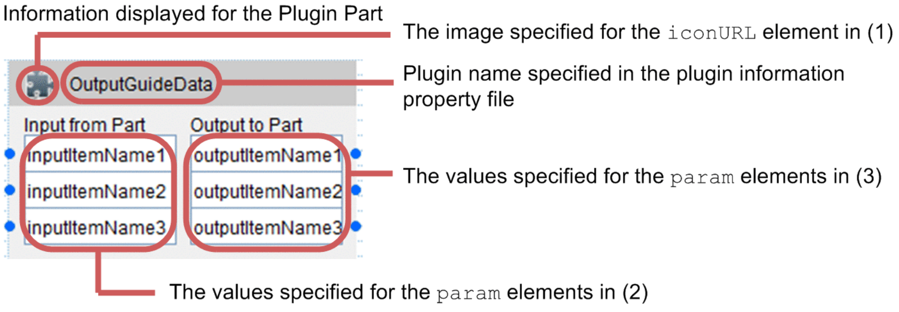
```
<?xml version="1.0" encoding="UTF-8"?>
<ioaction xmlns="http://model.xml.ioaction.navi.plugin.ucnp.soft.hitachi.co.jp"
        id="sample.OutputGuideData" name="OutputGuideData" version="2.0">
  <iopart id="io_part_1">
    <iotype name="sample.OutputGuideData" />
    <iconURL url="/ucnpPlugins/OutputGuideData/images/ogdIcon.gif"/>
    <priority value="700" />
    <execConfirm value="false" />
    <execPreview value="true" />
    <disableButtonType legacy="false" value="show_next_page" />
    <parameters type="title">
      <param name="OutputGuideData" description="description of OutputGuideData"/>
    </parameters>
    <parameters type="input">
      <param name="inputItemName1" description="description of inputItemName1"/>
      <param name="inputItemName2" description="description of inputItemName2"/>
      <param name="inputItemName3" description="description of inputItemName3"/>
    </parameters>
    <parameters type="output">
      <param name="outputItemName1" description="description of outputItemName1"/>
      <param name="outputItemName2" description="description of outputItemName2"/>
      <param name="outputItemName3" description="description of outputItemName3"/>
    </parameters>
  </iopart>
</ioaction>
```

■ Information displayed in the Operational Content Editing Window

● Display contents of the **Plugins** palette



● Information displayed for the Plugin Part


The image specified for the iconURL element in (1)
OutputGuideData
Plugin name specified in the plugin information property file
Input from Part
Output to Part
inputItemName1
inputItemName2
inputItemName3
outputItemName1
outputItemName2
outputItemName3
The values specified for the param elements in (3)
The values specified for the param elements in (2)

Tip

If you change the I/O plugin XML file (`ioaction.xml`) for a plugin associated with Operational Content, you also need to perform tasks such as replacing the plugin by exporting Operational Content. When you design I/O Plugins, make sure that item settings are suitable for operation of Operational Content, so that you do not have to modify the I/O plugin XML file later.

The following describes how to edit the I/O plugin XML file (`ioaction.xml`). This file is stored in the following directory.

```
Navigation-Platform-for-Developers-installation-directory\pluginSDK\plugin
\plugin-ID\WEB-INF\conf
```

To edit the I/O plugin XML file, start the editor by selecting **Run as administrator**.

The following shows an example of editing the I/O plugin XML file (`ioaction.xml`). Edit the bold text.

```
<?xml version="1.0" encoding="UTF-8"?>
<ioaction xmlns="http://
model.xml.ioaction.navi.plugin.ucnp.soft.hitachi.co.jp"
id="sample.OutputGuideData" name="OutputGuideData" version="2.0">
  <iopart id="io_part_1">
    <iotype name="sample.OutputGuideData" />
    <iconURL url="/ucnpPlugins/OutputGuideData/images/ogdIcon.gif"/>
    <priority value="700" />
    <execConfirm value="false" />
    <execPreview value="true" />
    <disableButtonType legacy="false" value="show_next_page" />
    <parameters type="title">
      <param name="OutputGuideData" description="description-of-
OutputGuideData"/>
    </parameters>
    <parameters type="input">
      <param name="inputItemName1" description="description-of-
inputItemName1"/>
      <param name="inputItemName2" description="description-of-
inputItemName2"/>
      <param name="inputItemName3" description="description-of-
inputItemName3"/>
    </parameters>
    <parameters type="output">
      <param name="outputItemName1" description="description-of-
outputItemName1"/>
      <param name="outputItemName2" description="description-of-
outputItemName2"/>
      <param name="outputItemName3" description="description-of-
outputItemName3"/>
    </parameters>
  </iopart>
</ioaction>
```

The table below describes how to edit the I/O plugin XML file (`ioaction.xml`). You need to edit only the parts shown in the table.

Table 4–2: How to edit the I/O plugin XML file (ioaction.xml)

No.	Information to be edited	Description	Character strings specifiable for the attribute value, and their meanings	String length specifiable for the attribute value
1	`url` attribute of the `<iconURL>` tag (file name part)	Specify the file name of the icon displayed in the **Plugins** palette and for Plugin Parts in the Operational Content Editing Window. Change the file name in the `url` attribute only when you want to change the file name of the icon created in *(1) Creating the icon of the [Plugins] palette and Plugin Parts* from the default.	• Single-byte alphanumeric characters • Single-byte underscores (_) • Single-byte periods (.) • Single-byte hyphens (-)	Maximum of 1,024 bytes (including `/ucnpPlugins/`*plugin-name*`/images/`)
2	`value` attribute of the `<priority>` tag	Specify the execution order of the I/O Plugin. The default value is `700`. 700 is the intermediate value of the execution	The following single-byte numeric values: • 100-500 • 600-900	--

No.	Information to be edited	Description	Character strings specifiable for the attribute value, and their meanings	String length specifiable for the attribute value
2	`value` attribute of the `<priority>` tag	order of System Plugins and User Plugins. If the `priority` element is omitted, the default value is assumed.	The following single-byte numeric values: • 100-500 • 600-900	--
3	`value` attribute of the `<execConfirm>` tag	If necessary, specify whether to display a confirmation dialog box during execution of the I/O Plugin. Specify `true` if you want to notify users that processing might require some time. If the `execConfirm` element is omitted, `false` is assumed.	• `true` Displayed • `false` Not displayed	--
4	`value` attribute of the `<execPreview>` tag	If necessary, specify whether to execute the I/O Plugin in the preview window.#1 If the `execPreview` element is omitted, `true` is assumed.	• `true` Executed • `false` Not executed	--
5	`legacy` attribute of the `<disableButtonType>` tag	Specify whether to provide compatibility of execution suppression of the I/O Plugin. Specify `true` to suppress execution of the I/O Plugin in Navigation Platform 10-10 the same way as in earlier versions. If the `disableButtonType` element is omitted, `true` is assumed.	• `true` Compatible • `false` Not compatible	--
6	`value` attribute of the `<disableButtonType>` tag	If necessary, specify whether to suppress execution of the I/O Plugin depending on the button type.#2 Set the `legacy` attribute to `false` to suppress the execution depending on whether transition occurs, by clicking the button or by directly selecting a node. To specify multiple values, use a single-byte comma to separate each value. If the `disableButtonType` element is omitted, an empty character string is assumed.	• `start` • `show_next_page` • `show_previous_page` • `show_complete_page` • `forward_jump` Valid only if the value of the `legacy` attribute is `false` • `back_jump` Valid only if the value of the `legacy` attribute is `false`	--
7	`description` attribute of the <param> tag	If necessary, specify the description (tool tip) of the I/O Plugin.	Any character string#3	0 to 64 characters
8	`name` attribute of the `<param>` tag under `<parameters type="input">`	Specify the input parameter name. Specify a unique value in the `<parameters type="input">` tag. You can specify 0 to 100 `<param>` tags. Specify the `<param>` tags for the number of input parameters.	• Single-byte alphanumeric characters • Single-byte underscores (_) • Single-byte periods (`.`) • Single-byte spaces • Single-byte hyphens (`-`)	1 to 64 bytes#4
9	`description` attribute of the `<param>` tag	If necessary, specify the description (tool tip) of the input parameter.	Any character string#3	0 to 64 characters

No.	Information to be edited	Description	Character strings specifiable for the attribute value, and their meanings	String length specifiable for the attribute value
9	under `<parameters type="input">`	If necessary, specify the description (tool tip) of the input parameter.	Any character string[#3]	0 to 64 characters
10	`name` attribute of the `<param>` tag under `<parameters type="output">`	Specify the output parameter name. Specify a unique value in the `<parameters type="output">` tag. You can specify 0 to 100 `<param>` tags. Specify the `<param>` tags for the number of output parameters.	• Single-byte alphanumeric characters • Single-byte underscores (_) • Single-byte periods (`.`) • Single-byte spaces • Single-byte hyphens (`-`)	1 to 64 bytes[#4]
11	`description` attribute of the `<param>` tag under `<parameters type="output">`	If necessary, specify the description (tool tip) of the output parameter.	Any character string[#3]	0 to 64 characters

Legend:

--: Not applicable

#1

Processing that determines whether to execute the I/O Plugin in the preview window can also be implemented by using the value of the `ucnp.screen.ispreview` key in the `inputFromNode` and `outputToNode` methods of the server processing implementation interface (`IIoPluginController`). However, the setting of the `ucnp.screen.ispreview` key is used to perform special processing by the server processing implementation interface (`IIoPluginController`). Therefore, if you want to simply skip the processing, specify the setting in the I/O plugin XML file (`ioaction.xml`). For details about the server processing implementation interface (`IIoPluginController`), see *5.2 IIoPluginController (server processing implementation interface)*.

#2

Processing that determines whether to suppress I/O Plugin execution depending on the button type can also be implemented by using the value of the `ucnp.button.type` key in the `inputFromNode` and `outputToNode` methods of the server processing implementation interface (`IIoPluginController`). However, the setting of the `ucnp.button.type` key is used to perform special processing by the server processing implementation interface (`IIoPluginController`). Therefore, if you want to simply skip processing, specify the setting in the I/O plugin XML file (`ioaction.xml`). For details about the server processing implementation interface (`IIoPluginController`), see *5.2 IIoPluginController (server processing implementation interface*.

#3

The character string specified for `description` is used as is for the HTML attribute when the tool tip is displayed. Therefore, if you want to enter a line feed in the character string displayed as the tool tip, specify `&#x0A;`. If a control character other than `&#x0A;` is specified, the display is unpredictable.

#4

If a long character string is specified, ending characters might not be displayed for a Plugin Part. For the I/O action controller class, the character string specified here is used as is.

Reference

Values specified for the `name` attribute of the `<param>` tag under `<parameters type="input">` and `<parameters type="output">` are used for the `param` parameter and return values of the `inputFromNode` and `outputToNode` methods of the server processing implementation interface (`IIoPluginController`). The following shows where the values specified for the `name` attribute are used.

`param` parameter

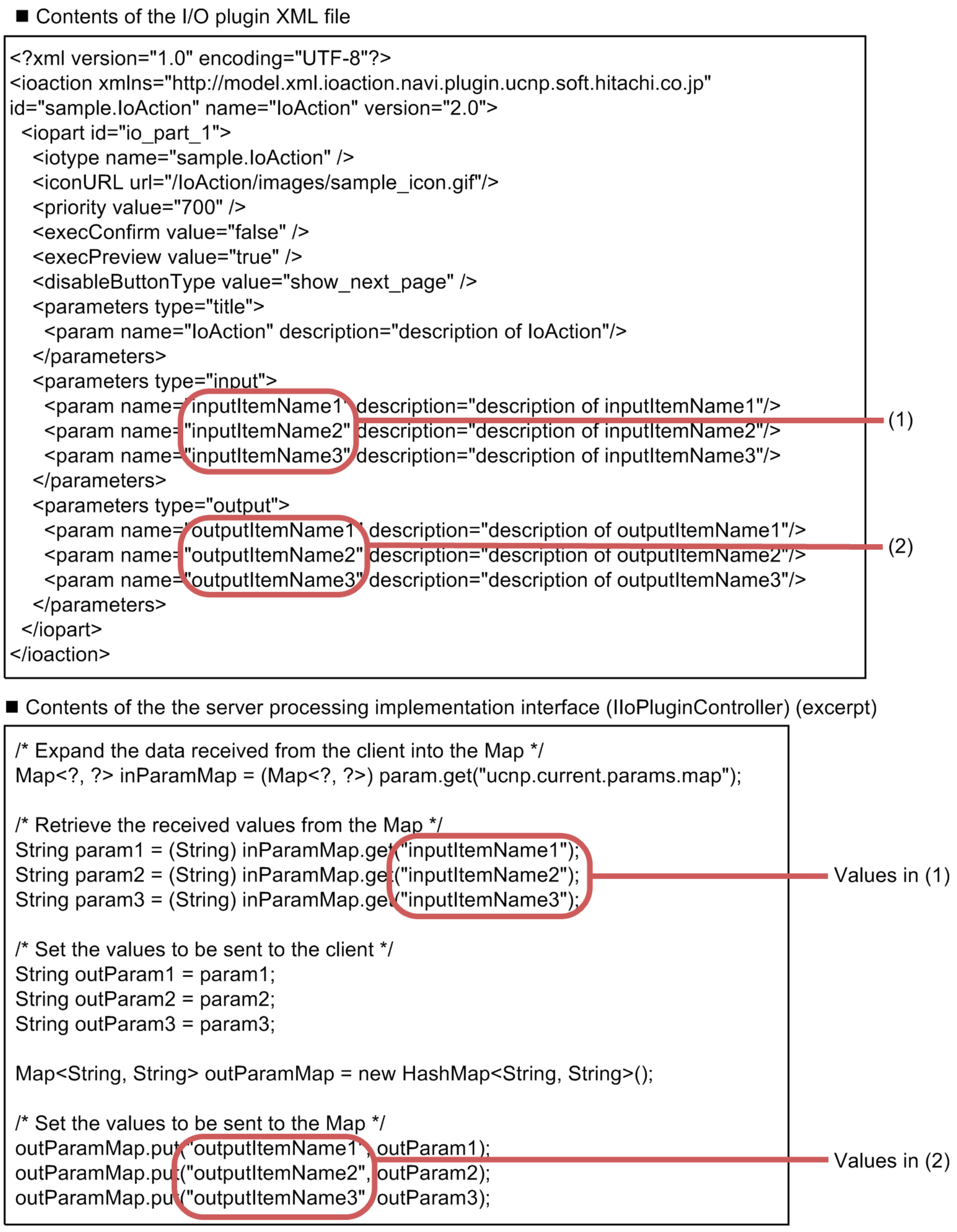
- Map object name for the value corresponding to the name `ucnp.current.params.map` in the `inputFromNode` method
- Map object name for the value corresponding to the name `ucnp.next.params.map` in the `outputToNode` method

Return values

- Map object name for the value corresponding to the name `ucnp.current.params.map` in the `inputFromNode` method
- Map object name for the value corresponding to the name `ucnp.next.params.map` in the `outputToNode` method

The following figure shows the correspondence between the values specified for the `name` attribute and the server processing implementation interface (`IIoPluginController`).

Figure 4–3: Correspondence between the values specified for the name attribute and the server processing implementation interface (IIoPluginController)



■ Contents of the I/O plugin XML file

```
<?xml version="1.0" encoding="UTF-8"?>
<ioaction xmlns="http://model.xml.ioaction.navi.plugin.ucnp.soft.hitachi.co.jp"
id="sample.IoAction" name="IoAction" version="2.0">
  <iopart id="io_part_1">
    <iotype name="sample.IoAction" />
    <iconURL url="/IoAction/images/sample_icon.gif"/>
    <priority value="700" />
    <execConfirm value="false" />
    <execPreview value="true" />
    <disableButtonType value="show_next_page" />
    <parameters type="title">
      <param name="IoAction" description="description of IoAction"/>
    </parameters>
    <parameters type="input">
      <param name="inputItemName1" description="description of inputItemName1"/>
      <param name="inputItemName2" description="description of inputItemName2"/>
      <param name="inputItemName3" description="description of inputItemName3"/>
    </parameters>
    <parameters type="output">
      <param name="outputItemName1" description="description of outputItemName1"/>
      <param name="outputItemName2" description="description of outputItemName2"/>
      <param name="outputItemName3" description="description of outputItemName3"/>
    </parameters>
  </iopart>
</ioaction>
```

■ Contents of the the server processing implementation interface (IIoPluginController) (excerpt)

```
/* Expand the data received from the client into the Map */
Map<?, ?> inParamMap = (Map<?, ?>) param.get("ucnp.current.params.map");

/* Retrieve the received values from the Map */
String param1 = (String) inParamMap.get("inputItemName1");
String param2 = (String) inParamMap.get("inputItemName2");
String param3 = (String) inParamMap.get("inputItemName3");

/* Set the values to be sent to the client */
String outParam1 = param1;
String outParam2 = param2;
String outParam3 = param3;

Map<String, String> outParamMap = new HashMap<String, String>();

/* Set the values to be sent to the Map */
outParamMap.put("outputItemName1", outParam1);
outParamMap.put("outputItemName2", outParam2);
outParamMap.put("outputItemName3", outParam3);
```

For details about the server processing implementation interface (`IIoPluginController`), see *5.2 IIoPluginController (server processing implementation interface)*.

(3) Setting tool tips for parameter descriptions

If you set a description text as a tool tip in the I/O plugin XML file (`ioaction.xml`), the description appears when you point the Plugin Part in the Operational Content Editing Window. You can set the following tool tips:

- I/O Plugin description
  A tool tip displayed in the title of the Plugin Part
- Input parameter description
  A tool tip displayed for input parameters of the Plugin Part
- Output parameter description
  A tool tip displayed for output parameters of the Plugin Part

For details about how to set the tool tips and details about the values, see *(2) Editing I/O plugin XML files*.

(4) Specifying the plugin execution order

In the I/O plugin XML file (`ioaction.xml`), you can change the plugin execution order. You can specify the following values for the execution order:

- 100-500
- 600-900

If multiple I/O Plugins are placed in the same node, they are executed in the ascending order specified for the execution order. If the same execution order is specified for multiple I/O Plugins, the execution order will be undefined.

For details about the values, see *(2) Editing I/O plugin XML files*.

(5) Specifying whether to display a confirmation dialog box

In the I/O plugin XML file (`ioaction.xml`), you can specify whether to display a confirmation dialog box during execution of the I/O Plugin. You can specify the following values:

- `true`
- `false`

If `true` is specified for a plugin placed in the node, a confirmation dialog box for the I/O Plugin appears when the `inputFromNode` method is executed. Even if `true` is specified for multiple plugins in the node, a confirmation dialog box appears only once. If **Cancel** is selected in the confirmation dialog box, the KDCZ00266-Q message is displayed, and then processing of all plugins set in the node is not executed.

For details about the values, see *(2) Editing I/O plugin XML files*.

(6) Specifying whether to execute plugins in the preview window

In the I/O plugin XML file (`ioaction.xml`), you can specify whether to execute the I/O Plugin in the preview window. You can specify the following values:

- `true`
- `false`

If multiple plugins are placed in the node, only the I/O Plugins for which `true` is specified are executed.

For details about the values, see *(2) Editing I/O plugin XML files*.

(7) Specifying the button type for suppressing execution

In the I/O plugin XML file (`ioaction.xml`), you can specify the button type for suppressing execution of I/O Plugins. You can specify the values shown below.

Table 4–3: Button type for suppressing execution of I/O Plugins

No.	Compatibility option (value of the "legacy" attribute)	Setting of the button type for suppressing execution (value of the "value" attribute)	Operation to suppress execution of the I/O Plugin
1	`false`	`start`	Switch from the Terminal Node (start) to a Process Node. Alternatively, display the first Process Node not connected to the Terminal Node (start).
2		`show_next_page`	Switch to the next node by clicking the button. Note, however, that if you switch to a Process Node for which the transition destination node does not exist and for which the **Back** button is hidden, `show_complete_page` is set.
3		`show_previous_page`	Switch to the previous node by clicking the button.
4		`show_complete_page`	Switch to a node with the **Done** button displayed. Alternatively, switch to a Process Node that does not have the transition destination node and for which the **Back** button is not displayed.
5		`back_jump`	Switch to the next node by directly selecting the node.
6		`forward_jump`	Switch to the previous node by directly selecting the node.
7	`true`	`start`	Switch from the Terminal Node (start) to a Process Node. Alternatively, display the first Process Node not connected to the Terminal Node (start).
8		`show_next_page`	Switch to a node by clicking the button or directly selecting the node. Note, however, that if you switch to a Process Node for which the transition destination node does not exist and for which the **Back** button is hidden, `show_complete_page` is set.
9		`show_previous_page`	Switch to the previous node by clicking the button or directly selecting the node.
10		`show_complete_page`	Switch to a node with the **Done** button displayed. Alternatively, switch to a Process Node for which the transition destination node does not exist and for which the **Back** button is hidden.

If multiple I/O Plugins are placed in the node, the I/O Plugins for which execution is suppressed are not executed.

For details about the values, see *(2) Editing I/O plugin XML files*.

(8) Implementing processing to be performed by the plugin

To implement the server processing in the I/O Plugin, specify the processing in the class whose name is specified by the `userplugin.server.controller.ioaction` key in the plugin information property file. At this time, specify the processing to be performed during node transition.

If you need to perform processing during plugin initialization or termination, see *4.5 Implementing processing to be performed during plugin initialization or termination*.

4.3.2 Customizing Suspend/Resume Plugins

This subsection describes the procedure for customizing Suspend/Resume Plugins.

To customize a template plugin created as a Suspend/Resume Plugin:

1. Configure the user property file.
2. Implement processing to be performed by the plugin.

The following describes details about each step.

(1) Configuring the user property file

See *4.9 Configuring the user property file*, and then edit the user property file.

(2) Implementing processing to be performed by the plugin

To implement the suspend information reference and update processing on the Suspend/Resume Plugin, specify the processing in the class whose name is specified by the `userplugin.server.controller.suspend` key in the plugin information property file.

If you need to perform processing during plugin initialization or termination, see *4.5 Implementing processing to be performed during plugin initialization or termination*.

4.3.3 Customizing Custom Window Plugins

This subsection describes the procedure for customizing Custom Window Plugins.

To customize a template plugin created as a Custom Window Plugin:

1. Edit the custom window JSP file.
2. Implement processing to be performed by the plugin.

(1) Editing the custom window JSP file

See *4.3.4 Creating the JSP file used in custom windows*, and then implement the processing required for the custom window JSP file.

(2) Implementing processing to be performed by the plugin

If you need to perform processing during plugin initialization or termination, see *4.5 Implementing processing to be performed during plugin initialization or termination*.

4.3.4 Creating the JSP file used in custom windows

When you execute the template plugin creation command with the plugin information property file in which Custom Window Plugin information is defined, the custom window JSP file (`ucnpCustom.jsp`) is created. Do not edit `ucnpCustom.jsp`. JSP files for editing are created in the following directory:

Navigation-Platform-for-Developers-installation-directory`\pluginSDK\plugin\`*plugin-ID*`\jsp\`*plugin-name*`.jsp`

JSP files for editing (*plugin-name*`.jsp`) are included in `ucnpCustom.jsp`. Because Navigation Platform loads `ucnpCustom.jsp` and performs processing, JSP files for editing cannot use the library placed on the layer of the Application Class Loader. Use the library placed on the layer of the System Class Loader.

The following describes the items that can be specified in the JSP file for editing (*plugin-name*`.jsp`).

Table 4–4: Items that can be specified in a JSP file used in custom windows

No.	Category	Item	Description
1	Tag	Directive	Specify a directive in the following format: `<%@` *directive*`%>`
2		Scriptlet	Specify a scriptlet in the following format: `<%` *Java-code* `%>`
3		Expression	Specify an expression in the following format: `<%=` *expression* `%>`
4		Comment	Specify a comment in the following format: `<%--` *comment* `--%>`
5	Directive	page	Define information such as JSP file encoding and Java `import` statement. To use a different encoding from that of the custom window JSP file (`ucnpCustom.jsp`), specify the `pageEncoding` attribute. Do not specify the following attributes: `contentType` attribute `"text/html; charset=UTF-8"` is automatically applied during execution of the JSP file. `language` attribute `"java"` is automatically applied during execution of the JSP file.
6		include	Include other files such as a text file and JSP file.
7	Implicit object	request	Object variable of the `javax.servlet.http.HttpServletRequest` class
8		response	Object variable of the `javax.servlet.http.HttpServletResponse` class
9		session	Object variable of the `javax.servlet.http.HttpSession` class

4.4 Adding database connection processing

To add database connection processing to a plugin, specify *DB Connector* as a Cosminexus resource adapter on the J2EE server. The required tasks are as follows:

1. Configure (add, change settings of, or delete) a resource adapter.
2. Set a plugin.
3. Implement database connection processing.

The database and resource adapter to be connected are as follows:

Database

HiRDB Single Server 08-00 or later

Resource adapter

DB_Connector_for_HiRDB_Type4

This resource adapter is imported when Navigation Platform is set up.

4.4.1 Adding a resource adapter

To add a resource adapter, you need the Connector attribute file. The following describes the procedure:

1. Execute the command as follows to display a list of resource adapters, and then make sure that a resource adapter named `DB_Connector_for_HiRDB_Type4` is not found:

```
Navigation-Platform-installation-directory\PP\uCPSB\CC\admin\bin
\cjlistrar uCNP_J2EE
```

2. Execute the command as follows to deploy the `DB_Connector_for_HiRDB_Type4` resource adapter:

```
Navigation-Platform-installation-directory\PP\uCPSB\CC\admin\bin
\cjdeployrar uCNP_J2EE -resname DB_Connector_for_HiRDB_Type4
```

3. Display a list of resource adapters again, and then make sure that the `DB_Connector_for_HiRDB_Type4` resource adapter has been added.

4. Execute the command as follows to acquire the Connector attribute file for the resource adapter you added:

```
Navigation-Platform-installation-directory\PP\uCPSB\CC\admin\bin
\cjgetrarprop uCNP_J2EE -resname DB_Connector_for_HiRDB_Type4 -c "C:\temp
\TP_Connector_for_HiRDB_Type4.xml"
```

   `"C:\temp\TP_Connector_for_HiRDB_Type4.xml"` is the file path to which the acquired Connector attribute file (extension is `.xml`) is stored. You can change this file path as needed.

   Note that this step is not necessary if you have already created a Connector attribute file.

5. Edit the Connector attribute file to specify the database information to which you want to add the connection processing.

   Specify values for the tags listed below according to the settings for the database:

Tag	Explanation
`display-name`	Resource adapter name
`description`	Database port number
`DBHostName`	Database's IP address or host name
`encodeLang`	Character set corresponding to the database's character codes
`User`	Name of the user connecting to the database
`Password`	Password of the user connecting to the database

Observe the following rules when editing the *display-name* tab:

- Do not specify a character string beginning with `ucnp` (the specified value is not case sensitive).
- Do not specify a value that is already used in another resource adapter.

Note that this step is not necessary if you have already created a Connector attribute file.

6. Execute the command as follows to apply the contents of the edited Connector attribute file to the resource adapter you added:

```
Navigation-Platform-installation-directory\PP\uCPSB\CC\admin\bin
\cjsetrarprop uCNP_J2EE -resname resource-adapter-display-name -c "C:\temp
\TP_Connector_for_HiRDB_Type4.xml"
```

The resource adapter display name is either `DB_Connector_for_HiRDB_Type4` or the name specified for the *display-name* tag in the Connector attribute file.

7. Execute the command as follows to check whether the added resource adapter can connect to the database:

```
Navigation-Platform-installation-directory\PP\uCPSB\CC\admin\bin
\cjtestres uCNP_J2EE -type rar -resname resource-adapter-display-name
```

8. Execute the command as follows to start the resource adapter:

```
Navigation-Platform-installation-directory\PP\uCPSB\CC\admin\bin
\cjstartrar uCNP_J2EE -resname resource-adapter-display-name
```

4.4.2 Changing resource adapter settings

To change settings of an added resource adapter, you need to stop the plugin and resource adapter. The following describes the procedure:

1. Execute the command as follows to display a list of resource adapters, and then check the status of the resource adapter for which you want to change the settings:

```
Navigation-Platform-installation-directory\PP\uCPSB\CC\admin\bin
\cjlistrar uCNP_J2EE
```

A resource adapter is running if its display name is preceded by `running`. A resource adapter is stopped if its display name is preceded by `stopped`.

Note that the resource adapter whose display name is `uCNP_DB_Connector_for_HiRDB_Type4` is for use with Navigation Platform. Resource adapters having other display names have been added for plugins.

2. If any plugin is using the resource adapter for which you want to change the settings, stop that plugin by executing the command as follows:

```
Navigation-Platform-installation-directory\PP\uCPSB\CC\admin\bin
\cjstopapp uCNP_J2EE -name plugin-name
```

3. Execute the command as follows to stop the resource adapter for which you want to change the settings:

```
Navigation-Platform-installation-directory\PP\uCPSB\CC\admin\bin
\cjstoprar uCNP_J2EE -resname resource-adapter-display-name
```

4. Perform step 5 and subsequent steps in *4.4.1 Adding a resource adapter*.

4.4.3 Deleting resource adapters

To delete a resource adapter, you must first delete the plugin that is using the resource adapter. The following describes the procedure:

1. Perform steps 1 and 3 in *4.4.2 Changing resource adapter settings*.

2. If any plugin is using the resource adapter to be deleted, execute the command as follows to delete that plugin:

```
Navigation-Platform-installation-directory\PP\uCPSB\CC\admin\bin
\cjdeleteapp uCNP_J2EE -name plugin-name
```

3. Execute the command as follows to delete the resource adapter:

```
Navigation-Platform-installation-directory\PP\uCPSB\CC\admin\bin
\cjdeleteres uCNP_J2EE -type rar -resname resource-adapter-display-name
```

4.4.4 Setting a plugin for database connection processing

To set database connection processing in a plugin, you need to edit the two files (`web.xml` and `cosminexus.xml`) to add resource adapter information. The following shows the storage location and editing contents of each file.

Important note

To edit `web.xml` and `cosminexus.xml`, use the editor started by selecting **Run as administrator**.

(1) Editing web.xml

Add the resource definition to the `web.xml` file of the User Plugin. The `web.xml` file is stored in:

```
Navigation-Platform-for-Developers-installation-directory\pluginSDK\plugin
\plugin-ID\WEB-INF\web.xml
```

The following shows a coding example when the resource reference name is `jdbc/TP_Connector_for_HiRDB_Type4`:

```
<resource-ref>
  <res-ref-name>jdbc/TP_Connector_for_HiRDB_Type4</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

Add the above lines to immediately before `</web-app>` at the end of the `web.xml` file.

(2) Editing cosminexus.xml

Edit the `cosminexus.xml` file of the User Plugin.

The following shows the `cosminexus.xml` file is stored in:

```
Navigation-Platform-for-Developers-installation-directory\pluginSDK\plugin
\plugin-ID\dd\META-INF\cosminexus.xml
```

The following shows a coding example when the resource reference name is `jdbc/TP_Connector_for_HiRDB_Type4` and the resource adapter display name is `TP_Connector_for_HiRDB_Type4`:

```
<resource-ref>
  <res-ref-name>jdbc/TP_Connector_for_HiRDB_Type4</res-ref-name>
  <linked-to>TP_Connector_for_HiRDB_Type4</linked-to>
</resource-ref>
```

Add the above lines to immediately before `</war>` at the end of the `cosminexus.xml` file.

4.4.5 Implementing database connection processing

The following indicates the conditions for developing plugins to be connected to a database:

- Processing that acquires the data source is implemented by the plugin initialization processing (`PluginInitializer` class of the `IPluginInitializer` interface).
- Data source lookup processing is implemented by the `init` method of the `PluginInitializer` class.

If you are developing plugins to be connected to a database, the following implementations are recommended:

- If an attempt to acquire data sources fails, `UCNPPluginUserException` is thrown.
- Data source update processing is implemented by the `inputFromNode` method.
- Data source reference processing is implemented by the `outputToNode` method.
- For plugins linked with external systems, the update and reference timing is synchronized by using other plugins.

Reference note

If you ignore a failure of the data source lookup processing and then terminate the plugin start processing, the next lookup processing is not executed until you restart Navigation Platform.

Because the lookup processing fails by either of the following causes, `UCNPPluginUserException` must be thrown to prevent Navigation Platform from starting:

- The resource adapter is not running.
- The resource adapter settings are inconsistent with the plugin implementation.

(1) Example of implementing plugin initialization processing

```
  // Store the data source in the static member variable so that the data
source can be acquired from the inputFromNode method.
  private static DataSource mDs;

  @Override
  public void init() throws UCNPPluginUserException {
    try {
      // Data source lookup processing must be performed within the init
method.
      InitialContext ic = new InitialContext();
      DataSource ds = (DataSource) ic.lookup(
          "java:comp/env/jdbc/TP_Connector_for_HiRDB_Type4");
      mDs = ds;
    } catch (NamingException e) {
      // If lookup processing fails, UCNPPluginUserException must be thrown.
      UCNPPluginUserException ue = new UCNPPluginUserException(
          "Lookup processing failed.", e);
      throw ue;
    }
  }

  public static DataSource getDataSource() {
    return mDs;
  }
```

(2) Example of implementing the outputToNode method

```
  public Map<String, Object> outputToNode(HttpSession session, Map<String,
Object> param){
    Map<String, Object> map = new HashMap<String, Object>();

    DataSource ds = PluginInitializer.getDataSource(); // Acquire the
instance
    Connection con = null;
    PreparedStatement statement = null;
    String name = null;
    try {
      // Establish a connection
      con = ds.getConnection();

      // Execute SQL
      // The outputToNode method must perform Reference processing
      statement = con.prepareStatement("SELECT NAME FROM TP_TBL");
      statement.setString(1, uid);
      ResultSet set = statement.executeQuery();
      // Obtain results
      if (set.next()) {
        name = set.getString(1);
      }
      Map<String, String> rtnParm = new HashMap<String, String>();
      rtnParm.put("outputParam1", name);
      map.put("ucnp.next.params.map ",rtnParm);
    } catch (SQLException e) {
       // Store the error information in response and interrupt transition.
       map.put("ucnp.error.message","An error occurred during DB access.");
       map.put("ucnp.error.type","NG");
    } finally {
      if (statement != null) {
        try {
          statement.close();
        } catch (SQLException e) {
        }
      }
      if (con != null) {
        try {
          con.close();
        } catch (SQLException e) {
        }
      }
    }
    return name;
  }
```

4.5 Implementing processing to be performed during plugin initialization or termination

In the `PluginInitializer` class of the `IPluginInitializer` interface, implement the processing that must be performed when a plugin starts or stops. If there is no such processing, the `PluginInitializer` class created by the template plugin creation command can be used without changes.

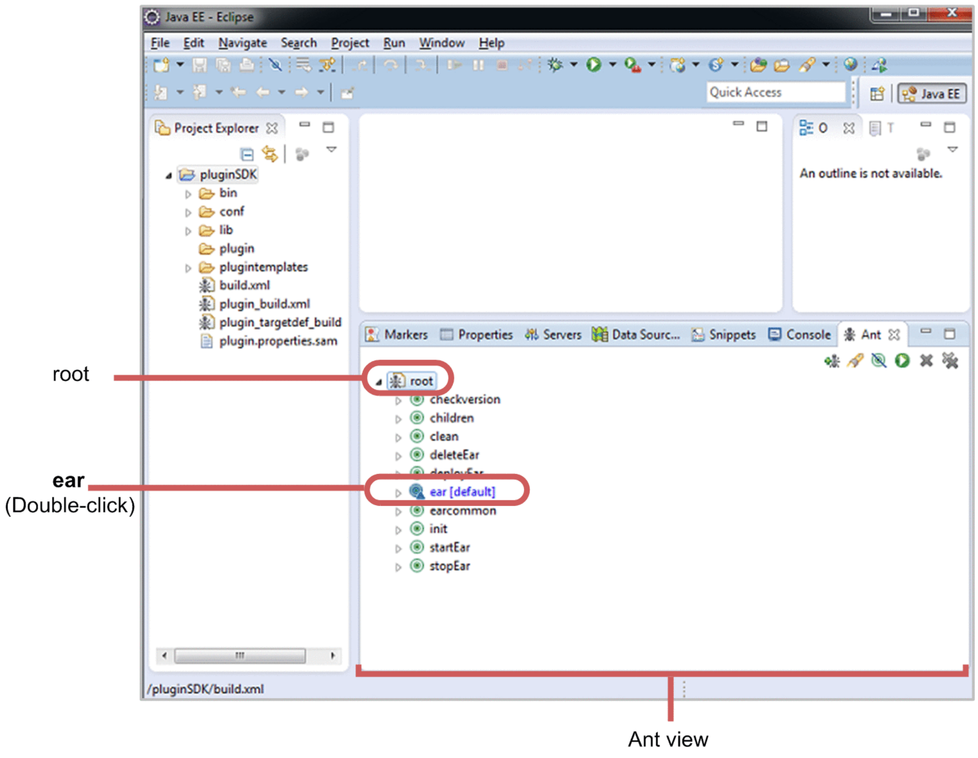
4.6 Building plugins

The section describes how to build plugins and how to take actions if an attempt to build a plugin fails.

4.6.1 Procedure for building plugins

You can build customized template plugins by executing `build.xml` of the pluginSDK project from Ant. To build plugins:

1. Start Eclipse.
2. Select **File**, **Import**, **General**, and then **Existing Projects into Workspace**.
3. In the **Select root directory** text box, specify the following value, and then click the **Finish** button:
   Navigation-Platform-for-Developers-installation-directory`\pluginSDK`
   Do not select the **Copy projects into workspace** check box.
4. Select **Window**, **Show View**, **Other**, **Ant**, and then **Ant**. The Ant view appears.
5. From the project imported in *3.2 Importing a pluginSDK project*, drag `build.xml` into the Ant view.
   `root` is added in the Ant view.
6. In the Ant view, click the + icon for `root` to display the target list.



7. In the target list, double-click **ear**.
   All plugins under the *Navigation-Platform-for-Developers-installation-directory*`\pluginSDK\plugins` directory are built, and then the EAR file is created in the following directory:

```
Navigation-Platform-for-Developers-installation-directory\pluginSDK\dest
```

When you build plugins after the first time, if necessary, double-click **clean** in the target list before double-clicking **ear** in the target list. This deletes intermediate files for the class, library, and plugins.

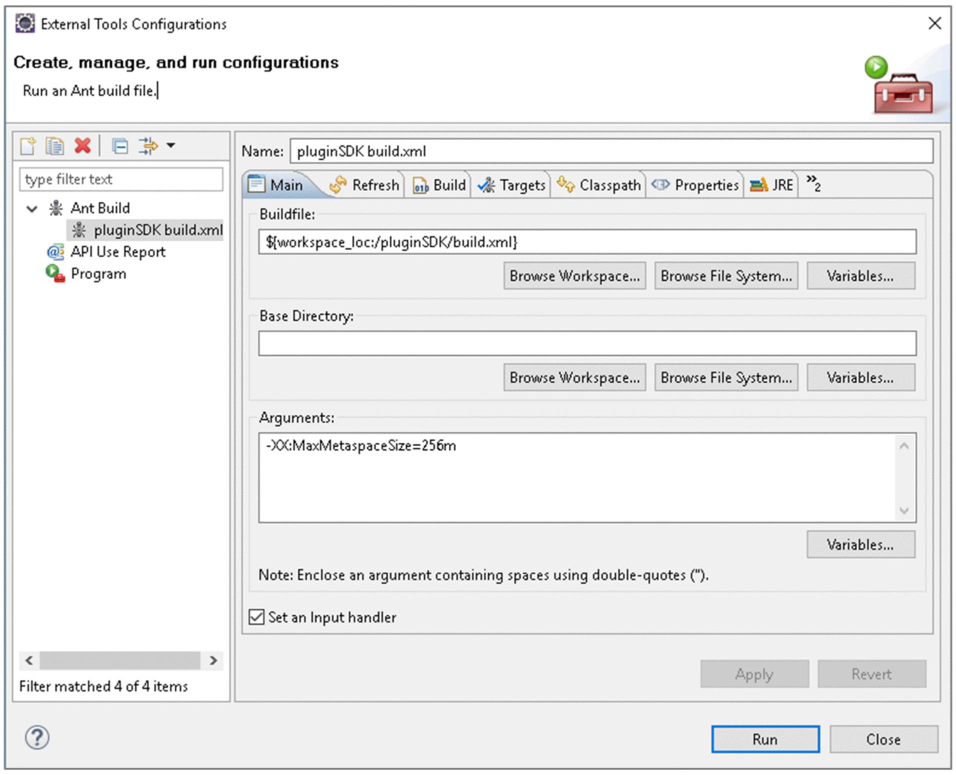
4.6.2 Actions to be taken if an attempt to build plugins fails

The following describes how to take action if an attempt to build plugins fails.

The probable cause of a failure in building plugins is an error of `java.lang.OutOfMemoryError`. This error is likely to occur if there are multiple plugins.

If an error (`java.lang.OutOfMemoryError`) occurs, perform the following, and then build the plugins again:

1. Right-click `root` in the Ant view. In the menu that opens, select **Run As**, and then **External Tools Configurations**
   The External Tools Configurations dialog box opens.
2. Click the **JRE** tab, and then enter `-XX:MaxMetaspaceSize=256m` in the **VM arguments** field.



External Tools Configurations

Create, manage, and run configurations

Run an Ant build file.

Name: pluginSDK build.xml

Buildfile: ${workspace_loc:/pluginSDK/build.xml}

Arguments: -XX:MaxMetaspaceSize=256m

Note: Enclose an argument containing spaces using double-quotes (").

☑ Set an Input handler

Filter matched 4 of 4 items

3. Click the **Apply** button, and then click the **Close** button.
   The External Tools Configurations dialog box is closed.

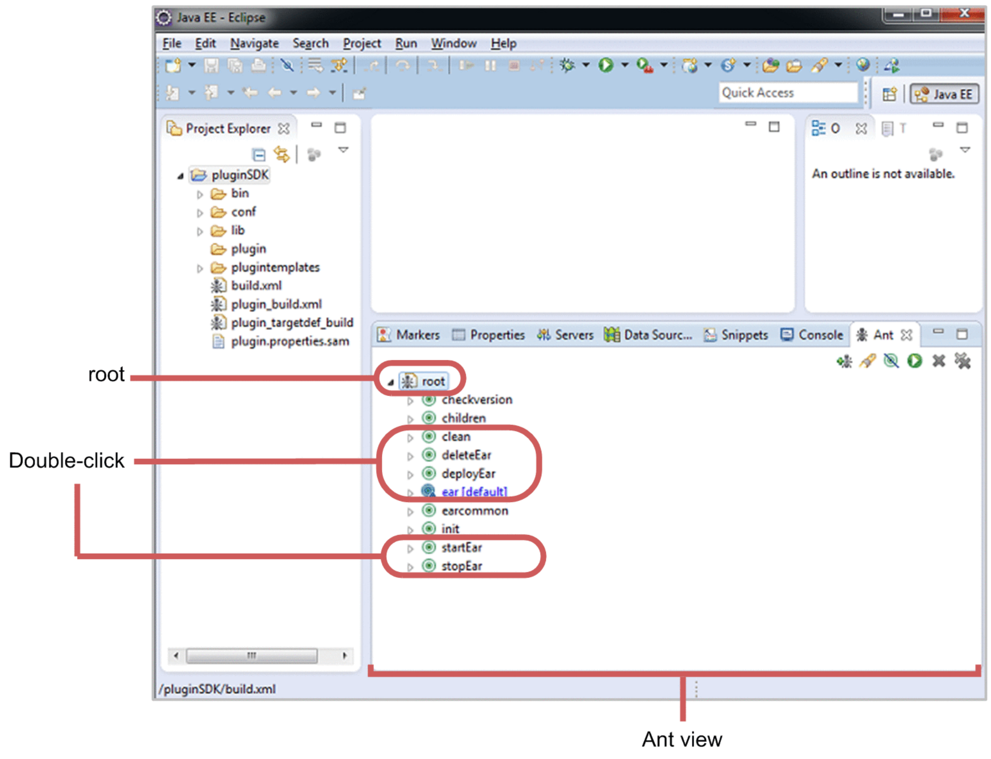
4.7 Deploying plugins

Before you can use developed plugins, you must deploy and start them by using Eclipse. The following describes the procedure.

Tip

You must perform this procedure as a user having Windows administrator roles. Note that the procedure assumes that the `root` node of the pluginSDK project has been added to the Eclipse Ant view.

1. Start Navigation Platform (J2EE server).
   This step is not necessary if Navigation Platform is already running. For details about how to start Navigation Platform, see the *Hitachi Navigation Platform Setup and Operations Guide*.
2. Select **Run as administrator**, and then start Eclipse.
3. Select **Window**, **Show View**, **Other**, **Ant**, and then **Ant**. The Ant view appears.
4. From the project imported in *3.2 Importing a pluginSDK project*, drag `build.xml` into the Ant view.
   `root` is added in the Ant view.
5. In the Ant view, click the + icon for `root` to display the target list.



6. If any plugin is running on the J2EE server, double-click **stopEar** in the target list.
   The J2EE application for the plugin stops.
7. If a plugin exists on the J2EE server, double-click **deleteEar** in the target list.
   The J2EE application for the plugin is deleted.
8. In the target list, double-click **clean**, and then double-click **ear**.

Plugins are built.

9. In the target list, double-click **deployEar**.
   The J2EE applications for plugins are imported.
10. In the target list, double-click **startEar**.
   The J2EE applications for plugins are started.
11. Restart Navigation Platform (J2EE server).

4.8 Associating I/O Plugins with Operational Content

In the Operational Content Editing Window, associate I/O Plugins with Operational Content that uses the plugins. You do not perform this for Suspend/Resume Plugins and Custom Window Plugins. A port (●) is displayed with a Guide Part that can be associated with an I/O Plugin.

This section describes how to associate I/O Plugins with Guide Parts by using mapping lines in the Operational Content Editing Window.

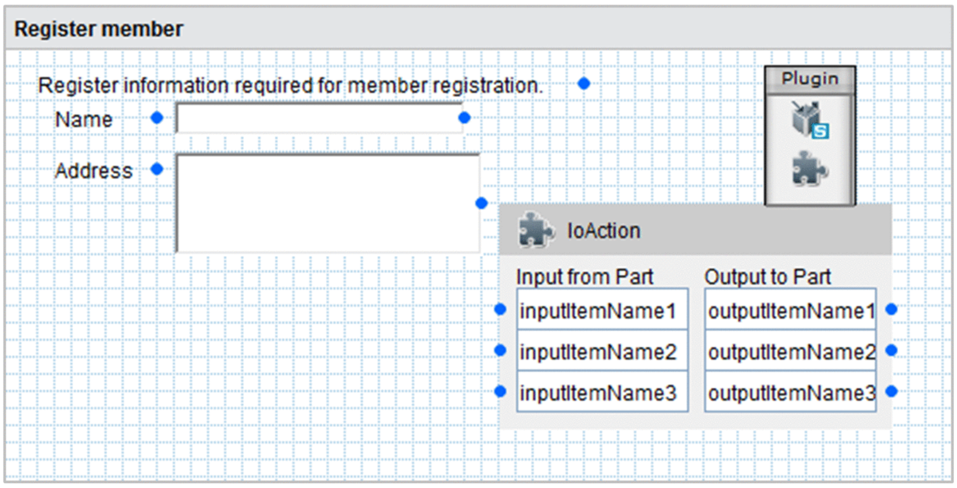
4.8.1 Drawing mapping lines (connecting Guide Parts and Plugin Parts)

The following describes the procedure for placing I/O Plugin parts and then drawing mapping lines.

Tip

You can view and work with Plugin Parts and mapping lines only when the Guide area is in mapping mode. For details about each step, see the *Hitachi Navigation Platform Content Editing Guide*.

1. Specify the following URL on the Web browser to invoke the login window:
   `http://`*host-name*`:`*port-number*`/ucnpBase/portal/screen/TitlePortlet/portlet/ucnp/pane/-44Ob44O844Og55S76Z2i/layout_id/default/tab_id/-44Ob44O844Og55S76Z2i?start_editor=true&open_editor=true`
   - *host-name*
     Host name or IP address of the J2EE server machine in a development environment.
   - *port-number*
     Web server port number in a development environment. This value will be specified for `ucnp.setup.server.cosminexus.hws.http.port property` in the user setup property file (`ucnp_setup_user.properties`).
2. Enter the user ID and password, and then click the **Login** button.
   If you enter incorrect information, click the **Reset** button to clear the entered information, and then enter information again.
   When you click the **Login** button, the Operational Content Editing Window appears.
3. In the Operational Flow area, click the Process Node in which you want to set a plugin.
   The Guide corresponding to the Operational Flow appears in the Guide area.
4. Click Plugin (**Plugin** button) in the toolbar.
   The Guide area switches to mapping mode.
5. In the **Plugins** palette in the Guide area, click (**Plugin** button).
6. Click a location in the Guide area.
   A Plugin Part is placed.



If you want to place other plugins, repeat steps 5 and 6.

7. Draw a mapping line by clicking and dragging from the port (●) of the transition-source Guide Part to an input parameter of the Plugin Part (the port on the left of the Plugin Part).

   The cache value of the Guide Part will be passed to the plugin as an input parameter.

   For details about cache values of each Guide Part, see *5.2.1 (5) Cache values of Guide Parts*.

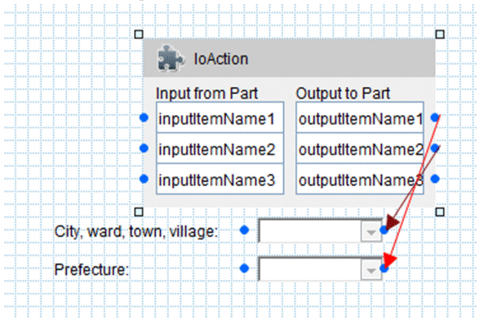
8. Draw a mapping line by clicking and dragging from an output parameter of the Plugin Part (the port to the right of the Plugin Part) to the transition-destination Guide Part.

   The value set in the plugin will be passed to the Guide Part as an output parameter.

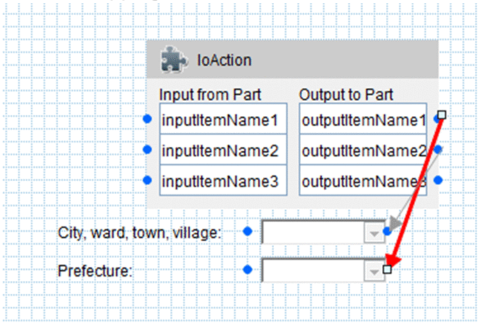
Supplementary note:

- Mapping lines appear as colored arrows as shown in the figure below. Because each mapping line has a different color, you can easily identify lines even if they cross over.

■ When a Plugin Part is selected



■ When mapping lines are selected



- You can draw mapping lines to multiple input parameters from one Guide Part.
- You cannot draw mapping lines from multiple output parameters to one Guide Part.
- It is not mandatory that you draw mapping lines to the ports of a Plugin Part.
- If you have placed a Plugin Part, plugin processing is invoked when a transition occurs between nodes containing the plugin in the Operational Content Editing Window, even if no mapping lines are drawn to the Plugin Part.
- Mapping lines or Plugin Parts set in this task are not displayed in the Operational Content Execution Window.

Notes on cache values of Guide Parts

The cache values of Guide Parts are shared between Guide Parts of the same type. When the Operational Flow includes a Branching node, if a Process Node that has already been displayed is displayed again via another route, the cache value used when the Process Node was first displayed is used.

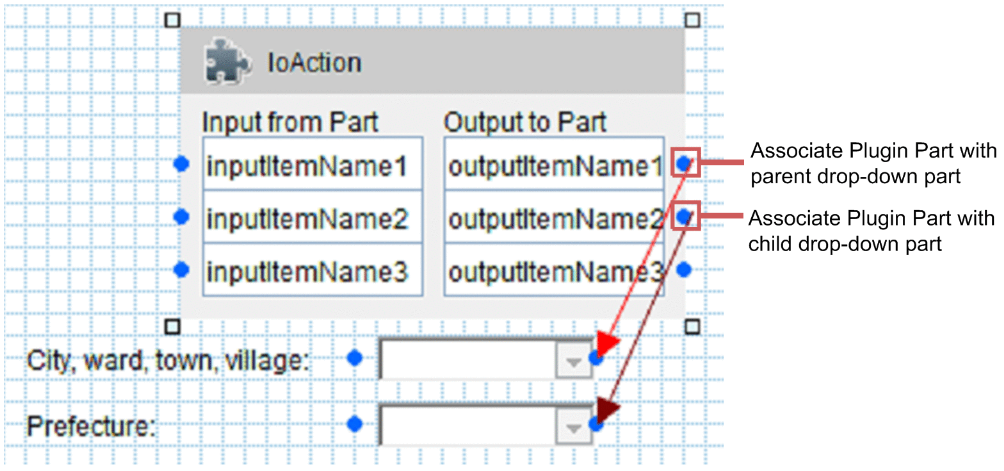
You can use I/O Plugins (server processing implementation interface) to change the cache value according to certain conditions. For details about the server processing implementation interface, see *5.2 IIoPluginController (server processing implementation interface)*.

Notes on associating Plugin Parts with drop-down parts in a parent-child relationship

To associate Plugin Parts with drop-down parts that are configured in a parent-child relationship, define the Operational Content so that the parameter linked to the parent drop-down part is above the parameter linked to the child drop-down part. The parameters of parent and child drop-down parts need not be contiguous.

The following figure shows an example in which output parameters (the ports on the right side of a Plugin Part) are associated with drop-down parts in a parent-child relationship.

Figure 4–4: Example of associating output parameters with drop-down parts in parent-child relationship



In this example, `outputItemName1` is associated with the parent drop-down part, and `outputItemName2` is associated with the child drop-down part. Because the parameters associated with drop-down parts in parent-child relationships need not be contiguous, for example, `outputItemName3` could also be associated with the child drop-down part.

These notes do not only apply to output parameters. They also apply when you associate input parameters with parent-child drop-down parts.

Operations after editing plugins

If you edit a plugin after associating its Plugin Part with parameters, you need to place the Plugin Part for the edited plugin again and redraw the mapping lines.

For details about how to enable edited plugins without placing them and redrawing mapping lines, see the description about updating User Plugins in the *Hitachi Navigation Platform Setup and Operations Guide*.

4.8.2 Details about values input to or output from plugins

The following table describes the values input to or output from plugins for each type of Guide Part.

Table 4–5: Input values from Guide Parts to plugins and output results from plugins to Guide Parts

Guide Part type	Value input to the plugin	Result output to the Guide Part
Static text	The character string displayed as static text	The character string is output as the static text.
Text box	The character string entered in the text box	The character string is output to the text box.
Text area	The character string entered in the text area	The character string is output to the text area.

Guide Part type	Value input to the plugin	Result output to the Guide Part
Radio button	The value of the selected radio button	The corresponding radio button becomes selected.

Check box	One of the following values depending on whether the check box is selected:
	If the check box is selected
	`true`
	If the check box is not selected
	`false`
		The check box becomes selected if the value is `true`.
		For any other value, the check box remains cleared.

Drop down	The value of the selected list	The value corresponding to the selected list entry is output.
Hyperlink	The anchor text and URL (specified in the Attribute Settings window) connected by a linefeed code (`\r\n`)	The value#1 consisting of the anchor text and URL connected by a linefeed code (`\r\n`) is output.
Image	The value#2 consisting of the image part URL and description (the value in the **Tool tip** text box) specified in the Attribute Settings window and connected by a linefeed code (`\r\n`)	The value#3 consisting of the URL set for the image part, a linefeed code (`\r\n`), and the image description (the value in the **Tool tip** text box) is output.
Inline frame	The URL used to display the inline frame	The URL used to display the inline frame is output.
HTML Part	Of the information specified in the **HTML source string** area for the HTML Part, a character string encoded after the `name` and `value` attributes of the following are acquired: • Text box • Password box • **hidden** field • Check box • Radio button • Selection box • Text area	The value set in the **HTML source string** area for the HTML Part is output.

#1

Note the following regarding the output value:

- An empty character string is output if the value is null, empty character string, or consists only of a linefeed code.
- If the value does not contain a linefeed code, the entire string is processed as anchor text (an empty character string is output for the URL).
- If the value contains two or more linefeed codes, the values on the third and subsequent lines are ignored.

#2

If the **Tool tip** text box was left blank (empty character string), the input value will be the URL followed by a linefeed code (`\r\n`). Take care not to include the linefeed code if your intention is to extract only the URL.

#3

Note the following regarding the output value:

- An empty character string is output if the value is null.
- If the value does not contain a linefeed code (`\r\n`), the entire string is processed as a URL. An empty character string is output for the tool tip.
- If the value contains two or more linefeed codes, the values on the third and subsequent lines are ignored.

4.8.3 Updating Plugin Parts

If you change any of the following items by editing the I/O plugin XML file, you need to update the Plugin Part to enable the changes:

- File name of a tool icon
- Execution order
- Execution confirmation dialog box
- Preview window execution flag
- Execution suppression by the button type
- Description character string
- Input parameter
- Output parameter

If the plugin before change is already associated with Operational Content, you need to delete the placed plugin and then perform the association process again. For details about how to enable changes without performing the association process again, see the description about updating User Plugins in the *Hitachi Navigation Platform Setup and Operations Guide*.

Note that the task for associating plugins again or updating User Plugins must also be performed in an editing environment and execution environment. Therefore, if you edit the I/O plugin XML file, notify the system administrator.

4.8.4 Checking configuration information for Operational Content that uses plugins

You can check configuration information for Operational Content that uses plugins. To do this, click the **Mapping List** button in the toolbar in the Operational Content Editing Window to output a mapping list file. For details about the output procedure and output contents, see the *Hitachi Navigation Platform Content Editing Guide*.

4.9 Configuring the user property file

To enable developed plugins so that they operate normally, you need to configure the user property file (`ucnp_user.properties`) as follows:

- Do not specify `all` for the `ucnp.base.client.directjump.enable` property.

  If you specify `all`, an error occurs in a Suspend/Resume Plugin, and Navigation Platform does not start. Similarly, I/O Plugins do not operate.
- Specify `true` for the `ucnp.base.client.suspend.enable` property only if a Suspend/Resume Plugin exists.

  If you specify `true`, Suspend/Resume Plugins operate. If you specify `true` when there is not a Suspend/Resume Plugin, an error occurs after the login, disabling the Operational Content Execution Window.
- Change the values of the `ucnp.base.server.logoutbutton.display` property and `ucnp.base.server.close.button.setting` property from the defaults according to custom window processing.

  To log out from a custom window, specify `false` for the `ucnp.base.server.logoutbutton.display` property and `close_only` for the `ucnp.base.server.close.button.setting` property, and then make sure that the **Close** button is displayed in the Operational Content Execution Window of Navigation Platform.

If necessary, change the values of the following properties from the defaults:

- `ucnp.base.client.viewer.confirm.discardinput.enable` property (whether to display a confirmation dialog box when input information is discarded)
- `ucnp.base.client.suspend.confirm.load.enable` property (whether to display a confirmation dialog box when operation is resumed)

For the storage location of the user property file and details about properties, see the *Hitachi Navigation Platform Setup and Operations Guide*.

4.10 Debugging plugins

To debug plugins, use Eclipse.

4.10.1 Conditions for debugging plugins

To debug plugins, you need to set up a development environment by specifying `true` for the `ucnp.setup.server.cosminexus.debug.enable` property in the user setup property file (`ucnp_setup_user.properties`). You also need to start Navigation Platform when you start debugging.

For details about the setup procedure, property details, and how to start Navigation Platform, see the *Hitachi Navigation Platform Setup and Operations Guide*.

4.10.2 How to debug plugins

To debug a plugin by using the Eclipse debugger function:

1. Start Eclipse.
2. In the Package Explorer View, click the template plugin project imported in *4.2 Importing a template plugin project*.
   The project is selected.
3. From the menu, select **Debug**, and then **Debug Configurations**.
   The Debug Configurations dialog box appears.
4. Right-click **Remote Java Application** and, in the menu that opens, click **New**.
   A new remote Java application is created. For the port number, specify the value set for the `ucnp.setup.server.cosminexus.debug.jdwp.port` property in the user setup property file (`ucnp_setup_user.properties`).
5. Click the **Debug** button.
   The debugger runs.
6. Set breakpoints in the implemented Java code.
7. In the Web browser, perform the operation below for debugging the plugin.
   For I/O Plugins
      Perform operation on Operational Content associated with the plugin and make sure that the plugin operates correctly.
   For Suspend/Resume Plugins
      Partially perform operation on any Operational Content, and then log out by clicking the **Suspend** button. Then, log in again and make sure that the contents of operation up to the suspended point were saved and you can resume the operation from the middle of the Operational Flow.
   For Custom Window Plugins
      After you log in, make sure that the custom window appears.

4.11 Deleting plugins

To delete plugins, use Eclipse. Before deleting plugins, you need to check the IDs of the plugins and whether the plugins use a library.

4.11.1 How to delete plugins

To delete plugins:

1. Start Navigation Platform (J2EE server).
   This step is not necessary if Navigation Platform is already running. For details about how to start Navigation Platform, see the *Hitachi Navigation Platform Setup and Operations Guide*.
2. To delete I/O Plugins, delete Operational Content that uses the plugins, or delete the association with those plugins from Operational Content.
3. Select **Run as administrator**, and then start Eclipse.
4. Select **Window**, **Show View**, **Other**, **Ant**, and then **Ant**. The Ant view appears.
5. In the target list, double-click **stopEar** and then **deleteEar** to delete all plugins.
6. Delete the directory of the plugins to be deleted (*Navigation-Platform-for-Developers-installation-directory* `\pluginSDK\plugin\`*plugin-ID*).
7. If the plugins to be deleted only use the library placed on the layer of the System Class Loader, delete both the library and class path settings.
8. If the plugins deleted in step 5 contain a plugin you do not want to delete, double-click **deployEar** and then **startEar** to deploy it again.
9. Restart Navigation Platform.

4.12 Calculating memory usage for plugins

The expression for obtaining the memory usage for a User Plugin is as follows:

```
memory-usage-A# - memory-usage-B = memory-usage-for-User-Plugin
```

Legend:

memory-usage-A: For executing Operational Content created by using a User Plugin

memory-usage-B: For executing Operational Content created without using a User Plugin

#

When using the `ucnpOptions` parameter, measure the memory usage by specifying the `ucnpOptions` parameter before executing Operational Content.

4.12.1 Procedure for creating Operational Content for measurement

To calculate the memory usage for User Plugins, you must execute the same Operational Content by using a User Plugin and without using a User Plugin, and then measure the memory usage for each case.

To create Operational Content that does not use User Plugins:

1. Log in to Navigation Platform.
2. Copy the Operational Content created by using a User Plugin.
3. Delete the User Plugin from the copy of the Operational Content.
4. Save the Operational Content.

4.12.2 Procedure for measuring the memory usage for plugins

The following describes the procedure for measuring the memory usage.

(1) Restarting Navigation Platform

1. Use the `npstop` command (stop) to stop Navigation Platform.
2. Use the `npstart` command (start) to start Navigation Platform.

(2) Executing Operational Content

1. Log in to Navigation Platform.
2. Select Operational Content for which you want to measure the memory usage.
3. Click the **Next** button repeatedly to execute Operational Content to the end.
   If Operational Content contains multiple routes with branching, use the route that passes the node in which the plugin is placed.

Note after execution is completed

After execution of Operational Content, do not display another Operational Content or log out.

(3) Performing a GC (garbage collection)

1. Execute the following command to acquire the process ID of the process that is using the RMI registry port of Navigation Platform:

```
netstat -abo
```

The following shows a command output example:

```
Protocol  Local address  External address    Status           PID
 TCP    0.0.0.0:24702       navipla:0     LISTENING      1876
```

Check the PID on the line on which the local address port number is `24702`. In this example, `1876` is the process ID of the process using the RMI registry port. If the RMI registry port is unknown, check the value specified for the `ucnp.setup.server.cosminexus.ejbserver.rmi.naming.port` property in the user setup property file (`ucnp_setup_user.properties`). The default is `24702`.

2. Use the following command to perform a GC:

```
"Navigation-Platform-installation-directory\PP\uCPSB\jdk\jre\bin
\javagc.exe" -p process-ID-of-the-process-using-the-RMI-registry-port
```

When you execute the command, the message below appears. Enter `y`, and then press the **Enter** key.

```
Force VM to execute GC: ? (y/n)
```

(4) Stopping J2EE applications for Navigation Platform and plugin

1. Execute the following command to stop the J2EE application for the plugin:

```
Navigation-Platform-installation-directory\PP\uCPSB\CC\admin\bin
\cjstopapp uCNP_J2EE -name User-Plugin-name
```

If there are multiple User Plugins, repeatedly execute this command until all plugins stop.

2. Execute the following command to stop the J2EE application for Navigation Platform:

```
Navigation-Platform-installation-directory\PP\uCPSB\CC\admin\bin
\cjstopapp uCNP_J2EE -name ucnp
```

(5) Checking operating statistics

Check the JavaVM operating statistics file output to the following path:

```
Navigation-Platform-installation-directory\PP\uCPSB\CC\server\public\ejb
\Navigation Platform-J2EE-server-name\stats\HJVMStats_YYYYMMDDhhmmTZ.csv
```

Legend:

YYYYMMDDhhmm: Date and time that the operating statistics file is created

TZ: Time zone

If you open the operating statistics file in Excel, the memory usage when Operational Content was executed is output to cell AK. This value shows the maximum memory usage (unit: bytes) per session released from the Explicit heap in the last 60 seconds after the J2EE application for Navigation Platform stopped.

(6) Restarting Navigation Platform

1. Use the `npstop` command (stop) to stop Navigation Platform.

2. Use the `npstart` command (start) to start Navigation Platform.

4.13 Changing J2EE server settings

This section describes J2EE server setting items that can be changed as required, and how to change them. For items not described in this manual, do not change the settings from the default.

4.13.1 J2EE server setting items that can be changed during plugin development

The following shows the J2EE server files that can be edited and the setting items that can be changed.

User property file for J2EE servers (`usrconf.properties`)

Java VM system property

Option definition file for J2EE servers (`usrconf.cfg`)

System class path

Minimum value for the Java heap memory usage for Java VM

Maximum value for the Java heap memory usage for Java VM

Maximum size of the Explicit memory block for the Explicit Memory Management functionality of Java VM

In addition, you can also change the resource adapter according to the plugin you develop. For details about changing the resource adapter, see *4.4 Adding database connection processing*.

4.13.2 Storage location of the files used for changing J2EE server setting items

The storage location of the user property file for J2EE servers (`usrconf.properties`) and option definition file for J2EE servers (`usrconf.cfg`) is as follows:

```
Navigation-Platform-installation-directory\PP\uCPSB\CC\server\usrconf\ejb
\uCNP_J2EE
```

4.13.3 Procedure for changing the user property file for J2EE servers (usrconf.properties)

The setting item that can be changed in the user property file for J2EE servers (`usrconf.properties`) is the Java VM system property. The following describes the procedure.

Important note

Do not edit anything not described in this procedure. Do not add any value not described here.

1. Start the editor as an administrator and then open the user property file for J2EE servers (`usrconf.properties`).
2. Add the value specified for the Java VM system property to a line below the following comment lines:

```
##########################################################################
## When you edit, please add description below.                         ##
##########################################################################
```

3. Execute the `npstop` command (stop) and `npstart` command (start) to restart Navigation Platform.

(1) Details about the Java VM system property

Format

Specify a key as follows:

```
key-name = value
```

Specification method

- Each value ends at the linefeed.
- A line that begins with a hash mark (`#`) is assumed to be a comment.
- A line that does not contain any value is ignored.
- Character strings such as empty characters and comments cannot be added following the value. If such a string is added, the value is interpreted to be invalid.

You can specify any character string for a system property key name. However, do not use any of the following prefixes:

- `ucnp.`
- `java.`
- `javax.`
- `javax.portlet.`
- `hptl`
- `com.cosminexus`
- `jp.co.hitachi.soft.portal`
- The following keys that are reserved for the J2EE server:
  - `com.cosminexus`
  - `cosminexus.jpa`
  - `ejbserver.application`
  - `ejbserver.client`
  - `ejbserver.commonj`
  - `ejbserver.compiler`
  - `ejbserver.connectionpool`
  - `ejbserver.connector`
  - `ejbserver.container`
  - `ejbserver.ctm`
  - `ejbserver.deploy`
  - `ejbserver.distributedtx`

- `ejbserver.DynamicStubLoading`
- `ejbserver.ejb`
- `ejbserver.ext`
- `ejbserver.http`
- `ejbserver.instrumentation`
- `ejbserver.jca`
- `ejbserver.jndi`
- `ejbserver.jpa`
- `ejbserver.jta`
- `ejbserver.logger`
- `ejbserver.management`
- `ejbserver.manager`
- `ejbserver.naming`
- `ejbserver.rmi`
- `ejbserver.security`
- `ejbserver.server`
- `ejbserver.stateful`
- `ejbserver.stdoutlog`
- `ejbserver.watch`
- `ejbserver.webj2ee`
- `https`
- `java`
- `javax`
- `vbj`
- `vbroker`
- `webserver.application`
- `webserver.connector`
- `webserver.container`
- `webserver.context`
- `webserver.dbsfo`
- `webserver.eadssfo`
- `webserver.errorpage`
- `webserver.http`
- `webserver.jsp`
- `webserver.logger`
- `webserver.servlet`

- `webserver.session`
- `webserver.static`
- `webserver.work`
- `webserver.xml`

(2) Example of editing the user property file for J2EE servers (usrconf.properties)

The following shows an example of adding a key named `com.example.property.key1` in the user property file for J2EE servers:

```
##############################################################################
## When you edit, please add description below.                            ##
##############################################################################
com.example.property.key1=value1
```

4.13.4 Procedure for changing the option definition file for J2EE servers (usrconf.cfg)

You can change the following items in the option definition file for J2EE servers (`usrconf.cfg`):

- System class path
- Java heap memory usage for Java VM
- Explicit memory block size for the Explicit Memory Management functionality of Java VM

The following describes the procedure.

Important note

Do not edit anything not described in this procedure. Do not add any value not described here.

1. Start the editor as an administrator and then open the option definition file for J2EE servers (`usrconf.cfg`).
2. To change the settings of the Java VM Java heap memory usage and the Explicit memory block size for the Java VM Explicit Memory Management functionality, change the key values on the lines below the following comment lines:

```
##########################################################################
## When you change settings, please correct the following description.  ##
##########################################################################
```

3. To change the system class path setting, add the specification to a line below the following comment lines:

```
##########################################################################
## When you edit, please add description below.                        ##
##########################################################################
```

4. Execute the `npstop` command (stop) and `npstart` command (start) to restart Navigation Platform.

(1) System class path

Specify a user-created program, such as a User Plugin or user library, to be loaded by the System Class Loader on the J2EE server.

Specify a program that can be referenced from the entire J2EE server. Multiple programs can also be specified.

Key name

`add.class.path`

Specified value

Specify a system class path. To specify multiple paths, delimit them with a semicolon (`;`).

Default value

None

(2) Details about the Java heap memory usage for Java VM

For the Java heap memory usage for Java VM on the J2EE server, you must specify a value greater than the memory usage for Navigation Platform. Specify the same value for the minimum value and maximum value.

Important note

You cannot omit the maximum and minimum values for the Java heap memory usage for Java VM on the J2EE server. Do not delete the `add.jvm.arg` key entered by default.

Key name

`add.jvm.arg`

Specified value

`-Xmx`*memory-usage-size*`m`

Specify the memory usage size in the range from 1024 to 1434. If you specify a value greater than 1434, the memory size will exceed the upper-limit for the OS memory size. Note that `m` at the end of the specified value indicates MB (megabytes).

Default value

`-Xmx1024m`

(3) Details about the Explicit memory block size for the Explicit Memory Management functionality of Java VM

For the maximum size of the Explicit heap for Java VM on the J2EE server, you must specify a value greater than the Explicit memory block size for Navigation Platform.

Important note

You cannot omit the maximum size of the Explicit heap. Do not delete the `add.jvm.arg` key entered by default.

Reference note

User Plugin instances and session information containing the value of the `ucnpOptions` parameter are stored in the Explicit heap. Therefore, we recommend that you assume a case in which the Explicit heap must be expanded for the *memory size required for User Plugin instances per user* **x** *maximum number of concurrently executing users*.

Key name

`add.jvm.arg`

Specified value

`-XX:HitachiExplicitHeapMaxSize=`*maximum-size-of-Explicit-heap*`m`

Specify a value equal to or greater than 160 for the maximum size of the Explicit heap. Note that `m` at the end of the specified value indicates MB (megabytes).

Default value

`-XX:HitachiExplicitHeapMaxSize=160m`

4.14 Notes on developing plugins used with Operational Content for iPad

- To execute a process that uses JavaScript to close a window created with the Custom Window Plugin, use the following code:

```
var lWindow = window.open('','_self');
lWindow.opener=window;
lWindow.close();
```

- If the window in which the window-closing processing is performed consists of multiple frames by using iframes or similar, implement the processing on the top-level frame.
- If execution of JavaScript takes 10 or more seconds, Mobile Safari processing might stop. Develop the plugin so that plugin processing is completed within 10 seconds, including the communication time.

4.15 About use of plugins developed in old versions

I/O Plugins developed in a development environment for a version earlier than uCosminexus Navigation Developer 09-50 (referred to as a *version earlier than 09-50*) do not operate in Navigation Platform. If you want to use I/O Plugins developed in a version earlier than 09-50, you must develop them again, starting with creation of a template plugin.

Similarly, I/O Plugins developed in a development environment for a version of uCosminexus Navigation Developer 09-50 or uCosminexus Navigation Developer 09-60 (referred to as *version 09-50 or later*) do not operate in Navigation Platform. However, you do not have to re-develop I/O Plugins developed in version 09-50 or later because they can be migrated to new template plugins of Navigation Platform.

For details about the migration procedure, see *C.1 Procedure for migrating plugins developed in old versions*.

Tip

Operational Content associated with I/O Plugins developed in a version earlier than 09-50 cannot be imported to Navigation Platform. To import such Operational Content, you must first delete all relevant plugins in an environment that is a version earlier than 09-50.

4.16 Setting up access permissions to Java packages to be used by User Plugins

For some Java packages to be used by User Plugins, you need to set up access permissions individually.

1. Start the editor by selecting **Run as administrator**, and open the following file:

```
Navigation-Platform-installation-directory\PP\uCPSB\CC\server\usrconf\ejb
\uCNP_J2EE\server.policy
```

2. Add accessClassInPackage settings to the following locations:

```
grant codeBase "file:${ejbserver.http.root}/ejb/${ejbserver.serverName}/
apps/-" {
  permission java.io.FilePermission "<<ALL FILES>>", "read, write,
delete, execute";
  permission java.lang.RuntimePermission "accessDeclaredMembers";
  permission java.lang.RuntimePermission "getProtectionDomain";
  permission java.lang.RuntimePermission "setFactory";
  permission java.lang.reflect.ReflectPermission "suppressAccessChecks";
  permission java.lang.RuntimePermission "getenv.*";
  permission java.lang.RuntimePermission "getClassLoader";
  permission java.lang.RuntimePermission "accessClassInPackage.Java-
package-name-to-be-used"; // Add
};
```

3. Restart Navigation Platform.

5 API Reference (for I/O Plugin Development)

This chapter describes the APIs used for developing I/O Plugins.

5.1 List of APIs (for I/O Plugin development)

The following describes the APIs used for developing I/O Plugins.

Table 5–1: List of APIs (I/O Plugin development)

Category	Interface name or class name	Description
Common to all plugins	`IUCNPSession`	An interface for using session information
	`IPluginInitializer`	An interface used to implement the processing for starting (initializing) and terminating User Plugins
	`UCNPPluginUserException`	User Plugin exception class
I/O Plugin	`IIoPluginController`	An interface used to implement the server processing for I/O Plugins
	`ParamConvertUtil`	A utility class that converts I/O parameters to the Map format
	`UCNPPluginException`	A class that indicates exceptions that occurred in I/O Plugins

5.2 IIoPluginController (server processing implementation interface)

This interface is used to implement the server processing for I/O Plugins.

Package

```
package jp.co.hitachi.soft.ucnp.plugin.inputoutput.controller;
```

Format

```
public interface IIoPluginController
```

Methods

- `inputFromNode` method

  When transition occurs from a node that contains an I/O Plugin to another node, this method performs processing to acquire and input information about the transition source node.

- `outputToNode` method

  When transition to a node that contains an I/O Plugin occurs, this method performs processing to output information to the transition destination node.

5.2.1 inputFromNode method

When transition occurs from a node that contains an I/O Plugin to another node, this method performs processing to acquire and input information about the transition source node.

You can use this method to effectively implement data update processing, such as acquiring information about the transition source node and linking with the external system, during node transition.

Notes on the inputFromNode method

- If multiple I/O Plugins are placed in the same node, the `inputFromNode` methods of all I/O Plugins are executed irrespective of the results of a previously executed `inputFromNode` method. Therefore, if you place multiple plugins whose input parameters must be checked, you need to check the input parameters for the `inputFromNode` methods of all I/O Plugins.
- The `inputFromNode` method is executed more than once in the following cases:
  - The result of an `inputFromNode` method that is executed later indicates an error.
  - The transition destination node contains an I/O Plugin and the result of the `outputToNode` method indicates an error.

  Therefore, when you implement the processing, maintain the integrity of data to be updated so that no problem occurs if the `inputFromNode` method is executed multiple times. Note that a method's execution results, other than error, are normal and warning.

(1) Format

```
public Map<String, Object> inputFromNode(HttpSession session, Map<String,
Object> param);
```

(2) Arguments

(a) session

This argument stores the current session. To execute the `setAttribute()` method for the `HttpSession` object acquired in a plugin, do not specify any of the following names for the `name` argument of the `setAttribute()` method:

- Name beginning with `"ucnp"`
- Name beginning with `"java."`
- Name beginning with `"javax."`
- Name beginning with `"javax.portlet."`
- Name beginning with `"hptl"`
- Name beginning with `"com.cosminexus"`
- Name beginning with `"jp.co.hitachi.soft.portal"`

You can also use the URL request parameter. To use this parameter, you must acquire the `ucnpOptions` parameter by using the `ucnp.request.options` key.

You can acquire the `ucnpOptions` parameter for each session or for each window ID. We recommend that you acquire the value of the `ucnpOptions` parameter for each window ID. For details about how to acquire the values for each window ID, see *(b) param*.

Reason why data acquisition for each session is not recommended

If multiple windows of Navigation Platform are displayed in the same session, the HTTP session is overwritten with the value of the `ucnpOptions` parameter of the window that you worked with last. As a result, information of the `ucnpOptions` parameters for previously used windows is deleted.

`ucnp.request.options` key

This key is used to acquire the value of the `ucnpOptions` parameter (specified for the URL) from the session. By specifying this key in the `HttpSession.getAttribute()` method, you can acquire the URL decoded value of the `ucnpOptions` parameter. If the `ucnpOptions` parameter is not specified for the URL, null is returned.

- Specification example of the `ucnp.request.options` key (for each session) (not recommended)

```
public class IoPluginController implements IIoPluginController {

    /**
     * Execute the server processing for I/O Plugins.
     *
     * @param session
     *            The current session
     * @param param
     *            The parameter sent from the client
     */
    public Map<String, Object> inputFromNode(HttpSession session,
Map<String, Object> param){

        /* Obtain the value of the ucnpOptions parameter from the current
session. */
        String ucnpOptions = (String)
session.getAttribute("ucnp.request.options");
```

```
        if (ucnpOptions != null) {
            /* If the ucnpOptions parameter is specified, describe the
processing for that value. */
            ...
        }
    }
}
```

(b) param

This argument stores the information sent from the client during execution of the I/O Plugin. The table below provides details. Do not update the `param` parameter.

Table 5–2: Keys stored in param (inputFromNode method)

No.	Key name	Value	Description
1	`ucnp.current.params.map`	`Map<String,String>` • Key: String of one or more characters • Value: String of zero or more characters	When an I/O Plugin is executed, a Map containing the following key and value pair is passed: • Key Input parameter name of the I/O Plugin • Value Cache value of the Guide associated with the input parameter Parameters not associated with the Guide are not contained in the key. For details about cache values of Guide Parts, see *(5) Cache values of Guide Parts*.
2	`ucnp.button.type`	String One of the following: • `"start"` • `"show_next_page"` • `"show_previous_page"` • `"show_complete_page"`	Indicates the type of the clicked button or the type of transition. • `"start"` Switch from the Terminal Node (start) to a Process Node. Alternatively, display the first Process Node not connected to the Terminal Node (start). • `"show_next_page"` Switch to the next node by clicking the button or directly selecting the node. Alternatively, switch to a Process Node for which the transition destination node exists or a Process Node with the **Back** button displayed. • `"show_previous_page"` Switch to the previous node by clicking the button or directly selecting the node. • `"show_complete_page"` Switch to a node with the **Done** button displayed. Alternatively, switch to a Process Node for which the transition destination node does not exist and for which the **Back** button is hidden.
3	`ucnp.isdirectjump`	String Either of the following: • `"true"` • `"false"`	Indicates whether the transition type is direct transition. • `"true"` Direct transition • `"false"`

No.	Key name	Value	Description
3	`ucnp.isdirectjump`	String Either of the following: • `"true"` • `"false"`	Not direct transition
4	`ucnp.current.node.name`	String String of zero or more characters	The node name corresponding to the transfer-source Guide is passed. If the node name has not been set, an empty character string is passed.
5	`ucnp.next.node.name`	String String of zero or more characters	The node name corresponding to the transfer-destination Guide is passed. If the node name has not been set, an empty character string is passed.
6	`ucnp.current.node.id`	String String of zero or more characters	The node ID corresponding to the transfer-source Guide is passed. If the node ID has not been set, an empty character string is passed.
7	`ucnp.next.node.id`	String String of zero or more characters	The node ID corresponding to the transfer-destination Guide is passed. If the node ID has not been set, an empty character string is passed.
8	`ucnp.flow.contents.id`	String String of one or more characters	The Operational Content ID of the selected Operational Content is passed.
9	`ucnp.flow.contents.name`	String String of one or more characters	The Operational Content name of the selected Operational Content is passed.
10	`ucnp.flow.contents.execute.id`	String	Indicates the Operational Content execution ID, which is assigned during execution of Operational Content
11	`ucnp.flow.contents.version.id`	String	Indicates the version ID of the Operational Content being used for execution.
12	`ucnp.screen.id`	String String of one or more characters	The window ID is passed, which uniquely identifies the window being used for execution of Operational Content.
13	`ucnp.screen.ispreview`	String Either of the following: • `"true"` • `"false"`	Indicates whether the window being used for execution of Operational Content is the preview window. • `"true"` Preview window • `"false"` Operational Content Execution Window
14	`ucnp.options.param`#	String	Indicates the URL decoded value of the `ucnpOptions` parameter acquired for each window ID. This key is not set if the `ucnpOptions` parameter is not specified.

#

This key is used to acquire the value of the `ucnpOptions` parameter (specified for the URL) for each window ID. By specifying this key in the `param` argument, you can acquire the URL decoded value of the `ucnpOptions` parameter. If the `ucnpOptions` parameter is not specified for the URL, null is returned.

- Specification example of the `ucnp.options.param` key (for each window ID) (recommended)

```
public class IoPluginController implements IIoPluginController {
    /**
     * Execute the server processing for I/O Plugins.
     *
     * @param session
     *           The current session
     * @param param
     *           The parameter sent from the client
     */
   public Map<String, Object> inputFromNode(HttpSession
session,Map<String, Object> param) {
        /*  Acquire the value of the ucnpOptions parameter from the
argument. */
        String ucnpOptions = (String) param.get("ucnp.options.param");
        if (ucnpOptions != null) {
            /* If the ucnpOptions parameter is specified, describe the
processing for that value. */
        }
    }
}
```

(3) Return values

The execution result of the I/O Plugin is returned as a Map. The table below provides details. Values not covered in the table are ignored.

Table 5–3: Return values of the inputFromNode method

No.	Key name	Value	Description
1	`ucnp.error.message`	String String of zero or more characters	Set this key if you want to display a message for users after execution of the `inputFromNode` method. The string specified for this key is displayed in the message dialog box after the `inputFromNode` methods of all I/O Plugins associated with the transfer-source node are executed. Note the following when setting this key: • Use `\n` to specify a linefeed. • If this key is set for multiple I/O Plugins, the specified strings are connected and displayed in the message dialog box, using a linefeed as a delimiter. If a message is too long, it might not be displayed fully in the window. Check the message size and make sure that the whole message can be displayed in the window.
2	`ucnp.error.type`	String Either of the following: • `"NG"` • `"WARNING"`	Specify whether to suppress node transition after the message dialog box specified for `ucnp.error.message` is displayed. • `"NG"`

No.	Key name	Value	Description
2	`ucnp.error.type`	String Either of the following: • `"NG"` • `"WARNING"`	Transition is suppressed. • `"WARNING"` Transition is not suppressed. The value specified for this key is ignored in the following cases: • A value is not specified for `ucnp.error.message`. • Null is specified for `ucnp.error.message`. `"NG"` is assumed in the following cases: • A string other than `"NG"` and `"WARNING"` is specified. • Null is specified. • A value is not specified. When you execute multiple I/O Plugins, operation is different depending on the setting as shown below: • `"WARNING"` is specified for all I/O Plugins. Transition is not suppressed. • `"WARNING"` is not specified for any I/O Plugin. Transition is suppressed.
3	`ucnp.error.params.list`	`List<String>` String of one or more characters	Set this key if you want to highlight a Guide Part for which node transition is suppressed. For the value of the key, specify the list of input parameters associated with that part. The Guide Part will be displayed with a colored frame. The default frame color is red. You can change the frame color by using the `ucnp.base.client.erroritem.emphasis.border.color` property in the user property file (`ucnp_user.properties`). For details about the user property file, see the *Hitachi Navigation Platform Setup and Operations Guide*. Specify this key together with the `ucnp.error.message` key in the Map of the return value. If you set this key without setting the `ucnp.error.message` key, the highlight display is disabled. If node transition is not suppressed, the value specified for this key is ignored.

The following table describes the return values of the `inputFromNode` method and operations after execution of the I/O Plugin associated with the transition source node.

Table 5–4: Return values and operations after execution of the I/O Plugin associated with the transition source node

No.	Value specified for "ucnp.error.message"	Value specified for "ucnp.error.type"	Value specified for "ucnp.error.params.list"	Execution result of the method	Dialog box display	Node transition	Highlight display
1	Not specified, or null	`NG`	Specified (other than null)	Normal	Disabled	Disabled	Disabled
2			Not specified, or null	Normal	Disabled	Disabled	Disabled
3		`WARNING`	Specified (other than null)	Normal	Disabled	Disabled	Disabled
4			Not specified, or null	Normal	Disabled	Disabled	Disabled
5		Values other than `NG` and `WARNING`	Specified (other than null)	Normal	Disabled	Disabled	Disabled
6			Not specified, or null	Normal	Disabled	Disabled	Disabled
7	Specified (other than null)	`NG`	Specified (other than null)	Error	Enabled	Enabled	Enabled
8			Not specified, or null	Error	Enabled	Enabled	Disabled
9		`WARNING`	Specified (other than null)	Warning	Enabled	Disabled	Disabled
10			Not specified, or null	Warning	Enabled	Disabled	Disabled
11		Values other than `NG` and `WARNING`	Specified (other than null)	Error	Enabled	Enabled	Enabled
12			Not specified, or null	Error	Enabled	Enabled	Disabled

(4) Exception

None

(5) Cache values of Guide Parts

The following table describes the cache values of Guide Parts retained in plugins.

Table 5–5: Cache values for each type of Guide Part

No.	Type of part	Cache value	Default value
1	Static text	Displayed character string	The value specified in the **Displayed string** text box in the Attribute Settings window
2	Image	A string consisting of the URL and the attribute value of the tool tip connected by a linefeed code (if the tool tip is an empty character string, the linefeed code is still included in the cache value).	A string consisting of the value specified in the **URL** text box, a linefeed code, and the value specified for the **Tool tip** text box in the Attribute Settings dialog box

No.	Type of part	Cache value	Default value
3	Text box	The value entered in the text box	The value specified in the **Default value** text box in the Attribute Settings window
4	Text area	The value entered in the text area	The value specified in the **Default value** text area in the Attribute Settings window
5	Radio button	The value entered in the **Value** text box in the Attribute Settings window for the selected button (if a radio button is not selected: null)	null
6	Check box	`true`: The check box is selected `false`: The check box is cleared (In any case, a single-byte character string is used.)	The value varies depending on whether the **Change the default value to the check status** check box in the Attribute Settings window is selected. • `true` is assumed if the check box is selected. • `false` is assumed if the check box is cleared.
7	Drop down	The value selected in the **Selections list** field (Even if the value including commas or quotation marks (`"`), specify the value as is without using escape characters in CSV. For example, `Cali"fornia"` specified for the cache value is not interpreted as `"Cali""fornia"""`.	• The first value in the **Selections list** field (in the Attribute Settings window) that corresponds to a value in the **Parent value** field (if no corresponding value exists in the **Parent value** field, an empty character string is used) • If no parent drop down exists, the first value of the **Selections list** field (in the Attribute Settings window) for which the **Parent value** field is an empty character string
8	Hyperlink	A string consisting of the anchor string and the value specified for the URL, connected by a linefeed code (A tool tip is not included in the string.)	• A string consisting of the anchor string and the value specified for the URL, connected by a linefeed code • The values specified in the **Anchor String** text box and **URL** text box in the Attribute Settings window
9	Inline frame	The URL of the displayed page (The setting of whether to display a frame is not included.)	The value specified in the **URL** text box in the Attribute Settings window
10	HTML Part	Displayed HTML Content	The value specified in the **HTML String** text box in the Attribute Settings window

Notes applicable to all types of parts

- If you display the Guide area while proceeding with a task or returning to a previous step in the Operational Content Execution Window, values are displayed with the following priority:
  1. Output values of the I/O Plugin
  2. Values previously entered by the user in the Operational Content Execution Window (not including static text parts)
  3. Default cache values for each part (Table 5-5)
- The lifecycle of a cache value is from starting of a task to its end or disposal. When you resume a task that was temporarily saved by using a Suspend/Resume Plugin, the cache values are also inherited.
- A cache value is shared between Guide Parts of the same type in the Guide areas corresponding to the same Process Node. Therefore, for an Operational Flow including a branch, if a Process Node that has already been displayed is

displayed again via another route, the cache value used when the Process Node was previously displayed is used as is. If you want to change the cache value according to certain conditions, use an I/O Plugin.

- The linefeed code for cache values is `\r\n`. To include a linefeed in a cache value specified for a return value of this method, specify `\r\n` as a linefeed.

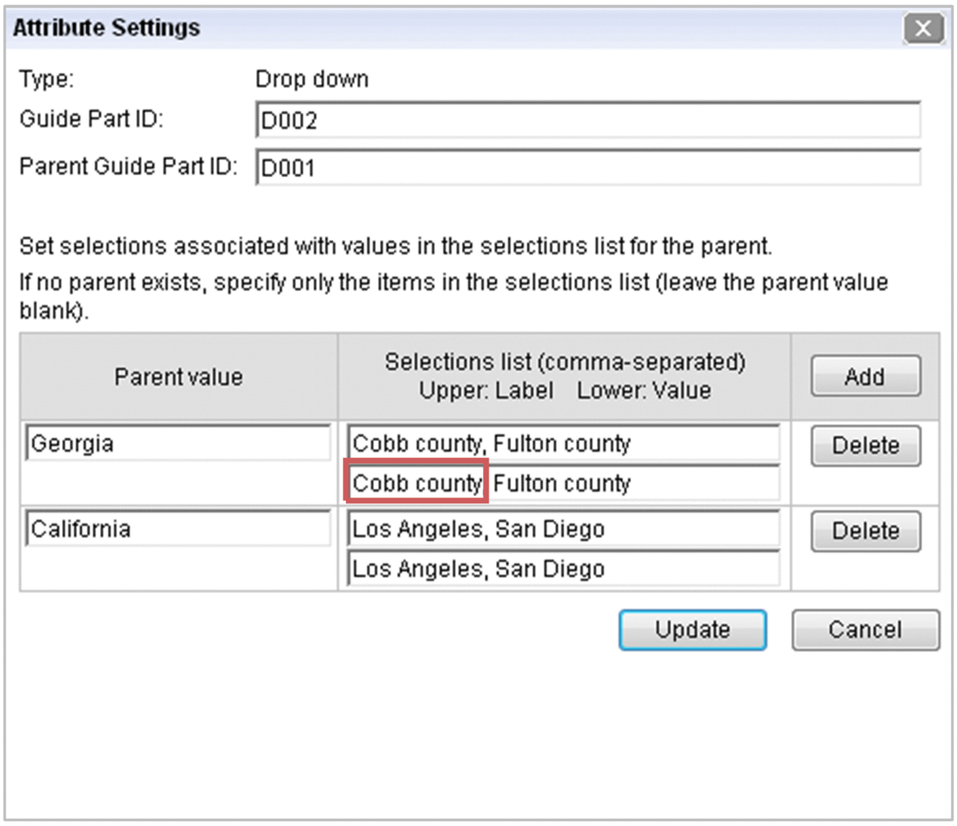
Note on text boxes

- If a string including a linefeed is output as a cache value from a plugin, the string from which the linefeed is excluded will be set as the cache value.

Notes on drop-down parts

- If the cache value specified for a parent drop-down part is different from the value specified in the plugin, the cache value of the child drop-down part is reset. The reset cache value of the child drop-down part is displayed as the first value of the corresponding parent drop-down part in the **Selections list** field in the Attribute Settings window.

  For example, the figure below shows drop-down parts specified in the Attribute Settings window. If the cache value of the parent drop-down part is `California` and the specified plugin value is `Georgia`, the reset cache value of the child drop-down part is `Cobb county`, which first corresponds to `Georgia`.


Attribute Settings

Type: Drop down

Guide Part ID: D002

Parent Guide Part ID: D001

Set selections associated with values in the selections list for the parent.

If no parent exists, specify only the items in the selections list (leave the parent value blank).

Parent value	Selections list (comma-separated) Upper: Label Lower: Value	Add
Georgia	Cobb county, Fulton county Cobb county, Fulton county	Delete
California	Los Angeles, San Diego Los Angeles, San Diego	Delete

  Note that if you change the selection status of a parent drop-down part in the Operational Content Execution Window, the values of child drop-down parts are also reset.

- Cache values are applied in the order of plugin parameters. Therefore, to update cache values of drop-down parts configured in a parent-child relationship, the child drop-down part must be associated with a parameter that appears later than the parameter associated with the parent drop-down part.

- If the value specified for the cache value is not found in the **Selections list** field corresponding to the **Parent value** field in the Attribute Settings window, the specified cache value is ignored.

Note on hyperlinks

- If the cache value does not contain a linefeed, the system assumes that only the label is specified.

5.2.2 outputToNode method

When transition to a node that contains an I/O Plugin occurs, this method performs processing to output information to the transition destination node.

You can use this method to effectively implement data reference processing, such as applying information acquired from external systems, during node transition.

If you want to switch the I/O Plugin processing for each transition path at a Branch Node, determine the transition path from the transition source node by using the I/O Plugin `outputToNode` method, and then switch the processing.

Notes on the outputToNode method

- If an I/O Plugin is placed in the transition source node and the execution result of the `inputFromNode` that was executed in advance indicates an error, the `outputToNode` methods of all I/O Plugins are not executed.
- If multiple I/O Plugins are placed in the same node and the execution result of the `outputToNode` method was executed in advance indicates an error, subsequent I/O Plugins are not executed.

Note that a method's execution results, other than error, are normal and warning.

(1) Format

```
public Map<String, Object> outputToNode(HttpSession session, Map<String,
Object> param);
```

(2) Arguments

(a) session

This argument stores the current session. To execute the `setAttribute()` method for the `HttpSession` object acquired in a plugin, do not specify any of the following names for the `name` argument of the `setAttribute()` method:

- Name beginning with "`ucnp`"
- Name beginning with "`java.`"
- Name beginning with "`javax.`"
- Name beginning with "`javax.portlet.`"
- Name beginning with "`hptl`"
- Name beginning with "`com.cosminexus`"
- Name beginning with "`jp.co.hitachi.soft.portal`"

You can also use the URL request parameter. To use this parameter, you must acquire the `ucnpOptions` parameter by using the `ucnp.request.options` key.

You can acquire the `ucnpOptions` parameter for each session or for each window ID. We recommend that you acquire the value of the `ucnpOptions` parameter for each window ID. For details about how to acquire the values for each window ID, see *(b) param*.

Reason why data acquisition for each session is not recommended

If multiple windows of Navigation Platform are displayed in the same session, the HTTP session is overwritten with the value of the `ucnpOptions` parameter of the window that you worked with last. As a result, information of the `ucnpOptions` parameters for previously used windows is deleted.

`ucnp.request.options` key

This key is used to acquire the value of the `ucnpOptions` parameter (specified for the URL) from the session. By specifying this key in the `HttpSession.getAttribute()` method, you can acquire the URL decoded value of the `ucnpOptions` parameter. If the `ucnpOptions` parameter is not specified for the URL, null is returned.

For details about a specification example of the `ucnp.request.options` key, see the specification example of the `inputFromNode` method in *5.2 IIoPluginController (server processing implementation interface)*.

(b) param

This argument stores the information sent from the client during execution of the I/O Plugin. The table below provides details. Do not update the `param` parameter.

Table 5–6: Keys stored in param (outputToNode method)

No.	Key name	Value	Description
1	`ucnp.next.params.map`	`Map<String,String>`	When an I/O Plugin is executed, a Map containing the following key and value pair is passed: • Key Output parameter name of the I/O Plugin • Value Cache value of the Guide associated with the output parameter at the point before transition Parameters not associated with the Guide are not contained in the key. For details about cache values of Guide Parts, see *5.2.1(5) Cache values of Guide Parts*.
2	`ucnp.button.type`	String One of the following: • `"start"` • `"show_next_page"` • `"show_previous_page"` • `"show_complete_page"`	Indicates the type of the clicked button or the type of transition. • `"start"` Switch from the Terminal Node (start) to a Process Node. Alternatively, display the first Process Node not connected to the Terminal Node (start). • `"show_next_page"` Switch to the next node by clicking the button or directly selecting the node. Alternatively, switch to a Process Node for which the transition destination node exists or a Process Node with the **Back** button displayed. • `"show_previous_page"` Switch to the previous node by clicking the button or directly selecting the node. • `"show_complete_page"` Switch to a node with the **Done** button displayed. Alternatively, switch to a Process Node for which the transition destination node does not exist and for which the **Back** button is hidden.

No.	Key name	Value	Description
3	`ucnp.isdirectjump`	String Either of the following: • `"true"` • `"false"`	Indicates whether the transition type is direct transition. • `"true"` Direct transition • `"false"` Not direct transition
4	`ucnp.current.node.name`	String String of zero or more characters	The node name corresponding to the transfer-source Guide is passed. If the node name has not been set, an empty character string is passed.
5	`ucnp.next.node.name`	String String of zero or more characters	The node name corresponding to the transfer-destination Guide is passed. If the node name has not been set, an empty character string is passed.
6	`ucnp.current.node.id`	String String of zero or more characters	The node ID corresponding to the transfer-source Guide is passed. If the node ID has not been set, an empty character string is passed.
7	`ucnp.next.node.id`	String String of zero or more characters	The node ID corresponding to the transfer-destination Guide is passed. If the node ID has not been set, an empty character string is passed.
8	`ucnp.flow.contents.id`	String String of one or more characters	The Operational Content ID of the selected Operational Content is passed.
9	`ucnp.flow.contents.name`	String String of one or more characters	The Operational Content name of the selected Operational Content is passed.
10	`ucnp.flow.contents.execute.id`	String	Indicates the Operational Content execution ID, which is assigned during execution of Operational Content
11	`ucnp.flow.contents.version.id`	String	Indicates the version ID of the Operational Content being used for execution.
12	`ucnp.screen.id`	String String of one or more characters	The window ID is passed, which uniquely identifies the window being used for execution of Operational Content.
13	`ucnp.screen.ispreview`	String Either of the following: • `"true"` • `"false"`	Indicates whether the window being used for execution of Operational Content is the preview window. • `"true"` Preview window • `"false"` Operational Content Execution Window
14	`ucnp.options.param`#	String	Indicates the URL decoded value of the `ucnpOptions` parameter acquired for each window ID. This key is not set if the `ucnpOptions` parameter is not specified.

#

This key is used to acquire the value of the `ucnpOptions` parameter (specified for the URL) for each window ID. By specifying this key in the `param` argument, you can acquire the URL decoded value of the `ucnpOptions` parameter. If the `ucnpOptions` parameter is not specified for the URL, null is returned. For details about the specification example of the `ucnp.options.param` key, see the specification example of the *inputFromNode method* in *5.2 IIoPluginController (server processing implementation interface)*.

(3) Return values

The execution result of the I/O Plugin is returned as a Map. The table below provides details. Values not covered in the table are ignored.

Table 5–7: Return values of the outputToNode method

No.	Key name	Value	Description
1	`ucnp.next.params.map`	`Map<String, String>`	When an I/O Plugin is executed, a Map containing the following key and value pair is passed: • Key Output parameter name of the I/O Plugin • Value Cache value of the Guide associated with the output parameter
2	`ucnp.error.message`	String	Set this key if you want to display a message for users after execution of the `outputToNode` method. If the I/O Plugin associated with the transition destination node terminates with either of the following results, the string specified for this key is displayed in the message dialog box: • The execution result type of the `outputToNode` method for any I/O Plugin is error. • The execution result type of the `outputToNode` method for all I/O Plugins is normal or warning. Note the following when setting this key: • Use `\n` to specify a linefeed. • If this key is set for multiple I/O Plugins, the specified strings are connected and displayed in the message dialog box, using a linefeed as a delimiter. If a message is too long, it might not be displayed fully in the window. Check the message size and make sure that the whole message can be displayed in the window.
3	`ucnp.error.type`	String Either of the following: • `"NG"` • `"WARNING"`	Specify whether to suppress node transition after the message dialog box specified for `ucnp.error.message` is displayed. You can specify the following strings. • `"NG"` Transition is suppressed. • `"WARNING"` Transition is suppressed. The value specified for this key is ignored in the following cases:

No.	Key name	Value	Description
3	`ucnp.error.type`	String Either of the following: • `"NG"` • `"WARNING"`	• A value is not specified for `ucnp.error.message`. • Null is specified for `ucnp.error.message`. `"NG"` is assumed in the following cases: • A string other than `"NG"` and `"WARNING"` is specified. • Null is specified. • A value is not specified. When you execute multiple I/O Plugins, operation is different depending on the setting as shown below: • `"WARNING"` is specified for all I/O Plugins. Transition is not suppressed. • `"WARNING"` is not specified for any I/O Plugin. Transition is suppressed.

The following table describes the return values of the `outputToNode` method and operations after execution of the I/O Plugin associated with the transition destination node.

Table 5–8: Return values and operations after execution of the I/O Plugin associated with the transition destination node

No.	Value specified for "ucnp.error.message"	Value specified for "ucnp.error.type"	Execution result of the method	Dialog box display	Node transition
1	Not specified, or null	`NG`	Normal	Disabled	Disabled
2		`WARNING`	Normal	Disabled	Disabled
3		Values other than `NG` and `WARNING`	Normal	Disabled	Disabled
4	Specified (other than null)	`NG`	Error	Enabled	Enabled
5		`WARNING`	Warning	Enabled	Disabled
6		Values other than `NG` and `WARNING`	Error	Enabled	Enabled

(4) Exception

None

5.2.3 Plugin processing during preview

I/O Plugins are executed when node transition occurs during preview, in addition to node transition in the Operational Content Execution Window. If you do not want I/O Plugins to be executed during a preview, use the `param` parameter when implementing the `inputFromNode` or `outputToNode` method. When this parameter is specified as shown in the following example, the system checks whether the window used to execute the plugin is a preview window, and from this check determines whether to allow plugin processing to be performed.

Implementation example

```
public Map<String,Object> outputToNode(HttpSession session,
Map<String,Object> param) {

 boolean isPreview = Boolean.valueOf((String)
param.get("ucnp.screen.ispreview"));
 if (!isPreview) {
   // Perform plugin processing only when a preview window is not used.

 }
}
```

5.3 IPluginInitializer (User Plugin startup (initialization) and termination processing implementation interface)

This interface is used to implement the processing for starting (initializing) and terminating User Plugins.

Package

```
package jp.co.hitachi.soft.ucnp.base.pluginmng.controller;
```

Format

```
public interface IPluginInitializer
```

Methods

- `init` method
  Implements the User Plugin initialization processing.
- `destroy` method
  Implements the User Plugin termination processing.

5.3.1 init method

This method is invoked when a User Plugin EAR starts. When this method is invoked, the User Plugin initialization processing is performed.

For example, when you implement a User Plugin that performs database connection processing, use this method to perform data source lookup processing. For details about how to implement database connection processing, see *4.4 Adding database connection processing*.

A data source lookup for connection with the database must be implemented within this method. If you implement the lookup with any other method, operation is unpredictable. To use the data source with the `inputFromNode()` method, or any other method, implement processing so that the data source instance found by the lookup in the `init()` method is retained in a static variable and passed.

(1) Format

```
public void init() throws UCNPPluginUserException;
```

(2) Arguments

None

(3) Return values

None

(4) Exception

If an error occurs with this method, startup of the User Plugin is interrupted and the `UCNPPluginUserException` exception is thrown.

5.3.2 destroy method

This method is invoked when a User Plugin EAR stops. When this method is invoked, the User Plugin termination processing is performed.

(1) Format

```
public void destroy();
```

(2) Arguments

None

(3) Return values

None

(4) Exception

None

5.4 UCNPPluginUserException (User Plugin exception class)

This exception is thrown by a User Plugin to Navigation Platform.

Package

```
package jp.co.hitachi.soft.ucnp.base.pluginmng.controller;
```

Format

```
public class UCNPPluginUserException extends Exception;
```

Constructor

- `UCNPPluginUserException(String)` constructor
  Invokes the `Exception(String)` constructor of the parent class.
- `UCNPPluginUserException(String, Throwable)` constructor
  Invokes the `Exception(String, Throwable)` constructor of the parent class.

5.5 ParamConvertUtil (I/O parameter conversion utility class)

This utility class converts I/O parameters to the Map format.

Package

```
package jp.co.hitachi.soft.ucnp.plugin.inputoutput.util;
```

Format

```
public class ParamConvertUtil
```

Method

- `decodeHtmlPartParam` method
  Receives encoded character strings input from an HTML Part as arguments, and then converts them to the Map format.

5.5.1 decodeHtmlPartParam method

This method receives encoded character strings input from an HTML Part as arguments, and then converts them to the Map format.

(1) Format

```
public static Map<String, String> decodeHtmlPartParam(String param)
              throws UCNPPluginException;
```

(2) Arguments

(a) param

This argument stores the encoded character strings input from the HTML Part.

(3) Return values

`Map<String, String>`

The value corresponding to an input item in the HTML Part is returned in a Map. The Map contains the `name` attribute as the key and the `value` attribute as the value. If the `param` argument is an empty character string, an empty Map is returned. You can acquire the value from the returned Map by using the `name` attribute of the element specified in the HTML Part as the key.

(4) Exception

`UCNPPluginException` - Conversion of an encoded character string fails.

If an error occurs with this method, conversion to the Map format is interrupted and the `UCNPPluginException` exception is thrown. The following shows a list of errors.

Table 5–9: List of errors that occur in the decodeHtmlPartParam method

Error	Message ID
This method is executed for data other than an HTML Part.	KDCZ10205-E

(5) Example

The following describes an example of using the `decodeHtmlPartParam` method within the `inputFromNode` method of the `IIoPluginController` interface.

Conditions

This example is based on the following conditions:

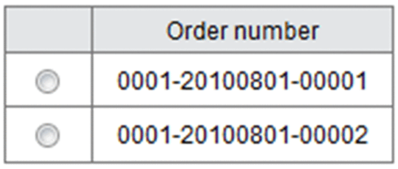
- Use a sample I/O Plugin.
  For details about sample I/O Plugins, see *A.1 How to use I/O Plugins (sample)*.
- The following external CSS files for HTML Parts are created:

```
table.sample {
    border:1px solid #777777;
    border-collapse:collapse;
    border-spacing:0;
    background-color:#ffffff;
}
th.sample {
    border-right:1px solid #777777;
    border-bottom:1px solid #777777;
    background-color:#e3e5e7;
    padding:0.3em 1em;
    text-align:center;
}
td.sample {
    border-right:1px solid #777777;
    border-bottom:1px solid #777777;
    padding:0.3em 1em;
}
```

- Input parameter `inputItemName1` is associated with the HTML Part to which the following source code is input:

```
<TABLE class="sample">
 <TBODY>
  <TR>
   <TH class="sample"></TH>
   <TH class="sample">Order number</TH>
  </TR>
  <TR>
   <TD class="sample"><input type="radio" name="order" 
value="0001-20100801-00001" /></TD>
   <TD class="sample"> 0001-20100801-00001</TD>
  </TR>
  <TR>
   <TD class="sample"><input type="radio" name="order" 
value="0001-20100801-00002" /></TD>
   <TD class="sample"> 0001-20100801-00002</TD>
  </TR>
 </TBODY>
</TABLE>
```

The HTML Part displayed in the window is as follows:



	Order number
○	0001-20100801-00001
○	0001-20100801-00002

- Output parameter `outputItemName1` is associated with static text parts.
- The order number displayed in the Guide area of the next step changes according to the radio button selected by the user.

I/O Plugin implementation example

The following shows an example of implementation to receive and process the input parameter from the HTML Part indicated in the conditions. The `decodeHtmlPartParam` method is used in the part in bold.

```
package jp.co.hitachi.soft.ucnp.plugin.sample.ioaction.controller;

import java.util.HashMap;
import java.util.Map;
import javax.servlet.http.HttpSession;
import
jp.co.hitachi.soft.ucnp.plugin.inputoutput.controller.IIoPluginController;
import
jp.co.hitachi.soft.ucnp.plugin.inputoutput.common.UCNPPluginException;
import jp.co.hitachi.soft.ucnp.plugin.inputoutput.util.ParamConvertUtil;

public class IoPluginController implements IIoPluginController {
    /* The member variable that retains the input parameter values received
from the client */
    private String param1 = null;
    private String param2 = null;
    private String param3 = null;

    public Map<String, Object> inputFromNode(HttpSession session,
            Map<String, Object> param) {

        /* Create a Map used for sending processing results to the client.
*/
        Map<String, Object> map = new HashMap<String, Object>();

        /* Processing changes depending on the button type.*/
        String buttonType = (String) param.get("ucnp.button.type");
        if ("show_next_page".equals(buttonType)) {
            /* If the Next button is clicked, input values are obtained and
then retained in the member variables. */

            /* Expand the data received from the client to the Map. */
            Map<?, ?> inParamMap = (Map<?, ?>)
param.get("ucnp.current.params.map");

           /* Obtain the value to be converted from the Map. */
           String inParam1 = (String) inParamMap.get("inputItemName1");
```

```
            Map<String, String> decodedMap = null;
            /* Enclose the API to be used in the try-catch block.*/
            try {
                decodedMap = ParamConvertUtil.decodeHtmlPartParam(inParam1);

            } catch (UCNPPluginException e) {
                /* Perform exception processing. */
                String errMsg = e.getMessage();
                /* Perform processing such as outputting plugin log data. */
                /* Return the map with an error message added.*/
                map.put("ucnp.error.message", errMsg);
                return map;
            }
            /* Obtain the value of the value attribute corresponding to the 
name attribute "order".*/
            String value = decodedMap.get("order");

            /* Set the input parameter values in the member variables. */
            param1 = value;
            param2 = (String) inParamMap.get("inputItemName2");
            param3 = (String) inParamMap.get("inputItemName3");

        } else if ("show_previous_page".equals(buttonType)) {
            /* If the Back button is clicked, nothing is performed. */
        }

        return map;
    }

    public Map<String, Object> outputToNode(HttpSession session,
            Map<String, Object> param) {

        /* Create a Map to be sent to the client. */
        Map<String, Object> map = new HashMap<String, Object>();

        /* Processing changes depending on the button type.*/
        String buttonType = (String) param.get("ucnp.button.type");
        if ("show_next_page".equals(buttonType)) {
            /* If the Next button is clicked, input values are mapped to 
the output items.  */

            Map<String, String> outParamMap = new HashMap<String, String>();

            /* Set data to be sent to the Map. */
            /* Because the static text parts are set for outputItemName1,
               the values obtained from the HTML Part are set as the values 
of the static text parts. */
            outParamMap.put("outputItemName1", param1);
            outParamMap.put("outputItemName2", param2);
            outParamMap.put("outputItemName3", param3);

            map.put("ucnp.next.params.map", outParamMap);

        } else if ("show_previous_page".equals(buttonType)) {
            /* If the Back button is clicked, nothing is performed. */
        }

        return map;
        }

}
```

5.6 UCNPPluginException (I/O Plugin exception class)

This class indicates exceptions that occurred in I/O Plugins.

Package

```
package jp.co.hitachi.soft.ucnp.plugin.inputoutput.common;
```

Format

```
public class UCNPPluginException extends Exception
```

Methods

- `getMessage` method
  Acquires detailed messages describing the causes of errors.
- `getMessageId` method
  Acquires message IDs.

5.6.1 getMessage method

This method acquires detailed messages describing the causes of errors. Message IDs are not included.

(1) Format

```
public String getMessage();
```

(2) Arguments

None

(3) Return values

Detailed messages describing the causes of errors

(4) Exception

None

5.6.2 getMessageId method

This method acquires message IDs.

(1) Format

```
public String getMessageId();
```

(2) Arguments

None

(3) Return values

Message IDs

(4) Exception

None

5.7 IUCNPSession (session information use interface)

This interface is used for using session information.

You can acquire an instance of this interface from the `HttpSession` object by using the following code:

```
IUCNPSession ucnpSession =
(IUCNPSession)session.getAttribute("ucnp.session");
```

Note that you can acquire the instance of the `IUCNPSession` interface only when you are logged in. If you are not logged in, the `ucnpSession` variable in the above code is set to null.

Package

```
package jp.co.hitachi.soft.ucnp.base.portlet;
```

Format

```
public interface IUCNPSession
```

Method

- `getLoginId` method
  Returns the user ID of the current login user.

5.7.1 getLoginId method

This method returns the user ID of the current login user.

(1) Format

```
public String getLoginId();
```

(2) Arguments

None

(3) Return values

User ID

(4) Exception

None

6 API Reference (for Suspend/Resume Plugin Development)

This chapter describes the APIs used for developing Suspend/Resume Plugins.

6.1 List of APIs (for Suspend/Resume Plugin development)

The following describes the APIs used for developing Suspend/Resume Plugins.

Table 6–1: List of APIs (for Suspend/Resume Plugin development)

Category	Interface name or class name	Description
Suspend/Resume Plugin	`ISuspendActionController`	An interface used to manipulate suspend information
	`ISuspendInfo`	An interface that indicates the suspend information to be saved or recovered
	`SupendInfoSerializeUtil`	A utility class used to serialize suspend information or recover the serialized suspend information
	`UCNPPluginException`	A class that indicates exceptions that occurred in Suspend/Resume Plugins

For details about the APIs that can be used with any type of plugin, see *5.1 List of APIs (for I/O Plugin development)*.

6.2 ISuspendActionController (suspend/resume action controller interface)

This interface is used to manipulate suspend information in Suspend/Resume Plugins.

Package

```
package jp.co.hitachi.soft.ucnp.plugin.suspend.controller;
```

Format

```
public Interface ISuspendActionController
```

Methods

- `save` method
  Saves suspend information associated with Operational Content IDs.
- `load` method
  Loads the suspend information associated with Operational Content IDs.
- `contains` method
  Checks whether suspend information associated with Operational Content IDs exists.
- `delete` method
  Deletes the suspend information associated with Operational Content IDs.
- `deleteAll` method
  Deletes all suspend information associated with Operational Content IDs.

About an instance of a user implementation class that inherits `ISuspendActionController`

An instance of a user implementation class that inherits `ISuspendActionController` is created only once when a method is invoked for the first time. All methods are invoked for the same instance.

6.2.1 save method

This method saves suspend information in Suspend/Resume Plugins. It saves the suspend information associated with the Operational Content ID specified by the parameter.

If this method throws the `UCNPPluginException` exception, Navigation Platform displays the message for the `UCNPPluginException` exception in the dialog box. A message ID is not appended to the message displayed in the dialog box.

Processing of this method must be implemented in the suspend/resume action controller class created by using the template plugin creation command.

(1) Format

```
public void save(String contentId, ISuspendInfo suspendInfo, HttpSession
session)
                  throws UCNPPluginException;
```

(2) Arguments

(a) contentId

This argument indicates the Operational Content ID.

(b) suspendInfo

This argument stores the suspend information of the Operational Content selected when the **Suspend** button is clicked.

(c) session

This argument stores the current session. To execute the `setAttribute()` method for the `HttpSession` object acquired in a plugin, do not specify any of the following names for the `name` argument of the `setAttribute()` method:

- Name beginning with "`java.`"
- Name beginning with "`javax.`"
- Name beginning with "`javax.portlet.`"
- Name beginning with "`hptl`"
- Name beginning with "`com.cosminexus`"
- Name beginning with "`jp.co.hitachi.soft.portal`"

You can also acquire and use the URL request parameter. To use this parameter, you must acquire the `ucnpOptions` parameter by using the following key:

`ucnp.request.options` key

This key is used to acquire the value of the `ucnpOptions` parameter specified for the URL from the session. By specifying this key in the `HttpSession.getAttribute()` method, you can acquire the URL decoded value of the `ucnpOptions` parameter. If the `ucnpOptions` parameter is not specified for the URL, null is returned.

For details about a specification example of the `ucnp.request.options` key, see the specification example of the *inputFromNode method* in *5.2 IIoPluginController (server processing implementation interface)*.

(3) Return values

None

(4) Exception

`UCNPPluginException` - An error occurred while, for example, suspend information was being saved.

Navigation Platform displays the detailed message for the `UCNPPluginException` exception thrown by this method in the dialog box in the Operational Content Execution Window. For the `UCNPPluginException` exception to be thrown, set the message that helps users understand what kind of error occurred and what action to take. If the error message output by the Suspend/Resume Plugin contains many linefeeds, the message dialog box might not be fully displayed in the window because a JavaScript Alert is used to display the error message. Therefore, make sure that the error message size does not exceed the limit that can be displayed in the window.

(5) Invocation timing

Navigation Platform invokes this method when:

- The **Suspend** button is clicked.

6.2.2 load method

This method loads and recovers suspend information in Suspend/Resume Plugins, by loading the suspend information associated with the Operational Content ID specified by a parameter.

This method is invoked only if the return value of the `contains()` method is `true`. Therefore, if the return value of this method is null, a plugin error message is displayed in the dialog box. If no suspend information is found, throw the `UCNPPluginException` exception for which a message is set.

If this method throws the `UCNPPluginException` exception, Navigation Platform displays the message for the `UCNPPluginException` exception in the dialog box. A message ID is not appended to the message displayed in the dialog box.

If the return value of this method is invalid or if this method throws the `UCNPPluginException` exception, a message appears in the dialog box, and then the operation starts from the beginning.

(1) Format

```
public ISuspendInfo load(String contentId, HttpSession session)
                throws UCNPPluginException;
```

(2) Arguments

(a) contentId

This argument indicates the Operational Content ID.

(b) session

This argument stores the current session. To execute the `setAttribute()` method for the `HttpSession` object acquired in a plugin, do not specify any of the following names for the `name` argument of the `setAttribute()` method:

- Name beginning with "`ucnp`"
- Name beginning with "`java.`"
- Name beginning with "`javax.`"
- Name beginning with "`javax.portlet.`"
- Name beginning with "`hptl`"
- Name beginning with "`com.cosminexus`"
- Name beginning with "`jp.co.hitachi.soft.portal`"

You can also acquire and use the URL request parameter. To use this parameter, you must acquire the `ucnpOptions` parameter by using the following key:

`ucnp.request.options` key

This key is used to acquire the value of the `ucnpOptions` parameter specified for the URL from the session. By specifying this key in the `HttpSession.getAttribute()` method, you can acquire the URL decoded value of the `ucnpOptions` parameter. If the `ucnpOptions` parameter is not specified for the URL, null is returned.

For details about a specification example of the `ucnp.request.options` key, see the specification example of the *inputFromNode method* in *5.2 IIoPluginController (server processing implementation interface)*.

(3) Return values

Suspend information associated with the Operational Content ID

(4) Exception

`UCNPPluginException` - An error occurred while, for example, suspend information was being loaded.

Navigation Platform displays the detailed message for the `UCNPPluginException` exception thrown by this method in the dialog box in the Operational Content Execution Window. For the `UCNPPluginException` exception to be thrown, set the message that helps users understand what kind of error occurred and what action to take. If the error message output by the Suspend/Resume Plugin contains many linefeeds, part of the message dialog box might not be displayed in the window because a JavaScript Alert is used to display the error message. Therefore, make sure that the error message size does not exceed the limit that can be displayed in the window.

(5) Invocation timing

Navigation Platform invokes this method if the following occurs:

- After the `contains()` method of the implementation class that inherits the suspend/resume action controller interface was invoked, that `contains()` method returns `true`.

  If the `contains()` method returns `false` or if the `UCNPPluginException` exception is thrown, this method is not invoked. For details about the time that the `contains()` method is invoked, see the description of the *contains method* in *6.2 ISuspendActionController (suspend/resume action controller interface)*.

6.2.3 contains method

This method checks the presence of suspend information associated in Suspend/Resume Plugins. It checks whether the suspend information associated with the Operational Content ID specified by the parameter exists. This method returns `true` if the associated suspend information exists, and returns `false` if such information does not exist.

If this method throws the `UCNPPluginException` exception, Navigation Platform displays the message for the `UCNPPluginException` exception in the dialog box. A message ID is not appended to the message displayed in the dialog box.

Processing of this method must be implemented in the suspend/resume action controller class created by using the template plugin creation command.

(1) Format

```
public boolean contains(String contentId, HttpSession session)
          throws UCNPPluginException;
```

(2) Arguments

(a) contentId

This argument indicates the Operational Content ID.

(b) session

This argument stores the current session. To execute the `setAttribute()` method for the `HttpSession` object acquired in a plugin, do not specify any of the following names for the `name` argument of the `setAttribute()` method:

- Name beginning with `"ucnp"`
- Name beginning with `"java."`
- Name beginning with `"javax."`
- Name beginning with `"javax.portlet."`
- Name beginning with `"hptl"`
- Name beginning with `"com.cosminexus"`
- Name beginning with `"jp.co.hitachi.soft.portal"`

You can also acquire and use the URL request parameter. To use this parameter, you must acquire the `ucnpOptions` parameter by using the following key:

`ucnp.request.options` key

This key is used to acquire the value of the `ucnpOptions` parameter specified for the URL from the session. By specifying this key in the `HttpSession.getAttribute()` method, you can acquire the URL decoded value of the `ucnpOptions` parameter. If the `ucnpOptions` parameter is not specified for the URL, null is returned.

For details about a specification example of the `ucnp.request.options` key, see the specification example of the *inputFromNode method* in *5.2 IIoPluginController (server processing implementation interface)*.

(3) Return values

- `true`: Suspend information associated with the Operational Content ID exists.
- `false`: Suspend information associated with the Operational Content ID does not exist.

(4) Exception

`UCNPPluginException` - An error occurred while, for example, suspend information was being loaded.

Navigation Platform displays the detailed message for the `UCNPPluginException` exception thrown by this method in the dialog box in the Operational Content Execution Window. For the `UCNPPluginException` exception to be thrown, specify the message that helps users understand what kind of error occurred and what action to take. If the error message output by the Suspend/Resume Plugin contains many linefeeds, part of the message dialog box might not be displayed in the window because a JavaScript Alert is used to display the error message. Therefore, make sure that the error message size does not exceed the limit that can be displayed in the window.

(5) Invocation timing

Navigation Platform invokes this method if either of the following occurs:

- A user selects Operational Content in the Operational Content Execution Window.
- A URL is directly specified and Operational Content is selected by using the `contentId` parameter that directly specifies the Operational Content ID.

6.2.4 delete method

This method deletes suspend information in Suspend/Resume Plugins. It deletes suspend information associated with the Operational Content ID specified by the parameter.

If this method throws the `UCNPPluginException` exception, Navigation Platform displays the message for the `UCNPPluginException` exception in the dialog box. A message ID is not appended to the message displayed in the dialog box.

Processing of this method must be implemented in the suspend/resume action controller class created by using the template plugin creation command.

(1) Format

```
public void delete(String contentId, HttpSession session)
                                   throws UCNPPluginException;
```

(2) Arguments

(a) contentId

This argument indicates the Operational Content ID.

(b) session

This argument stores the current session. To execute the `setAttribute()` method for the `HttpSession` object acquired in a plugin, do not specify any of the following names for the `name` argument of the `setAttribute()` method:

- Name beginning with "`ucnp`"
- Name beginning with "`java.`"
- Name beginning with "`javax.`"
- Name beginning with "`javax.portlet.`"
- Name beginning with "`hptl`"
- Name beginning with "`com.cosminexus`"
- Name beginning with "`jp.co.hitachi.soft.portal`"

You can also acquire and use the URL request parameter. To use this parameter, you must acquire the `ucnpOptions` parameter by using the following key.

`ucnp.request.options` key

This key is used to acquire the value of the `ucnpOptions` parameter specified for the URL from the session. By specifying this key in the `HttpSession.getAttribute()` method, you can acquire the URL decoded value of the `ucnpOptions` parameter. If the `ucnpOptions` parameter is not specified for the URL, null is returned.

For details about a specification example of the `ucnp.request.options` key, see the specification example of the *inputFromNode method* in *5.2 IIoPluginController (server processing implementation interface)*.

(3) Return values

None

(4) Exception

`UCNPPluginException` - An error occurred while, for example, suspend information was being deleted.

Navigation Platform displays the detailed message for the `UCNPPluginException` exception thrown by this method in the dialog box in the Operational Content Execution Window. For the `UCNPPluginException` exception to be thrown, set the message that helps users understand what kind of error occurred and what action to take. If the error message output by the Suspend/Resume Plugin contains many linefeeds, part of the message dialog box might not be displayed in the window because a JavaScript Alert is used to display the error message. Therefore, make sure that the error message size does not exceed the limit that can be displayed in the window.

(5) Invocation timing

Navigation Platform invokes this method if either of the following occurs:

- A user clicks the **Done** button in the Operational Content Execution Window.
- After the `load` method of the implementation class that inherits the suspend/resume action controller interface was invoked, an error occurs during the check of the suspend information returned by that `load` method.

  During the check, the system checks whether Operational Content has been updated since the suspend information was saved, and then assumes an error if any update was found. For details about the time that the `load` method is invoked, see the description of the *load method* in *6.2 ISuspendActionController (suspend/resume action controller interface)*.

Note:

This function is invoked only when Suspend/Resume Plugins are enabled in the system. If Suspend/Resume Plugins are disabled in the system, this function cannot be invoked even if the above conditions exist.

6.2.5 deleteAll method

This method deletes all suspend information associated with the Operational Content in Suspend/Resume Plugins. This method deletes all suspend information associated with the Operational Content ID specified by the parameter.

If a user who creates Operational Content edits or deletes Operational Content, the user invokes this method to delete all the associated suspend information. If Operational Content is edited or deleted, the saved suspend information is disabled. Therefore, delete all suspend information associated with the Operational Content.

If this method throws the `UCNPPluginException` exception, Navigation Platform displays the message for the `UCNPPluginException` exception in the dialog box. A message ID is not appended to the message displayed in the dialog box.

Processing of this method must be implemented in the suspend/resume action controller class created by using the template plugin creation command.

(1) Format

```
public void deleteAll(String contentId, HttpSession session)
                                      throws UCNPPluginException;
```

(2) Arguments

(a) contentId

This argument indicates the Operational Content ID.

(b) session

This argument stores the current session. If this method is executed during publishing start processing of Operational Content in the publishing reservation status, the value is null. To execute the `setAttribute()` method for the `HttpSession` object acquired in a plugin, do not specify any of the following names for the `name` argument of the `setAttribute()` method:

- Name beginning with `"ucnp"`
- Name beginning with `"java."`
- Name beginning with `"javax."`
- Name beginning with `"javax.portlet."`
- Name beginning with `"hptl"`
- Name beginning with `"com.cosminexus"`
- Name beginning with `"jp.co.hitachi.soft.portal"`

You can also acquire and use the URL request parameter. To use this parameter, you must acquire the `ucnpOptions` parameter by using the following key:

`ucnp.request.options` key

This key is used to acquire the value of the `ucnpOptions` parameter specified for the URL from the session. By specifying this key in the `HttpSession.getAttribute()` method, you can acquire the URL decoded value of the `ucnpOptions` parameter. If the `ucnpOptions` parameter is not specified for the URL, null is returned.

For details about a specification example of the `ucnp.request.options` key, see the specification example of the *inputFromNode method* in *5.2 IIoPluginController (server processing implementation interface)*.

(3) Return values

None

(4) Exception

`UCNPPluginException` - An error occurred while, for example, suspend information was being deleted.

Navigation Platform displays the detailed message for the `UCNPPluginException` exception thrown by this method in the dialog box in the Operational Content Execution Window. For the `UCNPPluginException` exception to be thrown, set the message that helps users understand what kind of error occurred and what action to take. If the error message output by the Suspend/Resume Plugin contains many linefeeds, part of the message dialog box might not be displayed in the window because a JavaScript Alert is used to display the error message. Therefore, make sure that the error message size does not exceed the limit that can be displayed in the window.

(5) Invocation timing

The following table describes when Navigation Platform invokes this method and the suspend information to be deleted.

Table 6–2: When the deleteAll method is invoked and suspend information to be deleted

Invoked when:	Suspend information to be deleted
The contents of Operational Content displayed in the Operational Content Execution Window change by operation on the Web browser.	All suspend information associated with the target Operational Content
A user who creates Operational Content deletes Operational Content in the Operational Content Editing Window.	All suspend information associated with the deleted Operational Content

6.3 ISuspendInfo (suspend information interface)

This interface indicates suspend information in Suspend/Resume Plugins. This interface is used to send and receive suspend information to be saved or recovered by each method of the `ISuspendActionController` interface. An object of this interface is created by the system and therefore, you do not need to create that object for plugins.

Package

```
package jp.co.hitachi.soft.ucnp.plugin.suspend.model;
```

Format

```
public interface ISuspendInfo
```

Methods

- `getContentSuspendedId` method
  Acquires the suspend ID, which can be used to match the log data output to the audit log with suspend information.
- `getWorkId` method
  Acquires the work IDs, which can be used for matching to check whether the operations output to the operation log are the same as the operations that were suspended.
- `getContentName` method
  Acquires the name of Operational Content being used for execution when operation is suspended.
- `getGroupName` method
  Acquires the group name of Operational Content being used for execution when operation is suspended.
- `getCurrentNodeName` method
  Acquires the name of the node being executed when operation is suspended.

6.3.1 getContentSuspendedId method

This method acquires the suspend ID, which can be used to match the log data output to the audit log with suspend information.

(1) Format

```
public String getContentSuspendedId();
```

(2) Arguments

None

(3) Return values

Suspend ID

(4) Exception

None

6.3.2 getWorkId method

This method acquires the work IDs, which can be used for matching to check whether the operations output to the operation log are the same as those that were suspended.

(1) Format

```
public String getWorkId();
```

(2) Arguments

None

(3) Return values

Work ID

(4) Exception

None

6.3.3 getContentName method

This method acquires the name of Operational Content being used for execution when operation is suspended.

(1) Format

```
public String getContentName();
```

(2) Arguments

None

(3) Return values

Name of Operational Content being used for execution when operation is suspended

(4) Exception

None

6.3.4 getGroupName method

This method acquires the group name of Operational Content being used for execution when operation is suspended.

(1) Format

```
public String getGroupName();
```

(2) Arguments

None

(3) Return values

Group name of Operational Content being used for execution when operation is suspended

(4) Exception

None

6.3.5 getCurrentNodeName method

This method acquires the name of the node being executed when operation is suspended.

(1) Format

```
public String getCurrentNodeName();
```

(2) Augments

None

(3) Return values

Name of the node being executed when operation is suspended

(4) Exception

None

6.4 SupendInfoSerializeUtil (utility class for suspend information serialization)

This utility class provides the methods for serializing and deserializing suspend information (`ISuspendInfo`).

You can use the `serialize` method of this class to serialize suspend information (`ISuspendInfo`) to XML format. You can also use the `deserialize` method to recover the suspend information (`ISuspendInfo`) serialized to XML format.

Package

```
package jp.co.hitachi.soft.ucnp.plugin.suspend.util;
```

Format

```
public class SuspendInfoSerializeUtil
```

Methods

- `serialize` method
  Serializes suspend information to XML format.
- `deserialize` method
  Loads and recovers the serialized suspend information.

6.4.1 serialize method

This method is used to serialize suspend information in Suspend/Resume Plugins. It serializes the suspend information (`suspendInfo`) specified by the parameter to XML format, and then writes the results into the output stream `out`. This method does not close the output stream `out`. The invocation side is responsible for performing the close processing.

If an attempt to serialize the suspend information or output the information to the stream fails, the method throws the `UCNPPluginException` exception. A Navigation Platform message ID is added to the detailed message for the `UCNPPluginException` exception thrown by this method. You need to handle the thrown exception and then throw the `UCNPPluginException` exception for which the message to be displayed in the dialog box is set.

(1) Format

```
public static void serialize(ISuspendInfo suspendInfo, OutputStream out)
                   throws UCNPPluginException;
```

(2) Arguments

(a) suspendInfo

This argument stores suspend information.

(b) out

This argument stores the output stream to which the serialized suspend information is written.

(3) Return values

None

(4) Exception

`UCNPPluginException` - An attempt to serialize and output suspend information fails.

If an error occurs with this method, serialization processing is interrupted and the `UCNPPluginException` exception is thrown. The following shows a list of errors.

Table 6–3: List of errors that occur in the serialize method

Error	Message ID
Null is specified for the `suspendInfo` parameter.	KDCZ10083-E
Null is specified for the `out` parameter.	KDCZ10083-E
An invalid object (an object not created by Navigation Platform) is specified for the `suspendInfo` parameter.	KDCZ10084-E
It was not possible to write information to the output stream specified by the parameter due to a problem with access permissions or disk space.	KDCZ10085-E

6.4.2 deserialize method

This method is used to recover the serialized suspend information in Suspend/Resume Plugins. It loads the serialized suspend information from the input stream `in` specified by the parameter, and then recovers the suspend information. This method does not close the input stream `in`. The invocation side is responsible for performing the close processing.

If an attempt to recover the serialized suspend information or input the information from the stream fails, the method throws the `UCNPPluginException` exception. A system message ID is added to the message for the exception thrown by this method. You need to handle the thrown exception and then throw the `UCNPPluginException` exception for which the message to be displayed in the dialog box is set.

(1) Format

```
public static ISuspendInfo deserialize(InputStream in)throws
UCNPPluginException;
```

(2) Arguments

(a) in

This argument stores the input stream from which the deserialized suspend information is loaded.

(3) Return values

Serialized suspend information recovered from the input stream

(4) Exception

`UCNPPluginException` - An attempt to recover and input suspend information fails.

If an error occurs with this method, deserialization processing is interrupted and the `UCNPPluginException` exception is thrown. The following shows a list of errors.

Table 6–4: List of errors that occur in the deserialize method

Error	Message ID
Null is specified for the `in` parameter.	KDCZ10083-E
The input stream specified for the `in` parameter cannot be used to recover suspend information (`ISuspendInfo`).	KDCZ10086-E
It was not possible to load information from the input stream specified by the parameter due to a problem with access permissions.	KDCZ10093-E

6.5 UCNPPluginException (Suspend/Resume Plugin exception class)

This class indicates exceptions that occurred in Suspend/Resume Plugins.

If an error occurs in a method of `ISuspendActionController`, the method throws this exception with a message set. The exception message is displayed in the dialog box.

If the message describing the Suspend/Resume Plugin error is too long, the full text might not be displayed because, depending on the Web browser, the string might be truncated in the message dialog box.

Package

```
package jp.co.hitachi.soft.ucnp.plugin.suspend.common;
```

Format

```
public class UCNPPluginException extends Exception
```

Constructors

- `UCNPPluginException(String message)` constructor
  Creates a new exception by using the specified detailed message.
- `UCNPPluginException(String message, Throwable cause)` constructor
  Creates a new exception by using the specified detailed message and cause of the error.

6.5.1 UCNPPluginException(String message) constructor

This constructor creates a new exception by using the specified detailed message.

(1) Format

```
public UCNPPluginException(String message);
```

(2) Arguments

(a) message

This argument displays the detailed message indicating the cause of the error.

This message is displayed in the dialog box in the Operational Content Execution Window. Therefore, specify the message that helps users to understand error details and how to take action.

(3) Exception

None

6.5.2 UCNPPluginException(String message, Throwable cause) constructor

This constructor creates a new exception by using the specified detailed message and cause of the error.

(1) Format

```
public UCNPPluginException(String message, Throwable cause);
```

(2) Arguments

(a) message

This argument displays the detailed message indicating the cause of the error.

This message is displayed in the dialog box in the Operational Content Execution Window. Therefore, specify the message that helps users to understand error details and how to take action.

(b) cause

This argument stores the `Throwable` object that caused the error.

(3) Exception

None

7

API Reference (for Custom Window Plugin Development)

This chapter describes the APIs used for developing Custom Window Plugins.

7.1 List of APIs (for Custom Window Plugin development)

The following describes the APIs used for developing Custom Window Plugins.

Table 7–1: List of APIs (Custom Window Plugin development)

Category	Interface name or class name	Description
Logout function	`CustomWindowUrlUtil`	A utility class used to acquire the URL of a custom window
	`LogoutActionUtil`	A utility class used to log out from Navigation Platform

For details about the APIs that can be used with any type of plugin, see *5.1 List of APIs (for I/O Plugin development)*.

7.2 CustomWindowUrlUtil (utility class for custom window URL acquisition)

This utility class is used to acquire the URL of a custom window.

Package

```
package jp.co.hitachi.soft.ucnp.base.common;
```

Format

```
public class CustomWindowUrlUtil
```

Method

- `getCustomWindowUrl` method
  Acquires the URL of a custom window.

7.2.1 getCustomWindowUrl method

This method acquires the URL of a custom window. You need to specify the `ucnpUserPageId` parameter for the URL acquired by this method.

(1) Format

```
public static String getCustomWindowUrl(HttpServletRequest request,
HttpServletResponse response);
```

(2) Arguments

None

(3) Return values

The custom window URL beginning with `ucnpBase` is returned.

(4) Exception

None

(5) Example of use

The following shows an example when the `form` tag is used in the JSP file of the custom window.

```
<%
  String url = CustomWindowUrlUtil.getCustomWindowUrl(request, response);
%>

<form action="<%= url %>" method="post" target="_self">
<input type="hidden" name="ucnpUserPageId" value="userCustomPage" />
```

```
(snip)
</form>
```

7.3 LogoutActionUtil (logout processing class)

This utility class is used to log out from Navigation Platform. This class is used in the JSP file. To make a branch to the logout processing from normal processing in the custom window, use the `ucnpUserData` parameter.

For details about the `ucnpUserData` parameter, see *2.5.2 Data that can be received by Custom Window Plugins*.

Package

```
package jp.co.hitachi.soft.ucnp.base.common;
```

Format

```
public class LogoutActionUtil
```

Method

- `logout` method

  Logs out from Navigation Platform.

7.3.1 logout method

This method logs out from Navigation Platform. If you have not logged in to Navigation Platform, nothing is performed.

To check the logout results, see the trace file. If an attempt to log out fails, the KDCZ18033-E message is output to the trace file.

The default storage directory of the trace file is as follows:

Storage directory

  Navigation-Platform-installation-directory`\logs\ucnp_trace`*[N]*`.log` (*[N]*: number of files)

(1) Format

```
public static void logout(HttpServletRequest request, HttpServletResponse
response);
```

(2) Arguments

(a) request

Specify the JSP implicit object request. If any other value is specified, an error occurs.

(b) response

Specify the JSP implicit object request. If any other value is specified, an error occurs.

(3) Return values

None

(4) Exception

None

Appendixes

A. How to Use Sample Plugins

Navigation Platform for Developers provides the following sample plugins:

- I/O Plugin (sample)

  When a node transition occurs, this plugin retains input values of Guide Parts associated with Plugin Part parameters. When the operation is completed, the retained input values are output to a file.
- Suspend/Resume Plugin (sample)

  This plugin is useful when you use multiple Operational Contents in parallel. Even if you reference another Operational Content or log out in the middle of operation, this plugin allows you to resume the Operational Content operation from the temporarily saved status.

  Suspend information is saved in a file system on the server. When you resume the operation, suspend information is read from the file system on the server.

This section describes how to use sample plugins. When using sample plugins, you must note the restrictions in *A.3 Notes on using sample plugins*.

A.1 How to use I/O Plugins (sample)

To use the I/O Plugin (sample):

1. Copy the sample plugin to the plugin development work directory.

   The following indicates the file to be copied and the destination directory.

   File to be copied

   Navigation-Platform-for-Developers-installation-directory`\sample\plugin\sample.OutputGuideData`

   Destination directory

   Navigation-Platform-for-Developers-installation-directory`\pluginSDK\plugin`

2. Build the plugin.

   For details about how to build the plugin, see *4.6 Building plugins*.

3. Deploy and start the plugin.

   For details about how to deploy the plugin, see *4.7 Deploying plugins*.

4. Create the output destination folder for files to which input values are output when an operation is completed.

   To use the sample plugin as is:

   `C:\ucnpwork`

   To change the output folder:

   Customize the plugin. For details about how to customize plugins, see *4.3.1(8) Implementing processing to be performed by the plugin*.

5. Associate the sample plugin with Operational Content, and then check the operation

   For details about how to associate sample plugins with Operational Content, see *4.8 Associating I/O Plugins with Operational Content*. To use sample plugins, you need to place Plugin Parts and a Terminal Node in the Operational Flow as follows:

   - Place a Plugin Part in the start Process Node.

- Place a Terminal Node at the end point of the Operational Flow, and place a Plugin Part in the Process Node immediately before the Terminal Node.

A.2 How to use Suspend/Resume Plugins (sample)

To use a Suspend/Resume Plugin (sample):

1. Copy the sample plugin to the plugin development work directory.
   The following indicates the file to be copied and the destination directory.
   File to be copied
   Navigation-Platform-for-Developers-installation-directory`\sample\plugin\ucnp.plugin.suspend`
   Destination directory
   Navigation-Platform-for-Developers-installation-directory`\pluginSDK\plugin`
2. Edit the value specified for the constant `DEFAULT_SAVE_FOLDER` to change the folder to which suspend information will be output.
3. Build the plugin.
   For details about how to build the plugin, see *4.6 Building plugins*.
4. Enable the Suspend/Resume Plugin in the system.
   Specify `true` for the `ucnp.base.client.suspend.enable` key in the user property file (`ucnp_user.properties`). For details about how to configure the user property file, see the *Hitachi Navigation Platform Setup and Operations Guide*.

A.3 Notes on using sample plugins

Note the following when using sample plugins:

- Sample plugins do not contain error processing. If necessary, add error processing.
- Before using plugins, sufficiently confirm the operation to ensure that no error occurs.
- If a user without Windows administrator roles adds a file to copy the sample plugin to a work directory under the *OS-installation-drive*`:\Program Files` directory, the file might be redirected to a user folder. Therefore, the user who adds the file must have Windows administrator privileges.
- If an installation directory other than the default is used in an attempt to import a sample plugin project into Eclipse, building of the project fails with the following error messages:

```
Project 'OutputGuideData' is missing required library: 'C:\Program Files
(x86)\Hitachi\HNP\lib\ucnppluginif.jar'
Project 'OutputGuideData' is missing required library: 'C:\Program Files
(x86)\Hitachi\HNP\syslib\ucnpsys.jar'
Project 'OutputGuideData' is missing required library: 'C:\Program Files
(x86)\Hitachi\HNP\lib\ucnpclasses.jar'
Project 'OutputGuideData' is missing required library: 'C:\Program Files
(x86)\Hitachi\HNP\PP\uCPSB\CC\client\lib\j2ee-javax.jar'
```

  In this case, change the JAR file paths displayed in the message as follows:
  Navigation-Platform-installation-directory`\lib\ucnppluginif.jar`

Navigation-Platform-installation-directory`\syslib\ucnpsys.jar`

Navigation-Platform-installation-directory`\lib\ucnpclasses.jar`

Navigation-Platform-installation-directory`\PP\uCPSB\CC\client\lib\j2ee-javax.jar`

B. Important Point for I/O Plugin Development

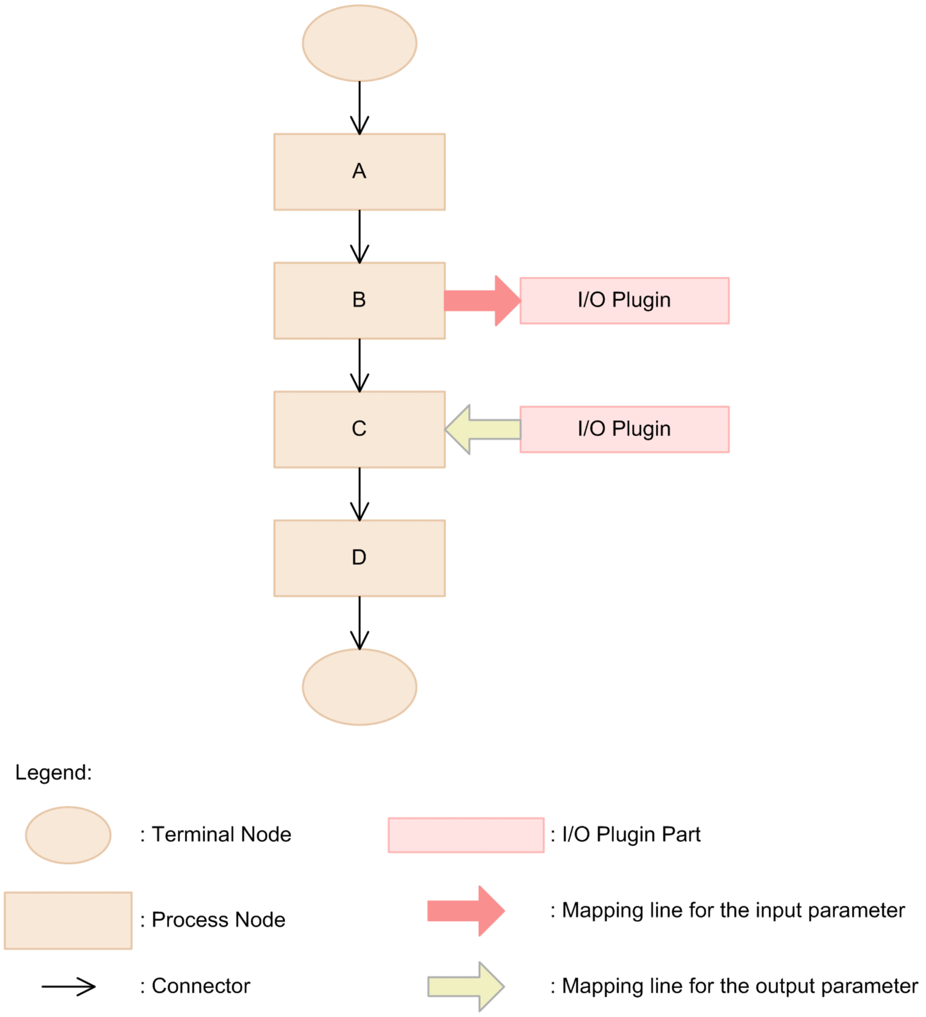
This section describes the important point for developing I/O Plugins.

B.1 Suppressing execution of I/O Plugins depending on the presence of mapping lines

You can suppress execution of I/O Plugins depending on the presence of mapping lines between I/O parameters and Guide Parts.

The following describes suppression of the I/O Plugin processing in the example of Operational Content shown below.

Figure B–1: Example of Operational Content



If transition occurs from Process Node A to Process Node D in Operational Content in this figure, the I/O Plugins perform processing in the following order:

1. `outputToNode` method of the I/O Plugin placed in Process Node B
2. `inputFromNode` method of the I/O Plugin placed in Process Node B
3. `outputToNode` method of the I/O Plugin placed in Process Node C

4. `inputFromNode` method of the I/O Plugin placed in Process Node C

If you want to suppress the processing of the method executed in 1, confirm the following and then suppress the processing:

- The value of the `ucnp.next.params.map` key in the `param` argument of the `outputToNode` method is an empty Map.

If you want to suppress the processing of the method executed in 4, confirm the following and then suppress the processing:

- The value of the `ucnp.current.params.map` key in the `param` argument of the `inputFromNode` method is an empty Map.

Note that if you do not want to suppress the processing of the method executed in 4, draw a mapping line to the input parameter of the I/O Plugin in Process Node C. This allows the processing to be performed without being suppressed.

C. Migration from Old Versions

This section describes the procedure for migrating old versions of development environment settings and developed files to Navigation Platform. This section also describes the functions that cannot be migrated from old versions and the functions that do not need migration.

C.1 Procedure for migrating plugins developed in old versions

By migrating I/O Plugins and Suspend/Resume Plugins developed in an old version (version 09-50 or later), you can use them in Navigation Platform. To migrate such plugins:

1. See *4.1 Creating template plugins* and create a new template plugin.

   Apply the values specified in version 09-50 or later to the plugin information property file of the new template.

   Values you must apply:

   The value specified for the `userplugin.id` key

   The value specified for the `userplugin.name` key

   The value specified for the `userplugin.server.controller.ioaction.type` key

   For details about the storage location of the plugin information property file for plugins of version 09-50 or later, see the manual used for development.

2. Overwrite the new template plugin files with the files of version 09-50 or later.

Table C–1: List of files for overwriting (common)

No.	Files of version 09-50 or later	Copy destination folder	Copy is required if:
1	Files directly under *Navigation-Platform-installation-directory*`\pluginSDK\plugin\`*plugin-name*`\WEB-INF\lib`	*Navigation-Platform-installation-directory*`\pluginSDK\plugin\`*plugin-name*`\WEB-INF\lib`	The file to be loaded to Application Class Loader is added.
2	Files directly under *Navigation-Platform-installation-directory*`\usrlib\app`#1		
3	Files directly under *Navigation-Platform-installation-directory*`\usrlib\sys`#2	*Navigation-Platform-installation-directory*`\usrlib\sys`	The file to be loaded to System Class Loader is added.
4	Files directly under the path specified for the `add.class.path` key added by the developer to the option definition file for J2EE servers (`usrconf.cfg`)#2		
5	*Navigation-Platform-installation-directory*`\pluginSDK\base\dd\META-INF\cosminexus.xml`#1	*Navigation-Platform-installation-directory*`\pluginSDK\plugin\`*plugin-name*`\dd\META-INF`	Resource adapter settings are added.
6	*portal-project*`\WEB-INF\web.xml`#1	*Navigation-Platform-installation-directory*`\pluginSDK\plugin\`*plugin-name*`\WEB-INF`	

#1

These files were shared by all plugins in old versions, but are no longer shared in Navigation Platform. Therefore, you need to place these files in all plugins individually.

#2

You also need to set the class path in the Option definition file for J2EE servers (`usrconf.cfg`).

3. For I/O Plugins, further overwrite the files listed below in the new template plugin files.

Table C–2: List of files for overwriting (I/O Plugin)

No.	Files of version 09-50 or later	Copy destination folder	Note
1	Files directly under*Navigation-Platform-installation-directory*`\pluginSDK\plugin\`*plugin-name*`\images`	*Navigation-Platform-installation-directory*`\pluginSDK\plugin\`*plugin-name*`\images`	Copy all the files even if there are more old version files than new template plugin files.

2	*Navigation-Platform-installation-directory*`\pluginSDK\plugin\`*plugin-name*`\WEB-INF\conf\ioaction.xml`	*Navigation-Platform-installation-directory*`\pluginSDK\plugin\`*plugin-name*`\WEB-INF\conf`	Apply the contents of the old version file by editing the file of the same name in the copy destination folder, rather than copying the old file to overwrite the new one.
			Note that the URL specified for the `iconURL` tag is different from that of old versions.
			The value specified for the `iconURL` tag in old versions
			`"/`*plugin-name*`/images/`*file-name*`"`
			The value specified for the new `iconURL` tag
			`"/ucnpPlugins/`*plugin-name*`/images/`*file-name*`"`
			• For migration from 09-60
			The button type specified for the `diableButtonType` tag allows you to distinguish between transfer by buttons and direct transfer. If you want the same operation as that in 09-60, specify `true` for the `legacy` attribute.

3	*Navigation-Platform-installation-directory*`\pluginSDK\plugin\`*plugin-name*`\WEB-INF\src\`*Java-package*`\controller\`*I/O-action-controller*`.java`	*Navigation-Platform-installation-directory*`\pluginSDK\plugin\`*plugin-name*`\WEB-INF\src\`*Java-package*`\controller`	Change the character code to `UTF-8` while retaining the file contents, and then overwrite the file.

4. For Suspend/Resume Plugins, further overwrite the files listed below in the new template plugin files.

Table C–3: List of files for overwriting (Suspend/Resume Plugin)

No.	File of version 09-50 or later	Copy destination folder	Note
1	*Navigation-Platform-installation-directory*`\pluginSDK\plugin\`*plugin-name*`\WEB-INF\src\`*Java-package*`\controller\`*I/O-action-controller*`.java`	*Navigation-Platform-installation-directory*`\pluginSDK\plugin\`*plugin-name*`\WEB-INF\src\`*Java-package*`\controller`	Change the character code to `UTF-8` while retaining the file contents, and then overwrite the file.

5. If the plugin uses a resource adapter, see *4.4 Adding database connection processing* and then add resource adapter settings.

6. Build, deploy, and then debug the plugin in the same way as for creating a new plugin.

- *4.6 Building plugins*
- *4.7 Deploying plugins*
- *4.10 Debugging plugins*

C.2 Migration of the menu area developed in old versions

In old versions, the menu area was able to be customized by using HTML or JSP files. However, the menu area in Navigation Platform cannot be customized.

C.3 Migration of custom windows (new windows) developed in old versions

In old versions, custom windows (new windows) were developed separately from plugins. In Navigation Platform, custom windows (new windows) must also be developed again as plugins.

In old versions, a custom window was required to use the search function. However, you do not need to develop a custom window for the search function in Navigation Platform because the search function is provided by default.

For details about how to develop plugins for custom windows, see *Chapter 4. Developing Plugins*.

C.4 J2EE server settings in old versions

If you have changed J2EE server settings in the development environment of an old version, return the settings to the default values, except for the items described in *4.13.1 J2EE server setting items that can be changed during plugin development*.

D. Reference Material for This Manual

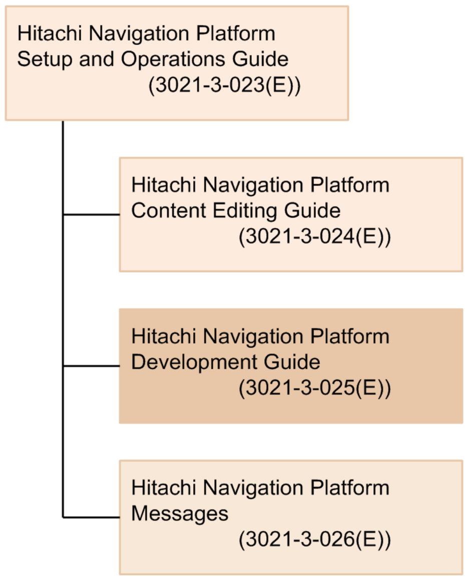
This appendix provides reference information, including various conventions, for this manual.

D.1 Related publications

This manual is part of a related set of manuals. The manuals in the set are listed below (with the manual numbers):

(1) Navigation Platform manuals

The Navigation Platform manuals are listed below. Note that in the texts of these manuals, references to these manuals omit the *Hitachi Navigation Platform* portion of the titles.


Hitachi Navigation Platform Setup and Operations Guide (3021-3-023(E))
Hitachi Navigation Platform Content Editing Guide (3021-3-024(E))
Hitachi Navigation Platform Development Guide (3021-3-025(E))
Hitachi Navigation Platform Messages (3021-3-026(E))

- *Hitachi Navigation Platform Setup and Operations Guide* (3021-3-023(E))

  This manual provides an overview of Navigation Platform and its functionality, and describes how to set up and operate a system that incorporates Navigation Platform.

  Users of Navigation Platform should read this manual first.
- *Hitachi Navigation Platform Content Editing Guide* (3021-3-024(E))

  This manual describes how to create, modify, and delete Operational Content, and how to manage the permissions that govern access to this content.
- *Hitachi Navigation Platform Development Guide* (3021-3-025(E))

  The manual you are reading. This manual describes how to develop plugins and custom windows for Navigation Platform.
- *Hitachi Navigation Platform Messages* (3021-3-026(E))

  This manual explains the messages output when using Navigation Platform.

(2) Manuals for related software

Manuals related to Navigation Platform are listed below.

- *uCosminexus Application Server Command Reference Guide* (3020-3-Y15(E))

- *uCosminexus Application Server Definition Reference Guide* (3020-3-Y16(E))
- *uCosminexus Application Server Application and Resource Definition Reference Guide* (3020-3-Y17(E))

D.2 Conventions: Abbreviations for product names

This manual uses the following abbreviations for product names and related software names:

Abbreviation	Full name or meaning
Cosminexus#	uCosminexus Primary Server Base
Eclipse	Eclipse Web Tools Platform

#

In descriptions of uCosminexus Navigation Platform, the term *Cosminexus* is sometimes used generically to refer to uCosminexus Application Server and uCosminexus Developer.

D.3 Conventions: Acronyms

This manual also uses the following acronyms:

Acronym	Full name or meaning
EAR	Enterprise ARchive
ISO	International Organization for Standardization
J2EE	Java 2 Platform, Enterprise Edition
JavaVM	Java Virtual Machine
UTF	UCS Transformation Format
WAR	Web ARchive
XML	eXtensible Markup Language

D.4 Conventions: KB, MB, GB, and TB

This manual uses the following conventions:

- 1 KB (kilobyte) is 1,024 bytes.
- 1 MB (megabyte) is $1{,}024^2$ bytes
- 1 GB (gigabyte) is $1{,}024^3$ bytes.
- 1 TB (terabyte) is $1{,}024^4$ bytes.

E. Glossary

application server

A server used to process user operations from the Web browser and apply the results to a business system such as a database.

Branch Node

A node required to create an Operational Flow with branching steps. A Branch Node is represented by a diamond in Navigation Platform windows.

cache

Data of the values input or set for Guides of Operational Content by the user. By using the cache, plugins can save the values in the middle of input and pass the input values to other windows or systems.

Content Editor

A user who has permission to access Operational Content as an editor.

Content Manager

A user who has permission to access Operational Content as a manager.

Custom Window Plugin

A plugin used to develop windows other than the windows of Navigation Platform. A window developed by this plugin is called a *custom window*.

development environment

An environment in which plugins are developed. You must have Hitachi Navigation Platform for Developers or JP1/Navigation Platform for Developers to use a development environment.

editing environment

An environment in which you can create and edit Operational Content, and assign access permission for Operational Content. You must have Hitachi Navigation Platform or JP1/Integrated Management - Navigation Platform to use the editing environment.

execution environment

The environment accessed by users who use Operational Content to perform an operation. You must have Hitachi Navigation Platform or JP1/Integrated Management - Navigation Platform to use the execution environment.

export

To output the configuration information of Operational Content and access permissions in ZIP format. Exported data can be imported to another environment.

Guide

A type of Operational Content. A Guide is a description of the individual tasks performed as part of an operation.

Guide Part

A component used to create Guides. You can create a Guide by selecting Guide Parts from the **Guide** pallette and placing them in the guide area.

import

To load the exported information into the local environment.

You can import the configuration information of Operational Content and access permissions.

I/O Plugin

A plugin that processes the information input in the Navigation Platform windows, and determines what information to display in the windows. By using I/O Plugins, you can check the integrity of values input into a Guide, output the input values to external programs, or prepare them for recording as log data.

I/O plugin XML file

A file that defies information (input and output parameters) required to associate I/O Plugins with Operational Content. Information defined in this file is displayed in Plugin Parts in the **Plugins** pallette.

In addition, icons in the **Plugins** pallette displayed in the Operational Content Editing Window can be set in any file.

mapping line

An arrow that represents the input or output relationship between Guide Parts and Plugin Parts.

Navigation Platform

A collective term for Hitachi Navigation Platform, JP1/Integrated Management - Navigation Platform, Hitachi Navigation Platform for Developers, and JP1/Navigation Platform for Developers.

node

A component displayed in an Operational Flow that represents a step in the operation. There are three types of node: Terminal Nodes, Process Nodes, and Branch Nodes.

Operational Content

The information required to perform an operation. You can create and display Operational Content in the windows of Navigation Platform.

Operational Content Editing Window

A window of Navigation Platform in which you can perform the following operations:

- Create, modify, copy, and delete Operational Content
- Publish and unpublish Operational Content
- Associate I/O Plugins
- Set access permissions for Operational Content

Operational Flow

A type of Operational Content. An Operational Flow is the flow of an operation presented as a flow chart.

part
A component placed in the Navigation Platform window.

plugin
A user program developed for use with Navigation Platform. Its purpose might be to link Navigation Platform with an external program, or to use certain extended functionality of Navigation Platform. Navigation Platform provides the interfaces required to develop plugins.

plugin information property file
A file that defines information required to create template plugins.

Plugin Part
A component that associates plugins with Operational Content. You can select Plugin Parts from the **Plugins** pallette and place them in the guide area.

Process Node
A node that represents an intermediate step in an operation. There must be at least one Process Node between Terminal Nodes. A Process Node is represented by a rectangle in Navigation Platform windows.

relation line
An arrow that shows the transition from one node to another.

Suspend/Resume Plugin
A plugin required to allow a user to resume the operation from the temporarily saved status even if the user references information about other operations or logs out in the middle of operation.

suspend information
Information that is temporarily saved when the user suspends window operation in an environment in which Suspend/Resume Plugins are used. Suspend information is recovered when the user resumes the window operation.

System Plugin
A plugin provided by Navigation Platform that is required to link with external programs or between Guide Parts.

template plugin
A file from which plugins are created. You can create plugins by editing a template plugin.

Template plugins are created in the format of an Eclipse Java project.

Terminal Node
A node that represents the start or end of an operation. A Terminal Node is placed at the beginning and end of an Operational Flow. A Terminal Node is represented by a circle in Navigation Platform windows.

User Plugin
A plugin developed by a developer using the API provided by Navigation Platform.

Web server
A server that performs processing relating to receiving requests from and sending data to the Web browser.

window ID

An ID that uniquely identifies the window in which an operation of Operational Content is being performed.

Index