

**Job Management Partner 1 Version 10**

**Job Management Partner 1/Automatic Operation  
Service Template Developer's Guide**

---

**3021-3-363-10(E)**

---

## Notices

### ■ Relevant program products

P-242C-E1AL Job Management Partner 1/Automatic Operation 10-50 (for Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2)

The above product includes the following:

- P-CC242C-EAAL Job Management Partner 1/Automatic Operation - Server 10-50 (for Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2)
- P-CC242C-EBAL Job Management Partner 1/Automatic Operation - Contents 10-50 (for Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2)

P-F242C-E1AL1 Job Management Partner 1/Automatic Operation Contents Set 10-50 (for Windows Server 2008 R2, Windows Server 2012, Windows Server 2012 R2)

### ■ Trademarks

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

Adobe and Flash Player are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

HP-UX is a product name of Hewlett-Packard Development Company, L.P. in the U.S. and other countries.

IBM, AIX are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide.

Intel is a trademark of Intel Corporation in the U.S. and/or other countries.

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

Itanium is a trademark of Intel Corporation in the United States and other countries.

Kerberos is a name of network authentication protocol created by Massachusetts Institute of Technology.

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

Microsoft and Hyper-V are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Microsoft .NET is software for connecting people, information, systems, and devices.

Microsoft and SQL Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Netscape is a trademark of AOL Inc. in the U.S. and other countries.

The OpenStack(R) Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

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

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

RSA and BSAFE are either registered trademarks or trademarks of EMC Corporation in the United States and/or other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc., in the United States and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

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

VMware and vCenter Server are registered trademarks or trademarks of VMware, Inc. in the United States and/or other jurisdictions.

VMware and VMware vSphere ESX are registered trademarks or trademarks of VMware, Inc. in the United States and/or other jurisdictions.

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

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

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

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

This product includes software developed by Andy Clark.

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

This product includes software developed by Ben Laurie for use in the Apache-SSL HTTP server project.

This product includes software developed by Daisuke Okajima and Kohsuke Kawaguchi (<http://relaxngcc.sf.net/>).

This product includes software developed by IAIK of Graz University of Technology.

This product includes software developed by the Java Apache Project for use in the Apache JServ servlet engine project (<http://java.apache.org/>).

This product includes software developed by Ralf S. Engelschall <[rse@engelschall.com](mailto:rse@engelschall.com)> for use in the mod\_ssl project (<http://www.modssl.org/>).

Portions of this software were developed at the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign.

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).

Regular expression support is provided by the PCRE library package, which is open source software, written by Philip Hazel, and copyright by the University of Cambridge, England. The original software is available from <ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/>



Job Management Partner 1/Automatic Operation includes RSA BSAFE(R) Cryptographic software of EMC Corporation.

**HITACHI**  
Inspire the Next

 Hitachi, Ltd.





■ **Issued**

Dec. 2014: 3021-3-363-10(E)

■ **Copyright**

All Rights Reserved. Copyright (C) 2012, 2014, Hitachi, Ltd.

## Summary of amendments

The following table lists changes in this manual (3021-3-363-10(E)) and product changes related to this manual.

Changes	Location
<p>For the manual issued in December 2014 or later, the title and reference number were changed as shown below.</p> <p>Before the change:</p> <p><i>Job Management Partner 1/Automatic Operation GUI and Command Reference (3021-3-315(E))</i></p> <p>After the change:</p> <p><i>Job Management Partner 1/Automatic Operation GUI, Command, and API Reference (3021-3-366(E))</i></p>	--
Windows Server 2012 R2 was added as a supported operating system.	--
Functionality was added that facilitates the debugging of service templates. Changes were made to the manual structure to accommodate this new functionality.	1.1, 1.2, 1.3, 1.4, 1.6, 5., 6., A.1
A description was added regarding the use of the content plug-ins provided by JP1/AO in service template development.	1.1.1
A function was added that allows users to be promoted to root privilege when executing a content plug-in. This function can be used when the OS of the operation target device is UNIX.	4.1.3
A description was added regarding the conditions under which files can be transferred.	4.1.5, 4.1.6
The folder in which transferred files are stored can now be set in the property file (config_user.properties).	4.1.6
A description of the setting in the plugin.suPassword reserved plug-in property when a plug-in is executed without promoting user permission to root was added.	4.3.6
Public key authentication was added as an authentication method for operation target devices.	2.2.13, 4.3.4, 4.3.6, 4.3.11
Release plug-ins can now be deleted.	2.5.1, 4.1.4, 4.4.1
<p>A description of reserved properties was added. Also, the following reserved properties were added:</p> <ul style="list-style-type: none"> <li>• reserved.loop.index</li> <li>• reserved.service.category</li> <li>• reserved.service.name</li> <li>• reserved.service.resourceGroupName</li> <li>• reserved.step.path</li> <li>• reserved.step.prevReturnCode</li> <li>• reserved.task.description</li> <li>• reserved.task.id</li> <li>• reserved.task.name</li> </ul>	3.2.7

Changes	Location
<ul style="list-style-type: none"> <li>reserved.task.submitter</li> <li>reserved.task.url</li> </ul>	3.2.7
For the reserved.step.prevReturnCode reserved property, a description of a scenario in which the preceding step is not executed when trying a task again was added.	3.2.7
Subsequent-step execution conditions were added to the information inherited when pasting a step or relational line.	3.3.7
In addition to Windows and Linux, content-plug-ins that execute commands and scripts in AIX, HP-UX, and Solaris are now supported.	4.1.2, 4.1.6, 4.3.6, 4.3.9, 4.3.10, 4.3.14, 4.3.15, 4.3.16
The description of the execution user for commands and scripts executed using content plug-ins was clarified.	4.1.3
An explanation that user profiles are not inherited when the OS of the operation target device is Windows was added.	4.1.3
The manual now mentions that certain commands must be installed in the OS of the operation target device before executing a content plug-in.	4.1.7
The ibm-943 character set used for communication by JP1/AO during plug-in execution was changed to ibm-943C.	4.1.8
The manual now mentions that certain commands must be installed in the OS of the operation target device before executing a plug-in.	4.1.10
A description of executing a non-standard script in the <b>Command line</b> text box was added.	4.3.10
A cautionary note about specifying the command line for a content plug-in was added.	4.3.10
A section on the standard output of plug-ins was added.	4.3.14
A description of the return values of content plug-ins was added.	4.3.11
A description of the relationship among the return values of executed commands or scripts, plug-ins, and steps was added.	4.3.12
A procedure for using a return value as the branching condition for a flow was added, for situations when a command or script executed as a plug-in returns codes outside the 0 to 63 range.	4.3.13
The manual now instructs users not to enclose the path of the execution directory in double or single-quotation marks, even if the path contains spaces.	4.3.16

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

# Preface

This manual describes how to develop the service templates and plug-ins used by Job Management Partner 1/Automatic Operation.

In this manual, Job Management Partner 1 is abbreviated to *JP1*, and Job Management Partner 1/Automatic Operation is abbreviated to *JP1/AO*.

For reference information on JP1/AO manuals and a glossary, see the manual *Job Management Partner 1/Automatic Operation Overview and System Design Guide*.

## ■ Intended readers

This manual is intended for the following readers:

- Users who create new service templates
- Users who edit service templates

Readers of this document are assumed to have a basic understanding of JP1/AO.

## ■ Microsoft product name abbreviations

This manual uses the following abbreviations for Microsoft product names.

Abbreviation		Full name or meaning	
.NET Framework	.NET Framework 3.5	Microsoft(R) .NET Framework 3.5	
Active Directory		Microsoft(R) Active Directory	
Hyper-V		Microsoft(R) Hyper-V(R)	
Internet Explorer	Microsoft Internet Explorer	Microsoft(R) Internet Explorer(R)	
	Windows Internet Explorer	Windows(R) Internet Explorer(R)	
Windows	Windows 7	Microsoft(R) Windows(R) 7 Enterprise	
		Microsoft(R) Windows(R) 7 Professional	
		Microsoft(R) Windows(R) 7 Ultimate	
	Windows Server 2003 <sup>#1</sup>	Windows Server 2003 <sup>#1</sup>	Microsoft(R) Windows Server(R) 2003, Enterprise Edition
			Microsoft(R) Windows Server(R) 2003, Standard Edition
		Windows Server 2003 (x64)	Microsoft(R) Windows Server(R) 2003, Enterprise x64 Edition
			Microsoft(R) Windows Server(R) 2003, Standard x64 Edition
Windows Server 2003 R2 <sup>#2</sup>	Windows Server 2003 R2 <sup>#2</sup>	Microsoft(R) Windows Server(R) 2003 R2, Enterprise Edition	

Abbreviation				Full name or meaning	
Windows	Windows Server 2003 <sup>#1</sup>	Windows Server 2003 R2 <sup>#2</sup>	Windows Server 2003 R2 <sup>#2</sup>	Microsoft(R) Windows Server(R) 2003 R2, Standard Edition	
			Windows Server 2003 R2 (x64)	Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition	
				Microsoft(R) Windows Server(R) 2003 R2, Standard x64 Edition	
	Windows Server 2008	Windows Server 2008 R2	Windows Server 2008 R2 Datacenter	Microsoft(R) Windows Server(R) 2008 R2 Datacenter	
			Windows Server 2008 R2 Enterprise	Microsoft(R) Windows Server(R) 2008 R2 Enterprise	
			Windows Server 2008 R2 Standard	Microsoft(R) Windows Server(R) 2008 R2 Standard	
		Windows Server 2008 x64	Windows Server 2008 Datacenter x64	Microsoft(R) Windows Server(R) 2008 Datacenter x64	
			Windows Server 2008 Enterprise x64	Microsoft(R) Windows Server(R) 2008 Enterprise x64	
			Windows Server 2008 Standard x64	Microsoft(R) Windows Server(R) 2008 Standard x64	
		Windows Server 2008 x86	Windows Server 2008 Datacenter x86	Microsoft(R) Windows Server(R) 2008 Datacenter x86	
			Windows Server 2008 Enterprise x86	Microsoft(R) Windows Server(R) 2008 Enterprise x86	
			Windows Server 2008 Standard x86	Microsoft(R) Windows Server(R) 2008 Standard x86	
		Windows Server 2012	Windows Server 2012 Datacenter		Microsoft(R) Windows Server(R) 2012 Datacenter
			Windows Server 2012 Standard		Microsoft(R) Windows Server(R) 2012 Standard
		Windows Server 2012 R2	Windows Server 2012 R2 Datacenter		Microsoft(R) Windows Server(R) 2012 R2 Datacenter
	Windows Server 2012 R2 Standard		Microsoft(R) Windows Server(R) 2012 R2 Standard		
	Windows Server Failover Cluster			Microsoft(R) Windows Server(R) Failover Cluster	
	Windows Vista			Microsoft(R) Windows Vista(R) Business	
				Microsoft(R) Windows Vista(R) Enterprise	
				Microsoft(R) Windows Vista(R) Ultimate	
Windows XP			Microsoft(R) Windows(R) XP Professional Operating System		

#### #1

In descriptions, if Windows Server 2003 (x64) or Windows Server 2003 R2 is noted alongside Windows Server 2003, the description for Windows Server 2003 does not apply to Windows Server 2003 (x64) or Windows Server 2003 R2.



In descriptions, if Windows Server 2003 R2 (x64) is noted alongside Windows Server 2003 R2, the description for Windows Server 2003 R2 does not apply to Windows Server 2003 R2 (x64).

## ■ Formatting conventions used in this manual

The following describes the formatting conventions used in this manual.

Text formatting	Description
<i>Character string</i>	Italic characters indicate a variable. Example: A date is specified in <i>YYYYMMDD</i> format.
<b>Bold - Bold</b>	Indicates selecting menu items in succession. Example: Select <b>File</b> - <b>New</b> . This example means that you select <b>New</b> from the <b>File</b> menu.
<b>key + key</b>	Indicates pressing keys on the keyboard at the same time. Example: <b>Ctrl</b> + <b>Alt</b> + <b>Delete</b> means pressing the <b>Ctrl</b> , <b>Alt</b> , and <b>Delete</b> keys at the same time.

### Representation of JP1/AO-related installation folders

In this manual, the default installation folders are represented as follows:

JP1/AO installation folder:

*system-drive*\Program Files (x86)\Hitachi\JP1AO

Common Component installation folder:

*system-drive*\Program Files (x86)\Hitachi\HiCommand\Base

# Contents

Notices 2

Summary of amendments 5

Preface 7

## 1 Flow of Service Template Development 16

1.1 Overview 17

1.1.1 Flow of service template development 17

1.1.2 Elements involved in service template development 19

1.1.3 Main windows used to develop service templates 22

1.2 Tasks associated with service templates 25

1.2.1 Tasks performed when creating new service templates 25

1.2.2 Tasks performed when editing service templates 26

1.2.3 Tasks performed when using existing service templates as-is 26

1.3 General procedure for creating new service templates 28

1.3.1 General procedure for creating new service templates 28

1.4 General procedure when editing an existing service template 30

1.4.1 General procedure when editing service template definition information 31

1.4.2 General procedure for editing a plug-in and applying the result to a service template 32

1.4.3 General procedure for creating new plug-ins and adding them to service templates 33

1.4.4 General procedure for changing the description displayed for a service in the user interface 35

1.4.5 General procedure for adding processing to a service template 35

1.4.6 General procedure for deleting processing from a service template 37

1.4.7 General procedure for dynamically or statically setting parameters during automated processing 38

1.4.8 General procedure for using a service resource file to set the information displayed for a service 39

1.4.9 General procedure for using a plug-in resource file to set display information for a plug-in 40

1.5 Using existing service templates provided by JP1/AO 41

1.5.1 General procedure for using an existing service template provided by JP1/AO 41

1.6 List of service template development features 42

## 2 Setting service template definition information 44

2.1 Displaying the **Editor** window 45

2.1.1 Procedure for displaying the **Editor** window 45

2.1.2 Overview of development service templates and release service templates 46

2.1.3 Available operations by service template configuration type 46

2.1.4 Behavior when an intervening action occurs in the **Editor** window 47

2.1.5 Version compatibility for service templates and plug-ins created in the **Editor** window 48

2.1.6 Compatibility with steps in service templates created in earlier versions of JP1/AO 48

2.2	Creating and editing service templates	50
2.2.1	Creating blank service templates	50
2.2.2	Parameters to set when creating or copying service templates	50
2.2.3	Procedure for setting service definition information	51
2.2.4	Parameters to set in service definition information	51
2.2.5	Example of mapping parameter definition and flow of data	52
2.2.6	Procedure for setting custom files	54
2.2.7	Overview of custom files	55
2.2.8	Format of custom files	56
2.2.9	Properties defined in services (service properties)	57
2.2.10	Adding Service Share Properties	58
2.2.11	Overview of Service Share Properties	58
2.2.12	Notes on defining Service Share Properties	59
2.2.13	Overview of shared built-in service properties	60
2.2.14	Property visibility	62
2.2.15	Viewing another service template during editing	64
2.3	Displaying a list of service templates	65
2.3.1	Procedure for displaying a list of service templates	65
2.4	Copying service templates	66
2.4.1	Procedure for copying service templates	66
2.4.2	Uniqueness of service templates and plug-ins	66
2.5	Deleting service templates	68
2.5.1	Procedure for deleting development service templates	68
2.5.2	Procedure for deleting release service templates	68
2.6	Setting display information for service templates in resource files	69
2.6.1	Procedure for setting service resource files	69
2.6.2	Format of service resource file	69
2.6.3	Definitions in service resource files	70
2.6.4	Correspondence between information displayed in service templates and properties in service resource files	71
2.6.5	Service resource files automatically generated when a service template is created	72
2.6.6	Service resource files updated when a service template is edited	73
2.6.7	Displaying a service template in a Web browser that is set to a locale for which no service resource file is available	74

### **3 Creating and editing flows 75**

3.1	Displaying the <b>Flow</b> view	76
3.1.1	Procedure for displaying the <b>Flow</b> view	76
3.1.2	Relationship between flow and steps	76
3.1.3	Creating flow hierarchies	77
3.2	Adding steps	78
3.2.1	Procedure for adding steps	78

3.2.2	Procedure for changing step definition information	78
3.2.3	Settings in step definition information	79
3.2.4	Overview of subsequent step conditions	80
3.2.5	Procedure for setting input property mapping	81
3.2.6	Procedure for setting output property mapping	82
3.2.7	List of reserved properties	82
3.2.8	Warning icon displayed for steps	85
3.3	Connecting steps with relational lines	86
3.3.1	Procedure for connecting steps with relational lines	86
3.3.2	Procedure for deleting steps and relational lines	86
3.3.3	Procedure for copying steps and relational lines	87
3.3.4	Procedure for cutting steps and relational lines	87
3.3.5	Procedure for selecting multiple steps	88
3.3.6	Procedure for pasting steps and relational lines	88
3.3.7	Information inherited when pasting steps or relational lines	88
3.3.8	Behavior when relational lines connect to multiple steps	89
3.3.9	Scenarios where relational lines cannot be drawn	90
3.3.10	Drawing relational lines when processing branches	91
<b>4</b>	<b>Creating and editing plug-ins</b>	<b>92</b>
4.1	Displaying a list of plug-ins	93
4.1.1	Procedure for displaying a list of plug-ins	93
4.1.2	Overview of basic plug-ins, release plug-ins, and development plug-ins	93
4.1.3	Plug-in executing users	94
4.1.4	Available operations by plug-in type	96
4.1.5	Files transferred to Windows systems	96
4.1.6	Files transferred to UNIX systems	97
4.1.7	Locale set for operation target devices during plug-in execution	97
4.1.8	Character set used for communication by JP1/AO during plug-in execution	98
4.1.9	Setting a specific character set during plug-in execution	99
4.1.10	Commands required for plug-in execution	99
4.2	Creating plug-ins	100
4.2.1	Procedure for creating plug-ins	100
4.2.2	Parameters to set when creating or copying plug-ins	100
4.3	Editing plug-ins	102
4.3.1	Procedure for editing plug-in definition information	102
4.3.2	Parameters to set in plug-in definition information	102
4.3.3	Image files usable as plug-in icons	103
4.3.4	plug-in credential types	103
4.3.5	Properties defined in plug-ins (plug-in properties)	104
4.3.6	Reserved plug-in properties for specifying execution-target hosts and credential information	104
4.3.7	Procedure for mapping standard output and standard error output to output properties	106

4.3.8	Specifying output filters	106
4.3.9	Procedure for setting scripts	107
4.3.10	Specifying commands in the <b>Command line</b> text box	109
4.3.11	Return values of content plug-ins	110
4.3.12	Relationship of command and script return values to the return values of plug-ins and steps	112
4.3.13	Procedure for using the return value of a command or script as a flow branching condition (for values outside the 0 to 63 range)	112
4.3.14	Information output to standard output by plug-ins	112
4.3.15	Differences between script settings methods	113
4.3.16	Specifying <b>Execution Directory</b>	114
4.3.17	Procedure for setting commands	115
4.4	Deleting plug-ins	116
4.4.1	Procedure for deleting plug-ins	116
4.5	Copying plug-ins	117
4.5.1	Procedure for copying plug-ins	117
4.6	Using resource files to set plug-in display information	118
4.6.1	Procedure for setting plug-in resource files	118
4.6.2	Format of plug-in resource files	118
4.6.3	Correspondence between properties in plug-in resource files and information displayed for plug-ins	119
4.6.4	Plug-in resource files automatically generated when a plug-in is created	120
4.6.5	Plug-in resource files updated when a plug-in is edited	121
4.6.6	Displaying a plug-in on a Web browser set to a locale for which no plug-in resource file is available	121
<b>5</b>	<b>Validating Service Templates</b>	<b>122</b>
5.1	Overview of service template validation	123
5.1.1	Flow of service template validation	123
5.1.2	Overview of building	124
5.1.3	Overview of debugging	125
5.1.4	Overview of operation tests	126
5.2	Building service templates	128
5.2.1	Procedure for building a service template	128
5.3	Debugging service templates	129
5.3.1	Flow of service template debugging	129
5.3.2	Functions used during debug operations	129
5.3.3	Example of service template debugging	131
5.3.4	Procedure for starting the debug process	132
5.3.5	Settings used when beginning the debug process	134
5.3.6	Procedure for debugging a service template again without rebuilding	134
5.3.7	Flow of debug process without pausing between steps	135
5.3.8	Flow of debug process when pausing between steps	136
5.3.9	Processing of debug process when pausing between steps	137

5.3.10	Plug-ins that cannot be paused during debugging	139
5.3.11	Step information that can be changed while step execution is paused	140
5.3.12	Procedure for skipping plug-in processing during the debug process	140
5.3.13	Procedure for retrying a task from a failed step during debugging	141
5.3.14	Procedure for retrying a task from the step after the failed step during debugging	141
5.3.15	Handling debug tasks that are waiting for a response (response entry)	142
5.3.16	Procedure for checking property mapping settings during debugging	142
5.3.17	Procedure for changing the value of a plug-in input property during debugging	144
5.3.18	Procedure for changing the value of a plug-in output property during debugging	145
5.3.19	Procedure for changing plug-in return values during debugging	146
5.3.20	Displaying the values of plug-in properties during debugging	146
5.3.21	Effect of changing the values of plug-in properties during debugging	147
5.3.22	When the value of a plug-in property includes surrogate pair characters or control characters	148
5.3.23	Linefeed codes in values of plug-in properties (when step execution is paused)	148
5.3.24	Displaying the flow of a debug task	149
5.3.25	Displaying the flow tree of a debug task	150
5.3.26	Displaying a repeated execution flow during debugging	151
5.3.27	Flow tree view for repeated flows during debugging	151
5.3.28	Information displayed for repeated execution plug-ins and repeated flows during debugging	152
5.4	Managing debug tasks	154
5.4.1	Procedure for checking the status of all debug tasks	154
5.4.2	Procedure for checking the progress of debug tasks from the <b>Tasks</b> window	154
5.4.3	Procedure for checking detailed progress of debug tasks in list format	155
5.4.4	Procedure for checking task log entries for debug tasks	155
5.4.5	Procedure for stopping debug tasks	156
5.4.6	Procedure for forcibly stopping debug tasks	157
5.4.7	Procedure for deleting debug tasks	158
5.5	Testing service templates	160
5.5.1	Procedure for testing service templates	160
<b>6</b>	<b>Releasing Service Templates</b>	<b>161</b>
6.1	Releasing service templates	162
6.1.1	Procedure for releasing a service template	162
6.2	Overview of service template release	163
6.2.1	Overview of service template release	163
6.3	Importing service templates into the active environment (when the active and development environments are separate)	164
6.3.1	Procedure for importing service templates into the active environment (when the active and development environments are separate)	164
6.3.2	Reason for maintaining separate development and active environments	164

## **Appendix 166**

A Reference Information 167

A.1 Reference information for build and release operations 167

## **Index 172**

# 1

## Flow of Service Template Development

This chapter provides a general overview of service template development. Service templates are used to define processing that automates the operating procedures in an IT system.



## 1.1 Overview

---

### 1.1.1 Flow of service template development

In JP1/AO, you can use service templates to automate operating procedures. This functionality is particularly effective when applied to the automation of complex operating procedures, or those that are executed often but at random times.

You can create new service templates. You can also use the existing templates provided by JP1/AO<sup>#1</sup> without modification, or copy an existing template and edit it<sup>#2</sup> by adding and removing steps as needed.

#1:

Service templates provided by JP1/AO include the service templates provided with the JP1/AO standard package and the JP1/AO Content Set (available separately).

#2:

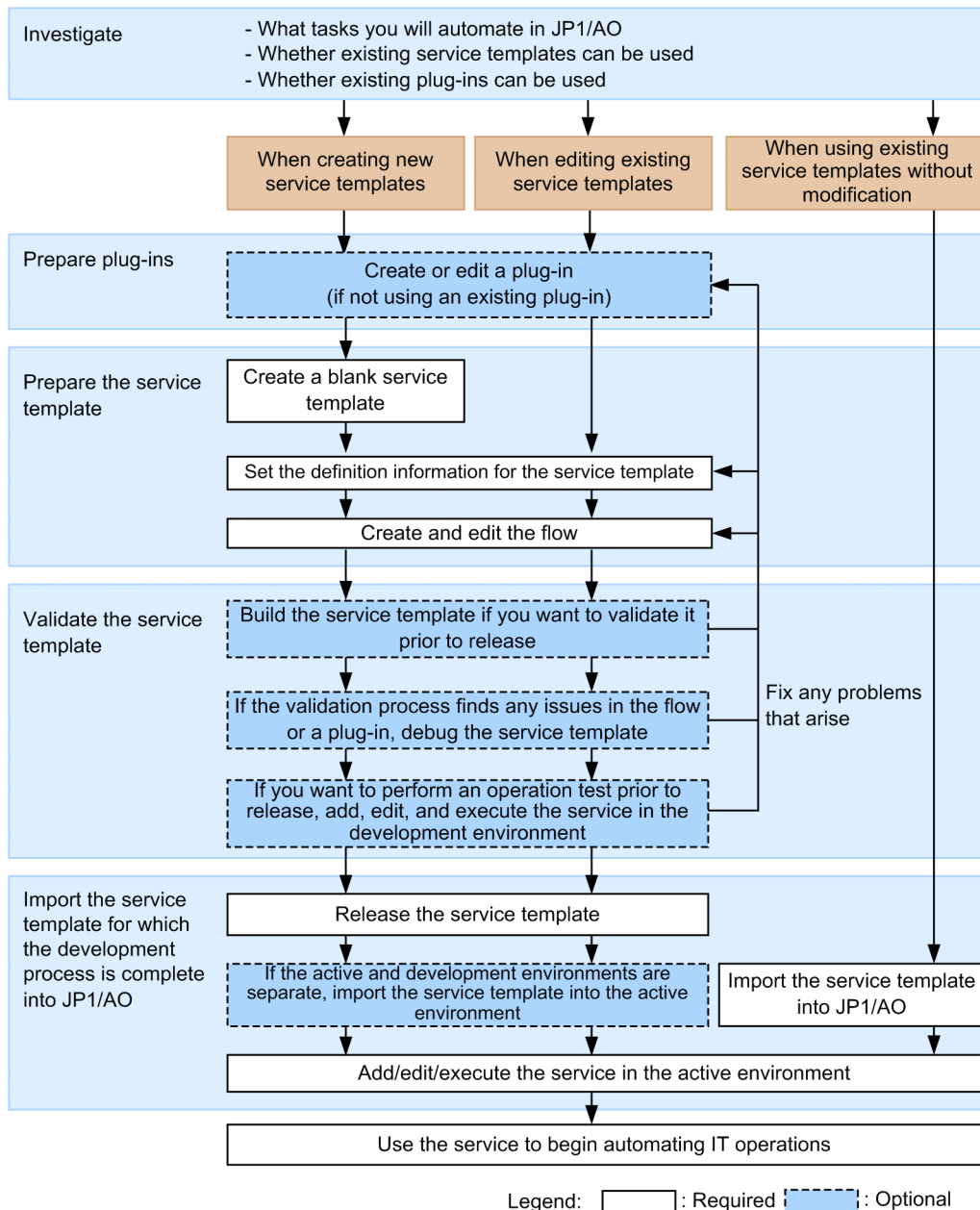
Before you can edit a service template provided by JP1/AO, you need to import the service template.

#### Tip

If you want to use a content plug-in provided by JP1/AO as the basis for service template development, either import the service template that contains the content plug-in or import the Utility Components service template. Importing the Utility Components service template imports every content plug-in in the JP1/AO standard package, making the JP1/AO Standard-package content plug-ins available for development purposes.

The following figure shows the general procedure for developing service templates.

Figure 1–1: Flow of service template development



### Related topics for creating new service templates

- 1.1.2 Elements involved in service template development
- 1.1.3 Main windows used to develop service templates
- 1.2.1 Tasks performed when creating new service templates
- 1.3 General procedure for creating new service templates

### Related topics for editing service templates

- 1.1.2 Elements involved in service template development
- 1.1.3 Main windows used to develop service templates
- 1.2.2 Tasks performed when editing service templates
- 1.4 General procedure when editing an existing service template

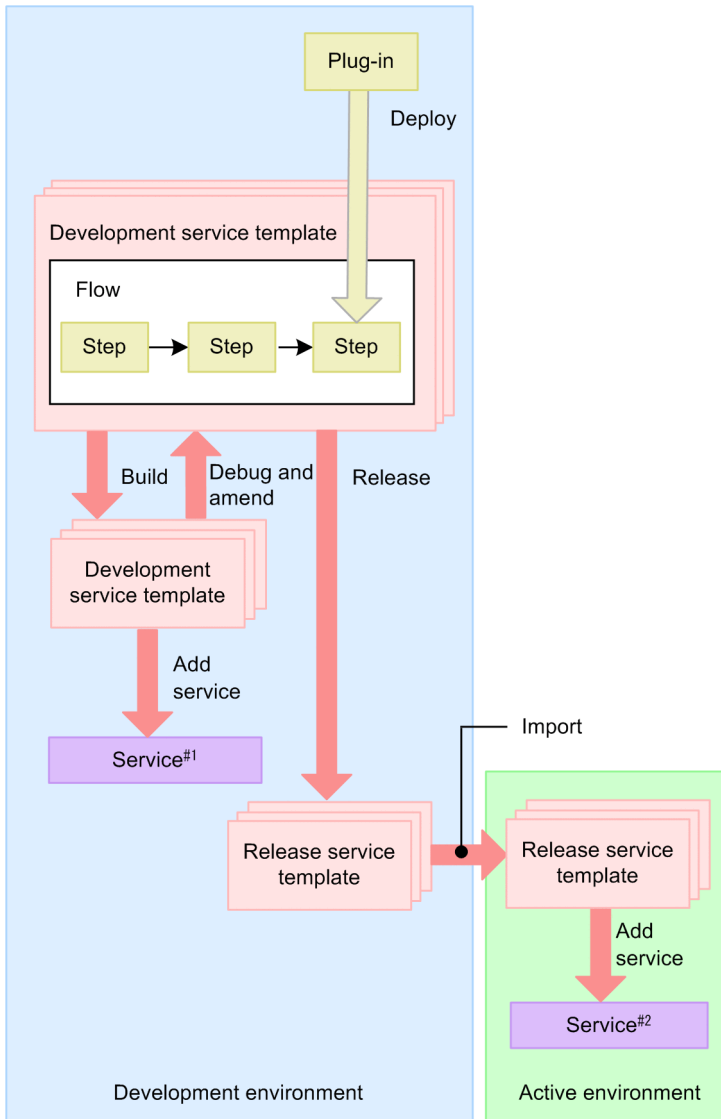
## Related topics for using unmodified service templates

- 1.1.2 Elements involved in service template development
- 1.2.3 Tasks performed when using existing service templates as-is
- 1.5 Using existing service templates provided by JP1/AO

### 1.1.2 Elements involved in service template development

Service templates define the information necessary to automate operating procedures. The following figure shows the elements involved in service template development:

Figure 1–2: Elements Involved in service template development



- #1 A service used to test the service template in the development environment.
- #2 The service executed in the active environment.

- Development environment

The environment in which the service template is developed. You can also conduct debugging and operation testing in this environment, to validate the operation of the service templates you develop. Although you can develop service templates in an active environment, we recommend that you keep the development and active environments separate.

- Active environment

The environment in which you can add and execute services based on service templates you have developed. The actual automation of operating procedures takes place in this environment.

- Plug-in

The smallest unit of processing you can define when automating IT operations.

- Service template

Defines the processing that automates the operating procedures in an IT system. A service template incorporates flows and steps.

- Development service template

A service template that is being developed by a user. Service templates created by copying a release service template are also classified as development service templates. Development service templates are used in the development environment.

- Release service template

When you release a development service template, it becomes a release service template. You cannot edit a release service template. The service templates provided by JP1/AO are also classified as release service templates. Release service templates are used in the active environment.

- Flow

Defines the flow of the operating procedure you are automating.

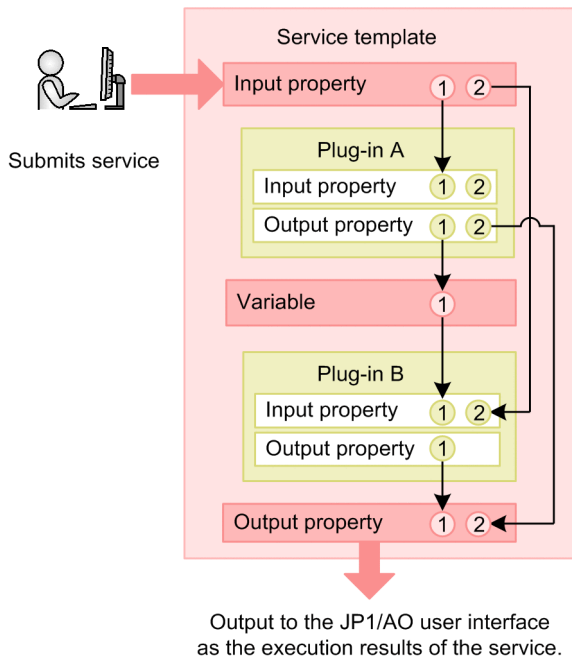
- Step

An element of a flow, each step executes a plug-in.

The following describes how property mapping takes place for service templates and plug-ins:

The following figure shows an example of property mapping.

Figure 1–3: Mapping service template and plug-in component properties



Legend:  $\longrightarrow$  : Property mapping

- Property mapping

A service template defines a generic operating procedure. For this reason, properties that store the input values required to execute the service, such as host names and resource limits, are defined when services are added from a service template. These are called *service input properties*. The execution results of a service are output to the JP1/AO user interface as the values of service output properties.

In plug-ins, input properties that store the input values required for plug-in execution and output properties that store execution results are defined. You can enter values into plug-in input properties directly, or pass values to them by linking them to a service input property or variable.

By linking a service output property to a plug-in output property, you can review the execution results of a plug-in in the JP1/AO user interface. Linking properties in this way and passing values between them is called *property mapping*.

- Mapping service input properties and plug-in input properties

In the example in [Figure 1–3: Mapping service template and plug-in component properties](#), Input property 1 of the service is mapped to Input property 1 of Plug-in A, and Input property 2 of the service is mapped to Input property 2 of Plug-in B. Mapping is not configured for Input property 2 of Plug-in A.

- The value input to Input property 1 of the service is input to Input property 1 of Plug-in A.
- The value input to Input property 2 of the service is input to Input property 2 of Plug-in B.
- The unmapped Input property 2 of Plug-in A is assigned the value entered when the service template was created or edited.

- Mapping service output properties and plug-in output properties

In the example in [Figure 1–3: Mapping service template and plug-in component properties](#), Output property 2 of Plug-in A is mapped to Output property 2 of the service, and Output property 1 of Plug-in B is mapped to Output property 1 of the service.

- The execution results of Plug-in A (command standard output and standard error output, and output properties) output as Output property 2 of Plug-in A are also output to Output property 2 of the service. The user can then review the execution results of Plug-in A in the JP1/AO user interface.

- The execution results of Plug-in B (command standard output and standard error output, and output properties) output as Output property 1 of Plug-in B are also output to Output property 1 of the service. The user can then review the execution results of Plug-in B in the JP1/AO user interface.
- Mapping variables to plug-in properties

In [Figure 1–3: Mapping service template and plug-in component properties](#), Output property 1 of Plug-in A is mapped to Variable 1, and Variable 1 is mapped to Input property 1 of Plug-in B.

Values output as Output property 1 of Plug-in A are stored in Variable 1 then input to Input property 1 of Plug-in B. This passes the execution results of Plug-in A to an input property of Plug-in B, allowing it to be used in the processing of Plug-in B.

### 1.1.3 Main windows used to develop service templates

The following describes the window in which most activity related to service template development takes place.

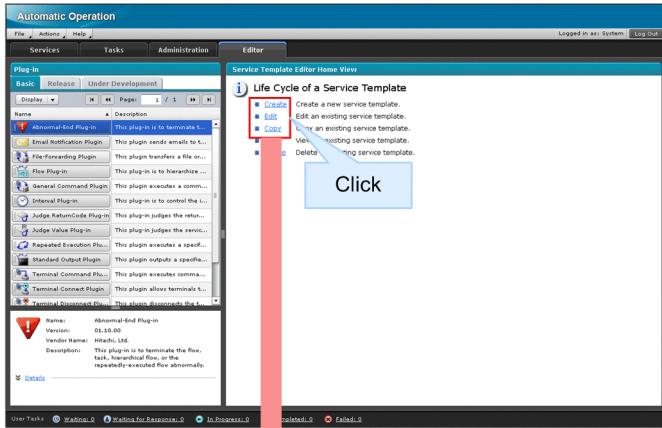
You can display the **Editor** window by clicking the **Editor** tab in the main window of JP1/AO.

The **Editor** window serves as the starting point for creating, editing, and copying service templates. After creating a blank service template or selecting a template to edit or copy, you can edit the service template by switching to service template editing view.

When you have finished the editing process, you build the service template and debug the result in the **Debug** view and the service template debugging view.

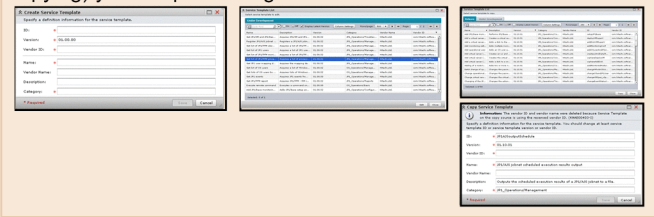
Figure 1–4: Main windows used in service template development (when creating, editing, or copying a service template)

Editor window



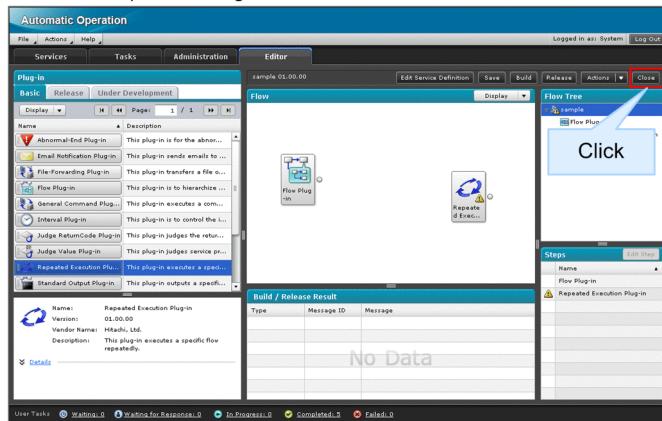
Window transition

The appropriate dialog box for the operation (creating, editing, or copying) you are performing



Window transition

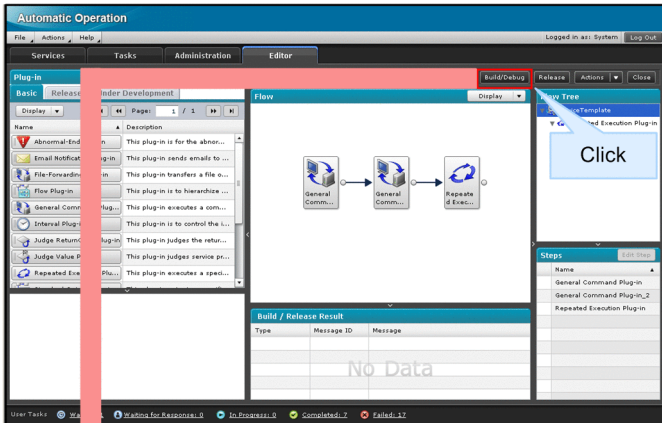
Service template editing view



Window transition

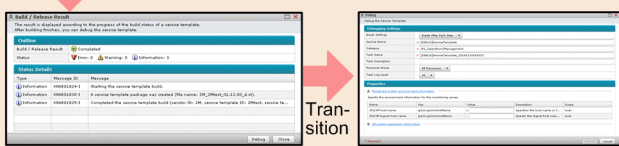
Figure 1–5: Main windows used in service template development (when debugging a service template)

Service template editing view

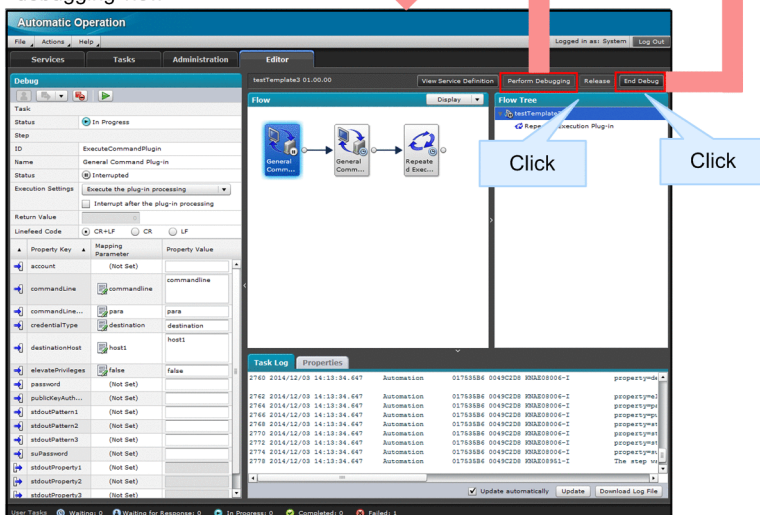


Transition

Build / Release Result dialog box and Perform Debugging dialog box



Debug view and service template debugging view



Transition

Transition

Transition

For information about the window contents, input restrictions, and other details, see the manual *Job Management Partner 1/Automatic Operation GUI, Command, and API Reference*.

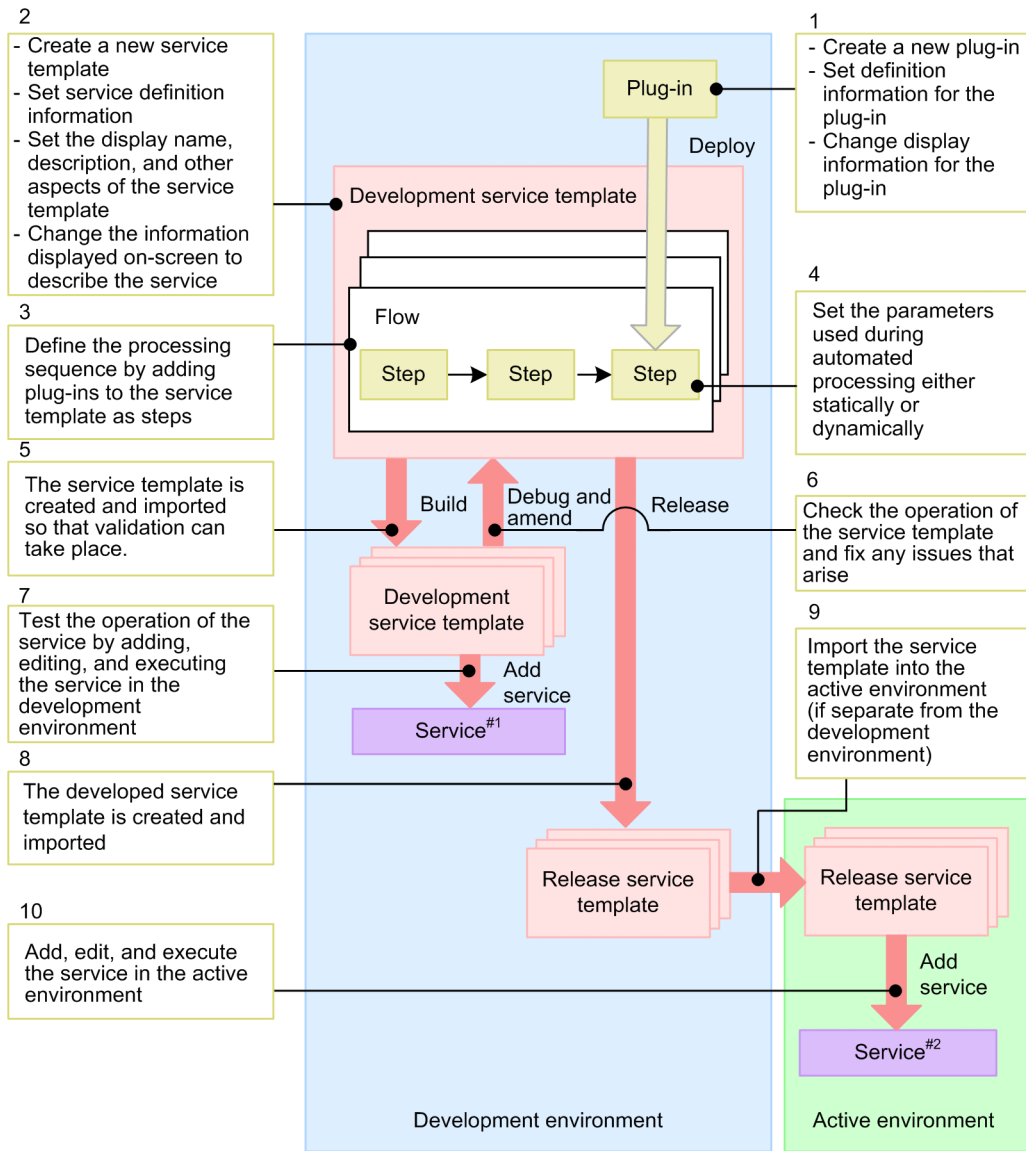


## 1.2 Tasks associated with service templates

### 1.2.1 Tasks performed when creating new service templates

The following figure shows the content and structure of the work performed when creating a new service template. The numbers in the figure indicate the order in which each step is performed.

Figure 1–6: Tasks performed when creating a new service template

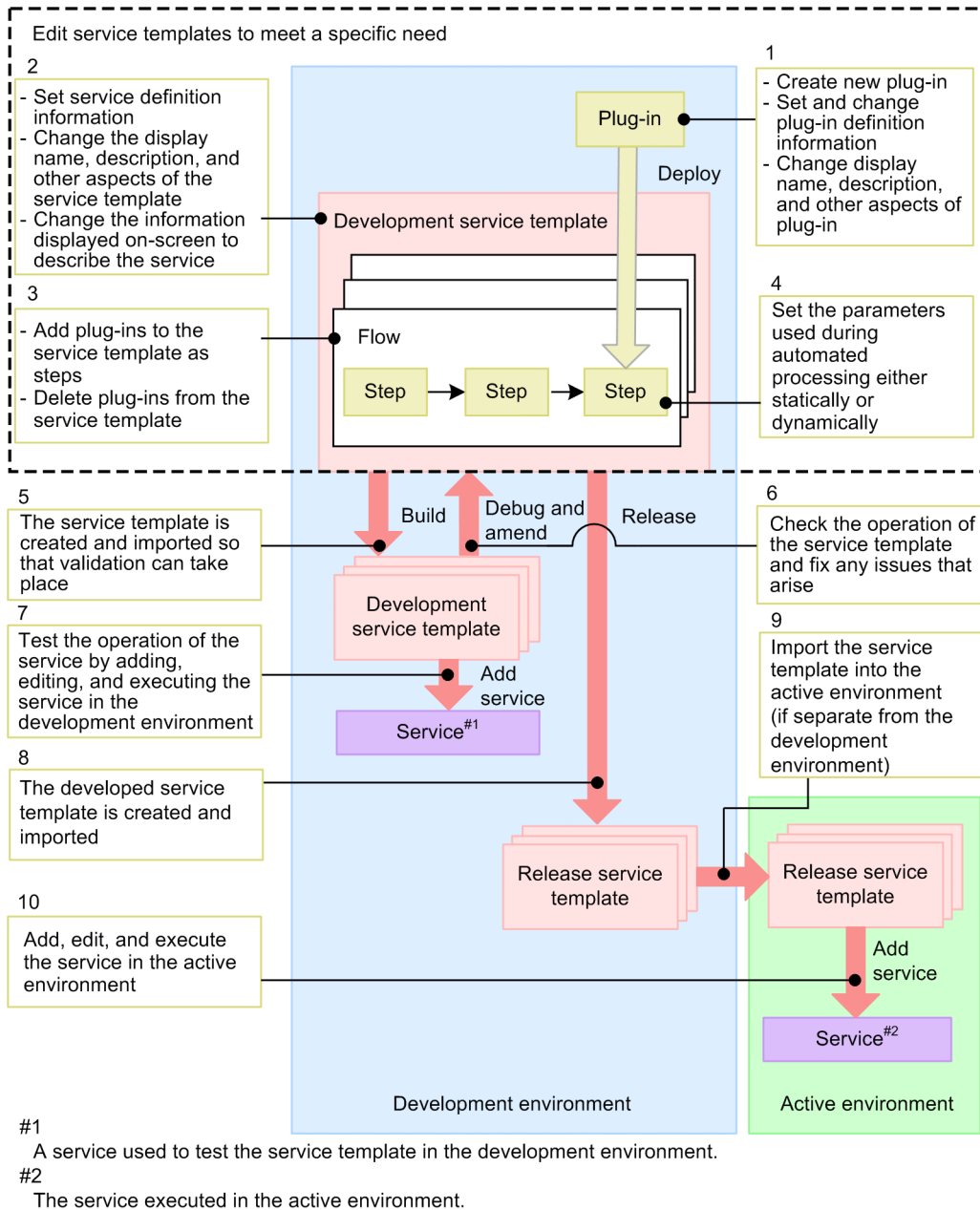


- #1  
A service used to test the service template in the development environment.
- #2  
The service executed in the active environment.

## 1.2.2 Tasks performed when editing service templates

The following figure shows the content and structure of the work performed when editing and re-using a service template. The numbers in the figure indicate the order in which each step is performed. Only perform the tasks in the section enclosed by the dashed line if they are needed.

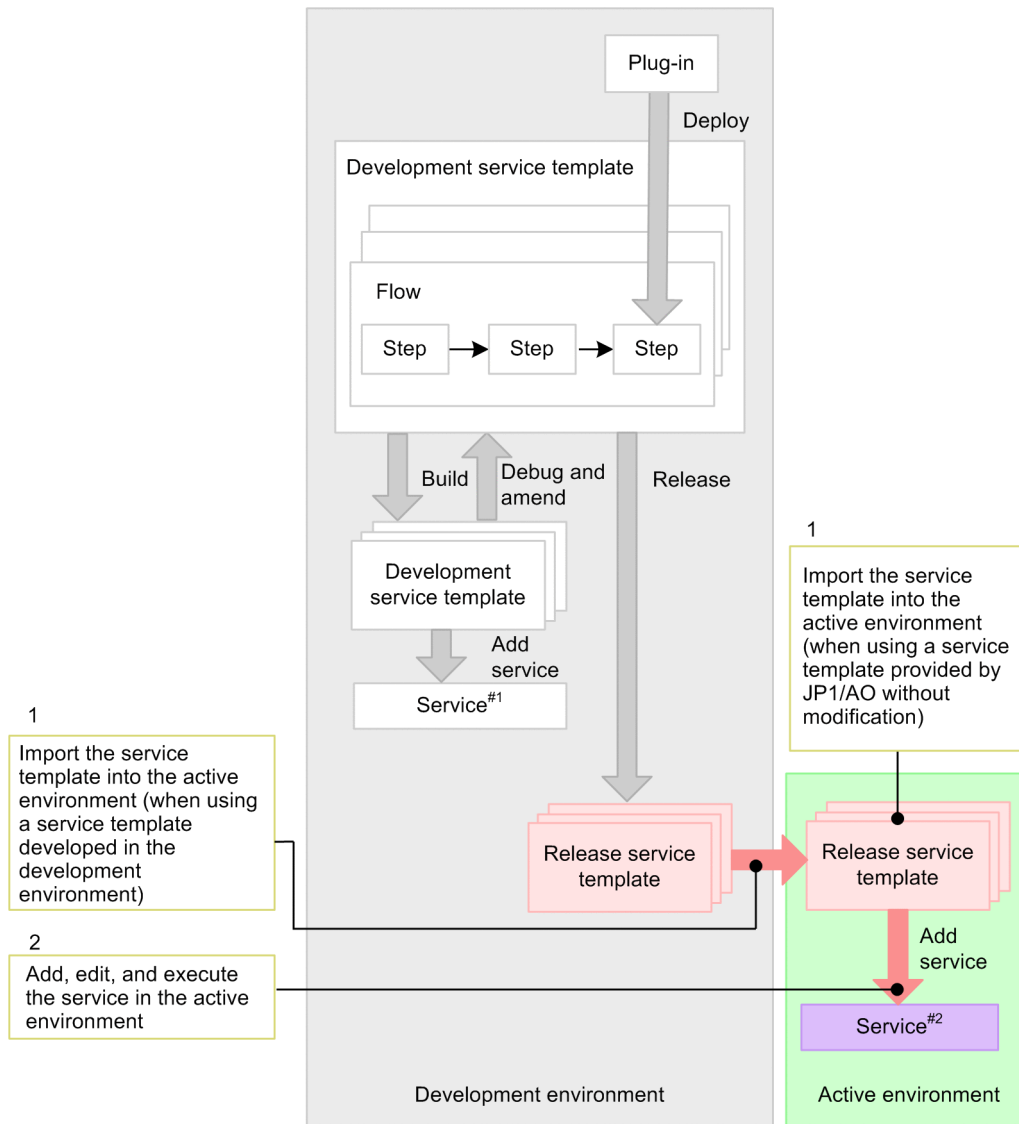
Figure 1–7: Tasks performed when editing a service template



## 1.2.3 Tasks performed when using existing service templates as-is

The following figure shows the content and structure of the work performed when using an existing service template without modification. The numbers in the figure indicate the order in which each step is performed.

Figure 1–8: Tasks performed when using an existing service template as-is



1  
Import the service template into the active environment (when using a service template developed in the development environment)

2  
Add, edit, and execute the service in the active environment

1  
Import the service template into the active environment (when using a service template provided by JP1/AO without modification)

- #1 A service used to test the service template in the development environment.
- #2 The service executed in the active environment.

## 1.3 General procedure for creating new service templates

### 1.3.1 General procedure for creating new service templates

Users are not limited to using and modifying the service templates provided by JP1/AO, and can create new service templates to meet specific needs.

#### You might use this procedure when:

- You want to create your own service template rather than use an existing one.

#### Required knowledge

- [2.1.2 Overview of development service templates and release service templates](#)
- [4.2.2 Parameters to set when creating or copying plug-ins](#)
- [4.3.2 Parameters to set in plug-in definition information](#)
- [2.2.2 Parameters to set when creating or copying service templates](#)
- [2.2.4 Parameters to set in service definition information](#)
- [6.2.1 Overview of service template release](#)

#### General procedure

Table 1–1: General procedure for creating new service templates

Task		Mandatory/ optional	Refer to
1	Display the <b>Editor</b> window	Mandatory	<a href="#">2.1.1 Procedure for displaying the <b>Editor</b> window</a>
2	Create a blank service template	Mandatory	<a href="#">2.2.1 Creating blank service templates</a>
3	If not using an existing plug-in, create a new plug-in	Optional	<a href="#">4.2.1 Procedure for creating plug-ins</a>
	To edit and re-use a release plug-in, copy the plug-in and edit the copy	Optional	<a href="#">4.5.1 Procedure for copying plug-ins</a>
			<a href="#">4.3.1 Procedure for editing plug-in definition information</a>
To edit and use a development plug-in, begin by editing the plug-in	Optional	<a href="#">4.3.1 Procedure for editing plug-in definition information</a>	
4	Set service definition information	Mandatory	<a href="#">2.2.3 Procedure for setting service definition information</a>
5	Add steps	Mandatory	<a href="#">3.2 Adding steps</a>
6	Connect relational lines	Mandatory	<a href="#">3.3 Connecting steps with relational lines</a>
7	Validate created service template <sup>#</sup>	Optional	<a href="#">5. Validating Service Templates</a>
8	Release the completed service template and prepare to add the service	Mandatory	<a href="#">6.1.1 Procedure for releasing a service template</a>

1. Flow of Service Template Development

Task		Mandatory/ optional	Refer to
9	If the development environment and active environment are separate, import the service template into the active environment	Optional	<a href="#">6.3.1 Procedure for importing service templates into the active environment (when the active and development environments are separate)</a>
10	Add, edit, and execute the service in the active environment	Mandatory	The <i>Job Management Partner 1/Automatic Operation Administration Guide</i> <ul style="list-style-type: none"> <li>• Adding services</li> <li>• Editing services</li> <li>• Executing services</li> </ul>

#

If the validation process finds any issues with the service template, repeat tasks 3 to 7 as needed.

## 1.4 General procedure when editing an existing service template

Users can create original service templates by copying and then editing the service templates provided by JP1/AO (in the standard JP1/AO package or JP1/AO Content Set). To edit a release service template, copy the template and then edit the copy as a new service template.

### Important note

Once edited, the service templates (standard JP1/AO package and JP1/AO Content Set) and plug-ins provided by JP1/AO are outside the scope of JP1/AO product support. However, the plug-ins provided by JP1/AO (bundled service templates or JP1/AO Content Set) that are called from such templates remain subject to product support.

The following table shows where to read about the tasks involved in editing a service template.

Table 1–2: Reading material for each development stage

Task	You might perform this task when:	Refer to
Investigation	<ul style="list-style-type: none"><li>You are investigating whether an existing service template or plug-in can be used without further modification.</li></ul>	The manual <i>Job Management Partner 1/Automatic Operation Service Template Reference</i>
Editing service template definition information	<ul style="list-style-type: none"><li>You want to change the name of the service template from "Stop virtual server" to "Stop virtual server and Notify by email".</li><li>You want to add properties to use to input and output values to and from the service template.</li></ul>	<a href="#">1.4.1 General procedure when editing service template definition information</a>
Editing plug-ins	<ul style="list-style-type: none"><li>You want to change aspects of scripts or commands defined in a plug-in.</li><li>You want to change the icon displayed for a plug-in in the <b>Flow</b> view.</li></ul>	<a href="#">1.4.2 General procedure for editing a plug-in and applying the result to a service template</a>
Creating new plug-ins	<ul style="list-style-type: none"><li>You want to create a new plug-in and define processing that executes a command.</li></ul>	<a href="#">1.4.3 General procedure for creating new plug-ins and adding them to service templates</a>
Changing the contents of the window that describes the nature of the service	<ul style="list-style-type: none"><li>You want to change the description that appears for the service in the <b>Service Details</b> window.</li><li>You want to add a cautionary note to the <b>Service Details</b> window.</li></ul>	<a href="#">1.4.4 General procedure for changing the description displayed for a service in the user interface</a>
Adding plug-ins to service templates	<ul style="list-style-type: none"><li>You want to insert an email-sending process at the end of the processing automated by the service template.</li><li>You want to insert processing that controls the execution interval between processes</li></ul>	<a href="#">1.4.5 General procedure for adding processing to a service template</a>

Task	You might perform this task when:	Refer to
Adding plug-ins to service templates	automated by the service template.	<a href="#">1.4.5 General procedure for adding processing to a service template</a>
Deleting plug-ins from service templates	<ul style="list-style-type: none"> <li>• A file transfer step is no longer required in processing that acquires log data for JP1/IM and JP1/Base.</li> <li>• Processing that deletes temporary files is no longer required when mapping JP1 users.</li> </ul>	<a href="#">1.4.6 General procedure for deleting processing from a service template</a>
Setting parameters dynamically or statically during automated processing	<ul style="list-style-type: none"> <li>• In processing that increases available memory, you want to be able to specify how much memory to allocate when executing a service, instead of using a fixed value.</li> <li>• In processing that increases available memory, you want the memory capacity to be fixed at 5 GB each time the service runs.</li> <li>• When a command is executed as part of a service, you want to extract an arbitrary character string from the standard output or standard error output of the command and display it in the <b>Task Details</b> dialog box.</li> </ul>	<a href="#">1.4.7 General procedure for dynamically or statically setting parameters during automated processing</a>
Batch modification of display information using a service resource file	<ul style="list-style-type: none"> <li>• You want to use a service resource file to change several display items defined for a service template in a single operation.</li> </ul>	<a href="#">1.4.8 General procedure for using a service resource file to set the information displayed for a service</a>
Batch modification of display information using a plug-in resource file	<ul style="list-style-type: none"> <li>• You want to use a plug-in resource file to change several display items defined in a plug-in in a single operation.</li> </ul>	<a href="#">1.4.9 General procedure for using a plug-in resource file to set display information for a plug-in</a>

## 1.4.1 General procedure when editing service template definition information

Service template definition information consists of the name of the service template as it appears in the JP1/AO interface, the input and output properties of the template, and other such information.

### You might use this procedure when:

- You want to change the name of the service template from "Stop virtual server" to "Stop virtual server and Notify by email".
- You want to add a property that inputs or outputs a value to or from a service template.

### Required knowledge

- [2.1.2 Overview of development service templates and release service templates](#)

- 2.2.4 Parameters to set in service definition information
- 6.2.1 Overview of service template release

## General procedure

Table 1–3: General procedure when editing service template definition information

Task	Mandatory/optional	Refer to
1 Display the <b>Editor</b> window	Mandatory	2.1.1 Procedure for displaying the <b>Editor</b> window
2 Copy the release service template that you want to edit	Optional	2.4.1 Procedure for copying service templates
3 Change the definition information (settings that relate to services, property groups, input properties, output properties, and variables) for the service template	Mandatory	2.2.3 Procedure for setting service definition information
4 Validate an edited service template <sup>#</sup>	Optional	5. Validating Service Templates
5 Release the completed service template and prepare to add the service	Mandatory	6.1.1 Procedure for releasing a service template
6 Import a service template to the active environment (if the development and active environments are separate)	Optional	6.3.1 Procedure for importing service templates into the active environment (when the active and development environments are separate)
7 Add, edit, and execute services in the active environment	Mandatory	The <i>Job Management Partner 1/Automatic Operation Administration Guide</i> <ul style="list-style-type: none"> <li>• Adding services</li> <li>• Editing services</li> <li>• Executing services</li> </ul>

#

If the validation process finds any issues with the service template, repeat tasks 3 and 4 as needed.

## 1.4.2 General procedure for editing a plug-in and applying the result to a service template

You can edit plug-ins (except for basic plug-ins). After editing a plug-in, you can apply the result to a service template. If the plug-in is already released, copy the plug-in and edit the copy.

### You might use this procedure when:

- You want to change the contents of a script or command defined in a plug-in.
- You want to change the icon displayed for a plug-in in the **Flow** view.

### Required knowledge

- 4.1.2 Overview of basic plug-ins, release plug-ins, and development plug-ins
- 4.3.2 Parameters to set in plug-in definition information
- 2.1.2 Overview of development service templates and release service templates



- 2.2.4 Parameters to set in service definition information
- 6.2.1 Overview of service template release

## General procedure

Table 1–4: General procedure for editing a plug-in and applying the result to a service template

Task		Mandatory/optional	Refer to
1	Display the <b>Editor</b> window	Mandatory	2.1.1 Procedure for displaying the <b>Editor</b> window
2	Copy the release plug-in that you want to edit	Optional	4.5.1 Procedure for copying plug-ins
3	Select the plug-in to edit from a list of plug-ins	Mandatory	4.1.1 Procedure for displaying a list of plug-ins
4	Edit plug-in definition information	Mandatory	4.3.1 Procedure for editing plug-in definition information
5	Copy the service template (when editing a release service template)	Optional	2.4.1 Procedure for copying service templates
6	Set service definition information when changes made to plug-ins have made the existing information invalid	Optional	2.2.3 Procedure for setting service definition information
7	Add or delete steps when editing a plug-in affects the flow of processing	Optional	3.2 Adding steps
8	Add and delete relational lines when editing a plug-in affects the flow of processing	Optional	3.3 Connecting steps with relational lines
9	Validate the edited service template <sup>#</sup>	Optional	5. Validating Service Templates
10	Release the completed service template and prepare to add the service	Mandatory	6.1.1 Procedure for releasing a service template
11	Import the service template to the active environment (if the development and active environments are separate)	Optional	6.3.1 Procedure for importing service templates into the active environment (when the active and development environments are separate)
12	Add, edit, and execute services in the active environment	Mandatory	The <i>Job Management Partner 1/Automatic Operation Administration Guide</i> <ul style="list-style-type: none"> <li>• Adding services</li> <li>• Editing services</li> <li>• Executing services</li> </ul>

#

If the validation process finds an issue with the service template, repeat tasks 6 to 9 as needed.

### 1.4.3 General procedure for creating new plug-ins and adding them to service templates

Users can create custom plug-ins and add them to service templates as steps.

## You might use this procedure when:

- You want to create a new plug-in and define processing that executes a command.

## Required knowledge

- [4.1.2 Overview of basic plug-ins, release plug-ins, and development plug-ins](#)
- [4.3.2 Parameters to set in plug-in definition information](#)
- [2.1.2 Overview of development service templates and release service templates](#)
- [2.2.4 Parameters to set in service definition information](#)
- [6.2.1 Overview of service template release](#)

## General procedure

Table 1–5: General procedure for creating new plug-ins and adding them to service templates

Task	Mandatory/optional	Refer to
1	Display the <b>Editor</b> window	Mandatory 2.1.1 Procedure for displaying the <b>Editor</b> window
2	Create a plug-in	Mandatory 4.2.1 Procedure for creating plug-ins
3	Edit plug-in definition information	Mandatory 4.3.1 Procedure for editing plug-in definition information
4	Copy the release service template that you want to edit	Optional 2.4.1 Procedure for copying service templates
5	Set service definition information if creating a new plug-in has made the existing information invalid	Optional 2.2.3 Procedure for setting service definition information
6	Add a plug-in you created as a step in a flow	Mandatory 3.2.1 Procedure for adding steps
7	Connect steps with relational lines	Mandatory 3.3.1 Procedure for connecting steps with relational lines
8	Validate the edited service template <sup>#</sup>	Optional 5. Validating Service Templates
9	Release a completed service template and prepare to add the service	Mandatory 6.1.1 Procedure for releasing a service template
10	Import the service template to the active environment (if the development and active environments are separate)	Optional 6.3.1 Procedure for importing service templates into the active environment (when the active and development environments are separate)
11	Add, edit, and execute services in the active environment	Mandatory The <i>Job Management Partner 1/Automatic Operation Administration Guide</i> <ul style="list-style-type: none"> <li>• Adding services</li> <li>• Editing services</li> <li>• Executing services</li> </ul>

#

If the validation process finds an issue with the service template, repeat tasks 5 to 8 as needed.

## 1.4.4 General procedure for changing the description displayed for a service in the user interface

When you select a service in the **Services** window, the **Add Service** dialog box, or the **Edit Service** dialog box and click **Show Service Details**, a description of the selected service appears in a new window. You can change the description displayed in the **Service Details** window by creating or editing a custom file.

### You might use this procedure when:

- You want to change the description of the detailed specification displayed for a service in the **Service Details** window.
- You want to add cautionary notes to the **Service Details** window.

### Required knowledge

- [2.2.7 Overview of custom files](#)
- [2.2.4 Parameters to set in service definition information](#)
- [6.2.1 Overview of service template release](#)

### General procedure

Table 1–6: General procedure for changing the description displayed for a service in the user interface

Task		Mandatory/optional	Refer to
1	Display the <b>Editor</b> window	Mandatory	<a href="#">2.1.1 Procedure for displaying the <b>Editor</b> window</a>
2	Select a custom file that defines the contents of the window that displays service descriptions	Mandatory	<a href="#">2.2.6 Procedure for setting custom files</a>
3	Validate the edited service template <sup>#</sup>	Optional	<a href="#">5. Validating Service Templates</a>
4	Release the completed service template and prepare to add the service	Mandatory	<a href="#">6.1.1 Procedure for releasing a service template</a>
5	Import the service template to the active environment (if the development and active environments are separate)	Optional	<a href="#">6.3.1 Procedure for importing service templates into the active environment (when the active and development environments are separate)</a>
6	Add, edit, and execute services in the active environment	Mandatory	The <i>Job Management Partner 1/Automatic Operation Administration Guide</i> <ul style="list-style-type: none"> <li>• Adding services</li> <li>• Editing services</li> <li>• Executing services</li> </ul>

#

If the validation process finds an issue with the service template, repeat tasks 2 and 3 as needed.

## 1.4.5 General procedure for adding processing to a service template

Users can add processing in the form of steps to a flow in an existing service template.

## You might use this procedure when:

- You want to insert processing that sends an email when the processing automated by a service template has finished.
- You want to insert processing that controls the execution interval between processes automated by a service template.

## Required knowledge

- [2.1.2 Overview of development service templates and release service templates](#)
- [3.1.2 Relationship between flow and steps](#)
- [3.1.3 Creating flow hierarchies](#)
- [2.2.4 Parameters to set in service definition information](#)
- [6.2.1 Overview of service template release](#)

## General procedure

Table 1–7: General procedure for adding processing to a service template

Task		Mandatory/ optional	Refer to
1	Display the <b>Editor</b> window	Mandatory	<a href="#">2.1.1 Procedure for displaying the <b>Editor</b> window</a>
2	Copy a release service template that you want to edit	Optional	<a href="#">2.4.1 Procedure for copying service templates</a>
3	Display the <b>Flow</b> view in order to define the flow of processing	Mandatory	<a href="#">3.1.1 Procedure for displaying the <b>Flow</b> view</a>
4	Add steps	Mandatory	<a href="#">3.2.1 Procedure for adding steps</a>
5	Connect steps with relational lines to define the order in which processing is executed	Mandatory	<a href="#">3.3.1 Procedure for connecting steps with relational lines</a>
6	Amend service definition information if the added step requires changes to the service properties	Optional	<a href="#">2.2.3 Procedure for setting service definition information</a>
7	Validate the edited service template <sup>#</sup>	Optional	<a href="#">5. Validating Service Templates</a>
8	Release the completed service template and prepare to add the service	Mandatory	<a href="#">6.1.1 Procedure for releasing a service template</a>
9	Import the service template to the active environment (if the development and active environments are separate)	Optional	<a href="#">6.3.1 Procedure for importing service templates into the active environment (when the active and development environments are separate)</a>
10	Add, edit, and execute services in the active environment	Mandatory	<i>The Job Management Partner 1/Automatic Operation Administration Guide</i> <ul style="list-style-type: none"> <li>• Adding services</li> <li>• Editing services</li> <li>• Executing services</li> </ul>

#

If the validation process finds an issue with the service template, repeat tasks 3 to 7 as needed.

## 1.4.6 General procedure for deleting processing from a service template

You can delete redundant steps from the flow of an existing service template.

### You might use this procedure when:

- A file transfer step is no longer required in processing that acquires JP1/IM and JP1/Base log files.
- Processing that deletes temporary files is no longer required in process that maps JP1 users.

### Required knowledge

- [2.1.2 Overview of development service templates and release service templates](#)
- [3.1.2 Relationship between flow and steps](#)
- [2.2.4 Parameters to set in service definition information](#)
- [6.2.1 Overview of service template release](#)

### General procedure

Table 1–8: General procedure for deleting processing from a service template

Task	Mandatory/optional	Refer to
1	Mandatory	<a href="#">2.1.1 Procedure for displaying the <b>Editor</b> window</a>
2	Optional	<a href="#">2.4.1 Procedure for copying service templates</a>
3	Mandatory	<a href="#">3.1.1 Procedure for displaying the <b>Flow</b> view</a>
4	Mandatory	<a href="#">3.3.2 Procedure for deleting steps and relational lines</a>
5	Optional	<a href="#">3.3 Connecting steps with relational lines</a>
6	Optional	<a href="#">2.2.3 Procedure for setting service definition information</a>
7	Optional	<a href="#">5. Validating Service Templates</a>
8	Mandatory	<a href="#">6.1.1 Procedure for releasing a service template</a>
9	Optional	<a href="#">6.3.1 Procedure for importing service templates into the active environment (when the active and development environments are separate)</a>
10	Mandatory	<i>The Job Management Partner 1/Automatic Operation Administration Guide</i> <ul style="list-style-type: none"> <li>• Adding services</li> <li>• Editing services</li> <li>• Executing services</li> </ul>

#

If the validation process finds an issue with the service template, repeat tasks 3 to 7 as needed.

## 1.4.7 General procedure for dynamically or statically setting parameters during automated processing

By mapping service properties to the properties of a plug-in, you can execute processing that maps a value to an input property. You can also assign a fixed value to an input property, or define a service template in such a manner that the output of a plug-in is passed to an output property of a service template or a variable.

This process is not limited to existing input properties or output properties. Users can create new properties and map them to plug-in properties.

### You might use this procedure when:

- In processing that increases available memory, you want to be able to specify how much memory to allocate when executing a service (dynamically setting input properties), instead of using a fixed value.
- In processing that increases available memory, you want the memory capacity to be fixed at 5 GB each time the service runs (statically setting input properties).
- When a command is executed as part of a service, you want to extract an arbitrary character string from the standard output or standard error output of the command and display it in the **Task Details** dialog box (dynamically setting output properties).

### Required knowledge

- [2.1.2 Overview of development service templates and release service templates](#)
- [3.2.3 Settings in step definition information](#)
- [2.2.5 Example of mapping parameter definition and flow of data](#)
- [3.2.3 Settings in step definition information](#)
- [6.2.1 Overview of service template release](#)

### General procedure

Table 1–9: General procedure for dynamically or statically setting parameters during automated processing

Task	Mandatory/optional	Refer to
1 Display the <b>Editor</b> window	Mandatory	<a href="#">2.1.1 Procedure for displaying the <b>Editor</b> window</a>
2 Copy a release service template that you want to edit	Optional	<a href="#">2.4.1 Procedure for copying service templates</a>
3 Select a step and change property mapping in the definition information	Mandatory	<a href="#">3.2.2 Procedure for changing step definition information</a>
4 Validate an edited service template <sup>#</sup>	Optional	<a href="#">5. Validating Service Templates</a>
5 Release the completed service template and prepare to add the service	Mandatory	<a href="#">6.1.1 Procedure for releasing a service template</a>
6 Import the service template to the active environment (if the development and active environments are separate)	Optional	<a href="#">6.3.1 Procedure for importing service templates into the active environment (when the active and development environments are separate)</a>

Task	Mandatory/optional	Refer to
7	Mandatory	The <i>Job Management Partner 1/Automatic Operation Administration Guide</i> <ul style="list-style-type: none"> <li>• Adding services</li> <li>• Editing services</li> <li>• Executing services</li> </ul>

#

If the validation process finds an issue with the service template, repeat tasks 3 and 4 as needed.

## 1.4.8 General procedure for using a service resource file to set the information displayed for a service

You can use a service resource file to set the information displayed for a service template in the JP1/AO user interface.

### You might use this procedure when:

- You want to use a service resource file to change several display items defined in a service template in a single operation.

### Required knowledge

- [2.1.2 Overview of development service templates and release service templates](#)
- [2.2.4 Parameters to set in service definition information](#)
- [2.6.4 Correspondence between information displayed in service templates and properties in service resource files](#)
- [6.2.1 Overview of service template release](#)

### General procedure

Table 1–10: General procedure for using a service resource file to set the information displayed for a service

Task	Mandatory/optional	Refer to
1	Mandatory	<a href="#">2.1.1 Procedure for displaying the <b>Editor</b> window</a>
2	Optional	<a href="#">2.4.1 Procedure for copying service templates</a>
3	Mandatory	<a href="#">2.6.1 Procedure for setting service resource files</a>
4	Optional	<a href="#">5. Validating Service Templates</a>
5	Mandatory	<a href="#">6.1.1 Procedure for releasing a service template</a>

Task		Mandatory/optional	Refer to
6	Import the service template to the active environment (if the development and active environments are separate)	Optional	6.3.1 Procedure for importing service templates into the active environment (when the active and development environments are separate)
7	Add, edit, and execute services in the development environment	Mandatory	The <i>Job Management Partner 1/Automatic Operation Administration Guide</i> <ul style="list-style-type: none"> <li>• Adding services</li> <li>• Editing services</li> <li>• Executing services</li> </ul>

#

If the validation process finds an issue with the service template, repeat tasks 3 and 4 as needed.

## 1.4.9 General procedure for using a plug-in resource file to set display information for a plug-in

You can use a plug-in resource file to set the information displayed for a plug-in in the JP1/AO user interface.

### You might use this procedure when:

- You want to use a plug-in resource file to change several display items defined for a plug-in in a single operation.

### Required knowledge

- [2.1.2 Overview of development service templates and release service templates](#)
- [2.2.4 Parameters to set in service definition information](#)
- [4.6.3 Correspondence between properties in plug-in resource files and information displayed for plug-ins](#)
- [6.2.1 Overview of service template release](#)

### General procedure

Table 1–11: General procedure for using a plug-in resource file to set display information for a plug-in

Task		Mandatory/optional	Refer to
1	Display the <b>Editor</b> window	Mandatory	2.1.1 Procedure for displaying the <b>Editor</b> window
2	Copy a release service template that you want to edit	Optional	2.4.1 Procedure for copying service templates
3	Configure a plug-in resource file that sets the information displayed for a plug-in	Mandatory	4.6.1 Procedure for setting plug-in resource files



## 1.5 Using existing service templates provided by JP1/AO

### 1.5.1 General procedure for using an existing service template provided by JP1/AO

When appropriate, you can use the service templates provided in the standard JP1/AO package and the JP1/AO Content Set.

#### You might use this procedure when:

In situations like the following, a template provided by JP1/AO exactly defines the task you want to automate.

- You want to use the Add monitoring setting service template to add multiple monitored servers to JP1/Cm2/NNMi or JP1/PFM
- You want to use the Add operational user service template to add OS users, JP1 users, and the associated mapping information.

#### Required knowledge

- [2.1.2 Overview of development service templates and release service templates](#)

#### General procedure

Table 1–12: General procedure for using an existing service template provided by JP1/AO

Task	Mandatory/optional	Refer to
1 Evaluate the service template you want to use	Mandatory	<ul style="list-style-type: none"><li>• Evaluating the service template to be used and the targets of operation in the <i>Job Management Partner 1/Automatic Operation Overview and System Design Guide</i></li><li>• The following sections in the <i>Job Management Partner 1/Automatic Operation Service Template Reference</i><ul style="list-style-type: none"><li>• List of JP1/AO Standard-package Service Templates</li><li>• List of JP1/AO Content Set Service Templates</li></ul></li></ul>
2 Add service templates to JP1/AO	Mandatory	Importing service templates in the <i>Job Management Partner 1/Automatic Operation Administration Guide</i>
3 Add, edit, and execute services	Mandatory	The <i>Job Management Partner 1/Automatic Operation Administration Guide</i> <ul style="list-style-type: none"><li>• Adding services</li><li>• Editing services</li><li>• Executing services</li></ul>

## 1.6 List of service template development features

The following table lists the features of JP1/AO that are used in the development of service templates.

Table 1–13: List of service template development features

Feature		Description	Reference
Service template management	Creating and editing service templates	Users can create new custom service templates. Users can also edit existing service templates, replacing parameters such as the service template name, description, service properties, and custom file to suit their needs.	<a href="#">2.2 Creating and editing service templates</a>
	Listing service templates	You can display a list of service templates.	<a href="#">2.3 Displaying a list of service templates</a>
	Copying service templates	You can copy a service template. You can then create a new service template by editing the copy.	<a href="#">2.4 Copying service templates</a>
	Deleting service templates	You can delete a service template.	<a href="#">2.5 Deleting service templates</a>
	Setting display information for service templates	You can create a service resource file that defines the information displayed for a service template.	<a href="#">2.6 Setting display information for service templates in resource files</a>
Flow management	Adding steps	You can insert processing where it is needed by adding a step to a flow. You can also edit the definition information for a step.	<a href="#">3.2 Adding steps</a>
	Connecting relational lines	After adding a step, you can define its place in the execution order by connecting relational lines.	<a href="#">3.3 Connecting steps with relational lines</a>
Plug-in management	Listing plug-ins	You can display a list of plug-ins.	<a href="#">4.1 Displaying a list of plug-ins</a>
	Creating plug-ins	Users can create custom plug-ins to suit their needs.	<a href="#">4.2 Creating plug-ins</a>
	Editing plug-ins	You can edit plug-ins. This might involve changing the name, input properties, output properties, remote commands, or other aspects of a plug-in. Note that you cannot edit the plug-ins provided in the standard JP1/AO package.	<a href="#">4.3 Editing plug-ins</a>
	Deleting plug-ins	You can delete development plug-ins and release plug-ins.	<a href="#">4.4 Deleting plug-ins</a>
	Copying plug-ins	You can copy a plug-in. You can then create a new plug-in by editing the copy.	<a href="#">4.5 Copying plug-ins</a>
	Setting plug-in display information	You can create a plug-in resource file that defines the information displayed for a plug-in, such as its name and description.	<a href="#">4.6 Using resource files to set plug-in display information</a>
Validation of service templates	Building service templates	When you build a service template, the JP1/AO system creates a package of the service template you are developing and imports it to the JP1/AO server. Use this feature when you need to validate a service template.	<a href="#">5.2 Building service templates</a>
	Debugging service templates	You can check the operation of a service template you have built and identify any issues. If the debug process reveals an issue in the flow or in a plug-in, you can edit the service template or plug-in from the <b>Editor</b> window.	<a href="#">5.3 Debugging service templates</a>

Feature		Description	Reference
Validation of service templates	Testing service template operation	You can add and execute a service based on a service template you have built, and test the service for issues. If the test identifies an issue in the service, you can edit the service template or plug-in in the <b>Editor</b> window.	<a href="#">5.5 Testing service templates</a>
Build and release	Releasing service templates	After validating a service template, you can release it. This creates a package from the service template which is imported to the JP1/AO server. You cannot edit a service template after releasing it. If your development environment and active environment are separate, you will need to import the service template manually.	<a href="#">6.1 Releasing service templates</a>
	Importing service templates	If your development and active environments are on different servers, this feature imports released services to the active environment.	<a href="#">6.3 Importing service templates into the active environment (when the active and development environments are separate)</a>

# 2

## Setting service template definition information

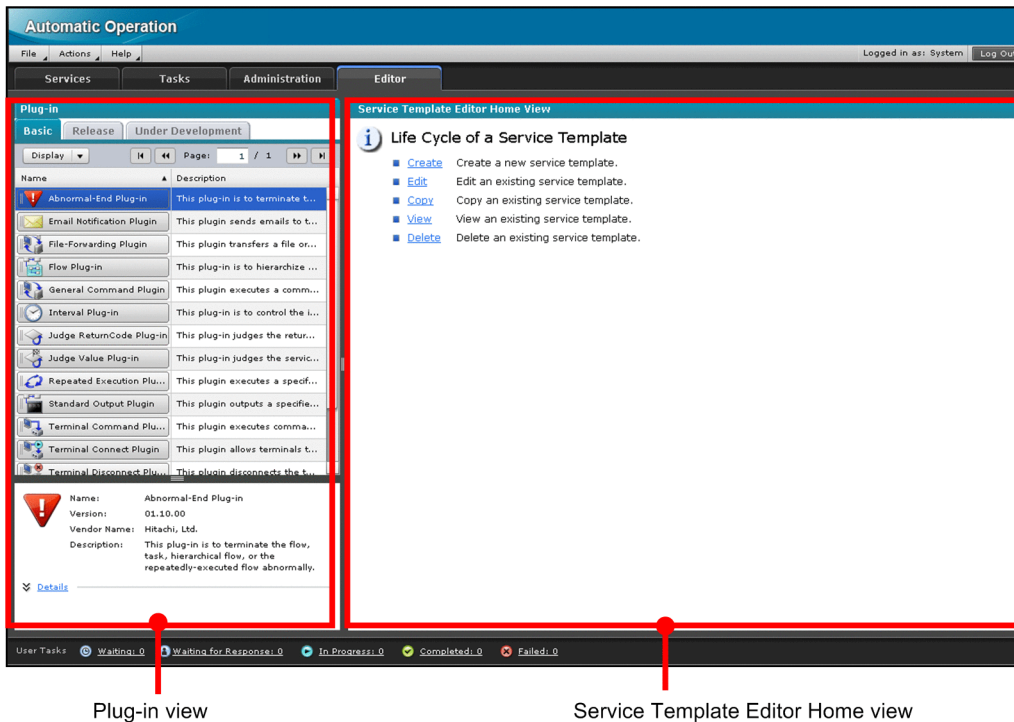
This chapter describes how to set definition information when creating and editing service templates.

## 2.1 Displaying the Editor window

### 2.1.1 Procedure for displaying the Editor window

Users develop service templates in the **Editor** window. There are two parts to the **Editor** window: **Plug-in** view and the **Service Template Editor Home** view.

Figure 2–1: **Editor** window



#### Who can display the Editor window:

Users in the Admin role or the Develop role

#### To display the Editor window:

1. Log in to JP1/AO and display the main window.
2. In the main window, click the **Editor** tab.  
The **Editor** window appears.

There are two types of service template in the **Editor** window: Development service templates and release service templates.

For details about the service templates provided by JP1/AO, see the manual *Job Management Partner 1/Automatic Operation Service Template Reference*.

#### Related topics

- 2.1.2 Overview of development service templates and release service templates
- 2.1.5 Version compatibility for service templates and plug-ins created in the **Editor** window
- 2.1.4 Behavior when an intervening action occurs in the **Editor** window

## 2.1.2 Overview of development service templates and release service templates

There are two types of service template you can edit in the **Editor** window:

### Development service template

A service template a user is developing. Service templates created by copying a release service template are also categorized as development service templates. Development service templates appear in the **Under Development** tab of the **Service Template List** dialog box.

When you build a development service template, *Debug* is set as the configuration type and execution of the service can be tested. Services added from a development service template are used in a development environment. Any service template that is not yet built is also categorized as a development service template.

### Release service template

A service template that was imported into the JP1/AO server by releasing a development service template. Service templates provided by JP1/AO are also categorized as release service templates. Release service templates are used for real-world applications in the active environment. *Released* is set as the configuration type of release service templates. These templates appear in the **Release** tab of the **Service Template List** dialog box.

Service templates that were imported to the JP1/AO server by executing the `importservicetemplate` command and have the configuration type *release* are handled as release service templates.

Note that you cannot edit a service template after its release. To edit such a template, copy the release template and then edit the copy as a development service template.

### Important note

Once edited, the service templates and plug-ins provided by JP1/AO are outside the scope of JP1/AO product support. However, product support is still offered for the plug-ins provided by JP1/AO (in the standard package or the JP1/AO Content Set) that are called from such templates.

### Related topics

- [2.1.3 Available operations by service template configuration type](#)
- [6. Releasing Service Templates](#)

## 2.1.3 Available operations by service template configuration type

The operations you can perform on development service templates and release service templates depend on the configuration type of the template.

Table 2–1: Available operations by service template configuration type

Template	Configuration type	Operation applied to service template							
		Display in list	Create	Edit	Copy	View	Delete	Build	Release
Development service template	Debug	Y	Y	Y	Y	Y	Y#	Y	Y
Release service template	Release	Y	N	N	Y	Y	N	N	N

Legend:

Y: Can be performed. N: Cannot be performed.

#

When you add a service from a development service template, you cannot delete the template while a task generated from the service is running.

## 2.1.4 Behavior when an intervening action occurs in the Editor window

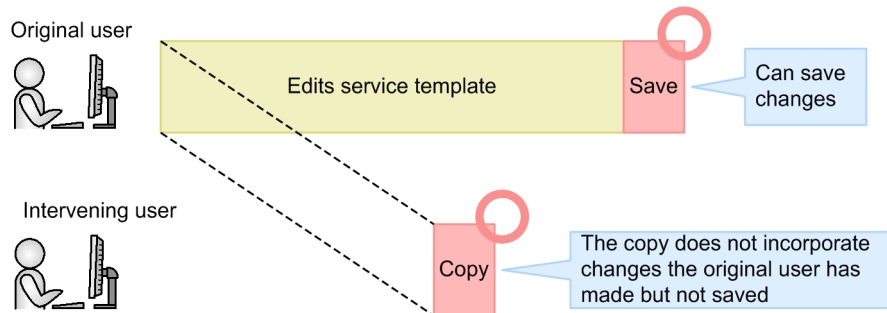
This section describes what happens when another user performs an intervening action on a service template or plug-in you are in the process of editing. Although the descriptions in this section use the example of a service template, the same applies to plug-ins.

A user who is in the process of editing the service template in the **Editor** window is called the original user. A user who performs an operation on a service template the original user is working on is called the intervening user.

- Copying a service template that is being edited

The intervening user can copy a service template that the original user is in the process of editing. In this case, the copy of the service template does not incorporate any changes the original user has made but not saved.

Figure 2–2: Copying a service template that is being edited

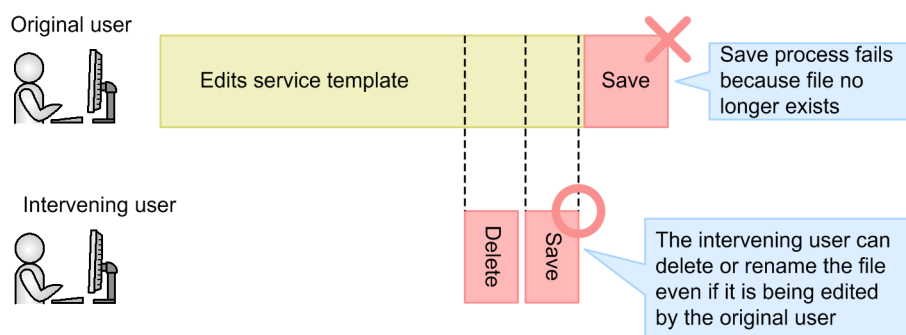


- Deleting or renaming a file while the associated service template or plug-in is being edited

If an intervening user performs any of the following operations on a service template or plug-in the original user is editing, an error message appears when the original user attempts to save, build, or release the template or plug-in.

- Deletes a plug-in icon assigned to the plug-in (excluding standard plug-in icons)
- Deletes or renames a script file assigned to a plug-in
- Deletes or renames a custom file assigned to a service template

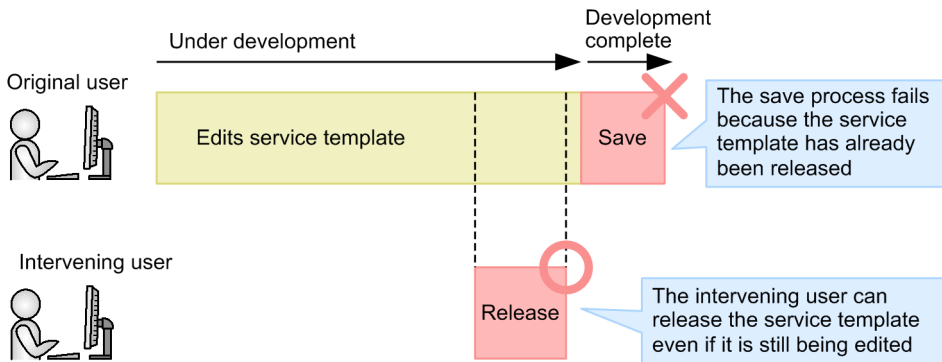
Figure 2–3: Renaming or deleting a file during editing



- Releasing a service template that is being edited

If an intervening user releases a service template an original user is editing, the service template is released normally. However, the original user is not made aware in the editing window that the template has been released. When the original user attempts to save his or her changes, the save process fails and an error message appears. Before you release a service template, make sure that it is not being edited by another user.

Figure 2–4: Releasing a service template that is being edited



## 2.1.5 Version compatibility for service templates and plug-ins created in the Editor window

The following describes the version compatibility for plug-ins and service templates created in the **Editor** window.

### Compatibility with later versions

Service templates are guaranteed to remain compatible with later versions of JP1/AO. You can work with service templates in the **Editor** window that meet the conditions below. However, a subset of displayed items and operations might differ between versions.

- Service template packages (\*.st) created by users in earlier versions
- Service templates and plug-ins imported to the JP1/AO server from an earlier version of JP1/AO

### Compatibility with earlier versions

Service templates are not guaranteed to be backwards compatible. If you attempt to apply a service template you created in the **Editor** window to an earlier version of JP1/AO, an error might occur during the import process.

## 2.1.6 Compatibility with steps in service templates created in earlier versions of JP1/AO

Some steps in service templates created in earlier versions of JP1/AO are not compatible with version 10-10. This might prevent you from changing the processing in the step. The icon for steps that are incompatible with the version of JP1/AO you are using is displayed in gray scale as follows:

Figure 2–5: Icon for incompatible steps





- You cannot change the processing defined in a step. You can only change the step ID, name, and description.
- The default step ID is `compatible.step`.  
If there are several incompatible steps within the same hierarchy, `_n` is appended to the end of the step ID (where `n` is a unique integer starting from 2) to give a step ID in the format `step-ID_n`. Step IDs are automatically allocated by the system. The value of `_n` has no bearing on the position of the step in the flow.
- The description of a step is initially a blank character.

## 2.2 Creating and editing service templates

### 2.2.1 Creating blank service templates

The first thing you need to do when creating a new service template is set the service template ID, service template version, vendor ID, and other definition information.

#### To create a blank service template:

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Create**.
2. In the **Create Service Template** dialog box, set the definition information for the service template and then click **Save**.  
Note that you cannot change the values in the **ID**, **Version**, or **Vendor ID** field after the service template has been created.

After creating a blank service template, you can then perform tasks like setting the service definition information and creating and editing the flow.

#### Related topics

- [2.2.2 Parameters to set when creating or copying service templates](#)
- [2.2.3 Procedure for setting service definition information](#)
- [3. Creating and editing flows](#)

### 2.2.2 Parameters to set when creating or copying service templates

The following table lists the parameters you can set in the **Create Service Template** dialog box and the **Copy Service Template** dialog box:

Table 2–2: Parameters in the **Create Service Template** and the **Copy Service Template** dialog boxes

Parameter	Description
<b>ID</b> <sup>#1</sup>	Specify an ID that identifies the service template.
<b>Version</b> <sup>#1</sup>	Specify the version number of the service template, in the format <i>aa.bb.cc</i> .
<b>Vendor ID</b> <sup>#1</sup>	Specify the ID of the vendor who created the service template. Create a unique vendor ID by specifying the domain name in reverse order from the top level as a period-separated value. For example, specify vendor IDs in the format <i>com.xxx</i> or <i>jp.co.yyy</i> . If you choose not to use domain names as vendor IDs, make sure that the vendor ID you specify is not being used for another vendor. You cannot specify a vendor ID that starts with <i>com.hitachi.software.dna</i> .
<b>Name</b>	Specify the name of the service template.
<b>Vendor</b> <sup>#2</sup>	Specify the name of the vendor who created the service template.
<b>Description</b>	Specify a description of the service template.
<b>Category</b>	Specify a category name for the service template.

#1

You cannot change the values specified in the **ID**, **Version**, and **Vendor ID** fields after you create or copy the service template. These three parameters together ensure that the service template can be uniquely identified within the JP1/AO system.

#2

If you omit this parameter, the value specified in **Vendor ID** is set as the **Vendor**. For a development service template, the **Vendor** field remains blank in the **Editor** window.

### Related topics

- [2.4.2 Uniqueness of service templates and plug-ins](#)

## 2.2.3 Procedure for setting service definition information

In the **Edit Service Definition** dialog box, set definition information such as the service template name and vendor name, in addition to property groups, input properties, output properties, and variables.

### If the service template editing view is not displayed:

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Edit**.
2. Select a service template in the **Service Template List** window, and then click **Edit**.

### To set service template definition information:

1. In the service template editing view, click the **Edit Service Definition** button.
2. Set service definition information in the **Edit Service Definition** dialog box.
3. Click **OK**.



### Important note

If another user has performed an intervening action while you were editing the service template, the save process might fail.

### Related topics

- [2.2.4 Parameters to set in service definition information](#)
- [2.2.5 Example of mapping parameter definition and flow of data](#)
- [2.1.4 Behavior when an intervening action occurs in the \*\*Editor\*\* window](#)

## 2.2.4 Parameters to set in service definition information

In the **Edit Service Definition** dialog box, set the service information, property group, input properties, output properties, and variables.

Use service properties (input properties and output properties) to specify the parameters the service needs to run, and to acquire the execution results of the service.

Table 2–3: Parameters set in the **Edit Service Definition** dialog box

Parameter	Description
Service information	Set the service template name, vendor name, description, category name, and custom file.
Property groups	You can use property groups to manage groups of properties in the <b>Service Definition</b> dialog box and the <b>Submit Service</b> dialog box. You can define a maximum of five property groups per service template.
Input properties <sup>#</sup>	Properties that store the input values required for service execution.
Output properties <sup>#</sup>	Properties that store the execution results of the service
Variables <sup>#</sup>	Variables that temporarily hold values to be passed between plug-ins.

#

In total, you can define a maximum of 100 input properties, output properties, and variables per service template.

### Related topics for service information

- [2.2.2 Parameters to set when creating or copying service templates](#)
- [2.2.6 Procedure for setting custom files](#)
- [2.2.7 Overview of custom files](#)
- [2.2.8 Format of custom files](#)

### Related topics for input properties, output properties, and variables

- [2.2.9 Properties defined in services \(service properties\)](#)

### Related topics for input properties

- [2.2.10 Adding Service Share Properties](#)
- [2.2.11 Overview of Service Share Properties](#)
- [2.2.12 Notes on defining Service Share Properties](#)
- [2.2.13 Overview of shared built-in service properties](#)
- [2.2.14 Property visibility](#)

## 2.2.5 Example of mapping parameter definition and flow of data

The process of defining mapping parameters links the input and output properties of a service to the input and output properties and variables of a plug-in. By defining mapping parameters, you can execute processing that uses a value specified at service execution as a value of the input property of a plug-in. You can also assign the output of a plug-in to an output property of a service template or a variable when the plug-in finishes processing.

The following describes an example of mapping parameter definitions, and explains what kind of data is exchanged between properties.

### Conditions

The following describes an example of creating a service template that executes certain plug-ins and is defined as shown below.

- The properties of the plug-ins are defined as follows:

**Table 2–4: Definition of plug-in A**

Property type	Property name
Input property	Capacity
	Execution target server
Output property	Output result of plug-in

**Table 2–5: Definition of plug-in B**

Property type	Property name
Input property	Execution-target server
	Storage folder path

- The operator specifies the value of the Execution-target server and Storage folder path properties when he or she submits the service for execution.
- The value of the Capacity property is fixed at 10 and does not need to be changed.
- After the service has been executed, you can check the output result of plug-in A in the **Task Details** dialog box.

**Definition**

- The service is defined as follows in the **Edit Service Definition** dialog box:

**Table 2–6: Definition in Edit Service Definition dialog box**

Property type	Property name
Input property	Server name
	Storage folder path
Output property	Service execution results

- Add plug-in A and plug-in B to the flow as steps. Then, define the steps as follows in the **Edit Step** dialog box:

**Table 2–7: Definition of step A**

Property name	Mapping parameter
Capacity	10
Execution-target server	Server name property key
Plug-in output results	Service execution results property key

**Table 2–8: Definition of step B**

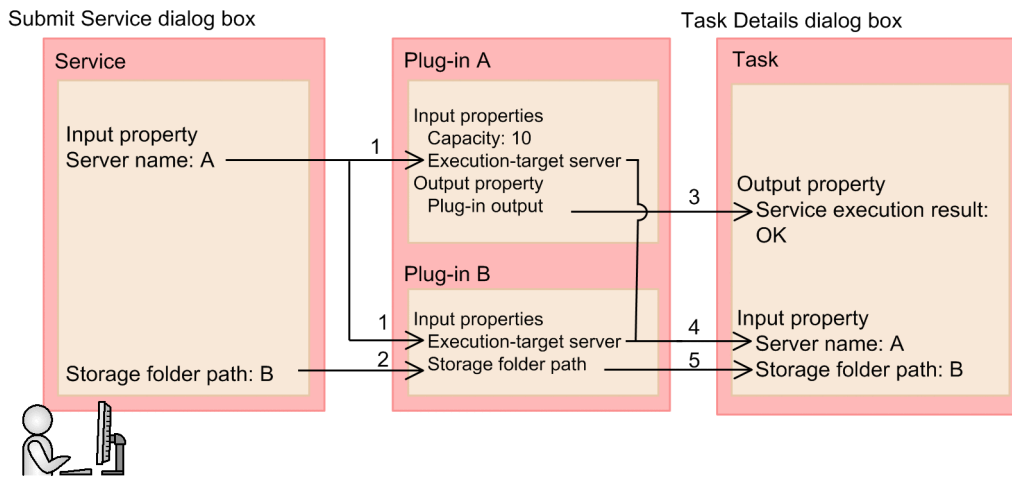
Property name	Mapping parameter
Execution-target server	Server name property key
Storage folder path	Storage folder path property key

The properties defined in the **Edit Service Definition** dialog box are mapped to the execution-target server, plug-in output results, and storage folder path.

**Flow of data**

The following figure shows the flow of data between properties:

Figure 2–6: Flow of data between properties



1. The value specified for the Server name parameter in the **Submit Service** dialog box is stored as the value of the Execution-target server parameter for plug-in A and plug-in B.
2. The value specified for the Storage folder path parameter in the **Submit Service** dialog box is stored as the value of the Storage folder path parameter for plug-in B.
3. The value of the Plug-in output results parameter of plug-in A appears for the Service execution result property in the **Task Details** dialog box.
4. The value of the Execution-target server parameter of plug-in A and plug-in B appears in the Server name field of the **Task Details** dialog box.
5. The value of the Storage folder path parameter of plug-in B appears for the Storage folder path property in the **Task Details** dialog box.

## 2.2.6 Procedure for setting custom files

You can define the contents of the **Service Details** window by selecting a custom file in the **Edit Service Definition** dialog box. A custom file can consist of one file or several. For example, if you want to include images and links in the dialog box, you can archive several files or folders in zip format and register the archive. The procedure for setting a custom file differs depending on whether it consists of one file or several.

Note that you can assign a different custom file at a later stage if circumstances change.

### To set a single file as a custom file:

1. In the **Edit Service Definition** dialog box, click the **Select** button in the **Custom Files** area.
2. Select the file you want to specify as the custom file.

### To set multiple files as a custom file:

1. Place the files you created in *any-folder\webrout\language-code\*.  
As *language-code*, you can specify a two-character language code (ja, en, or zh) as defined in ISO-639.
2. Archive the contents of the *language-code* folder to a file in .zip format.

3. In the **Edit Service Definition** dialog box, click the **Select** button in the **Custom Files** area.
4. Select the file (.zip) you want to specify as the custom file.
5. In the **Custom File Name for Service Details Dialog** field, enter the path of the file you want to use to define the contents of the **Service Details** window.  
Specify a relative path whose current directory is the location where the archive will be extracted. Use forward slashes to delimit the path.

### Tip

If the file has the extension the .zip, it is automatically renamed to `webroot.zip` when you build or release the service template.

### To change the window contents based on the Web browser locale:

1. Store the file or files you created in *any-folder*\webroot\*language-code*.  
As the language code, you can specify a two-character language code as defined in ISO-639 (ja, en, or zh) in lower-case letters. You can also store a file directly under the webroot folder. This file is used when either of the following conditions is met:
  - There is no language code folder that corresponds to the locale of the Web browser.
  - There are no files in the language code folder that correspond to the locale of the Web browser.

### Important note

- The file stored directly under the webroot folder must have the same name as the files in the language code folders.
- Even if you do not store a file directly under the webroot folder, the files in each language code folder must have the same name.

2. Archive the contents of the webroot folder to a file in .zip format.
3. In the **Edit Service Definition** dialog box, click the **Select** button in the **Custom Files** area.
4. Select the .zip file you want to specify as the custom file.
5. In the **Custom File Name for Service Details** field, enter the file name you want to set.

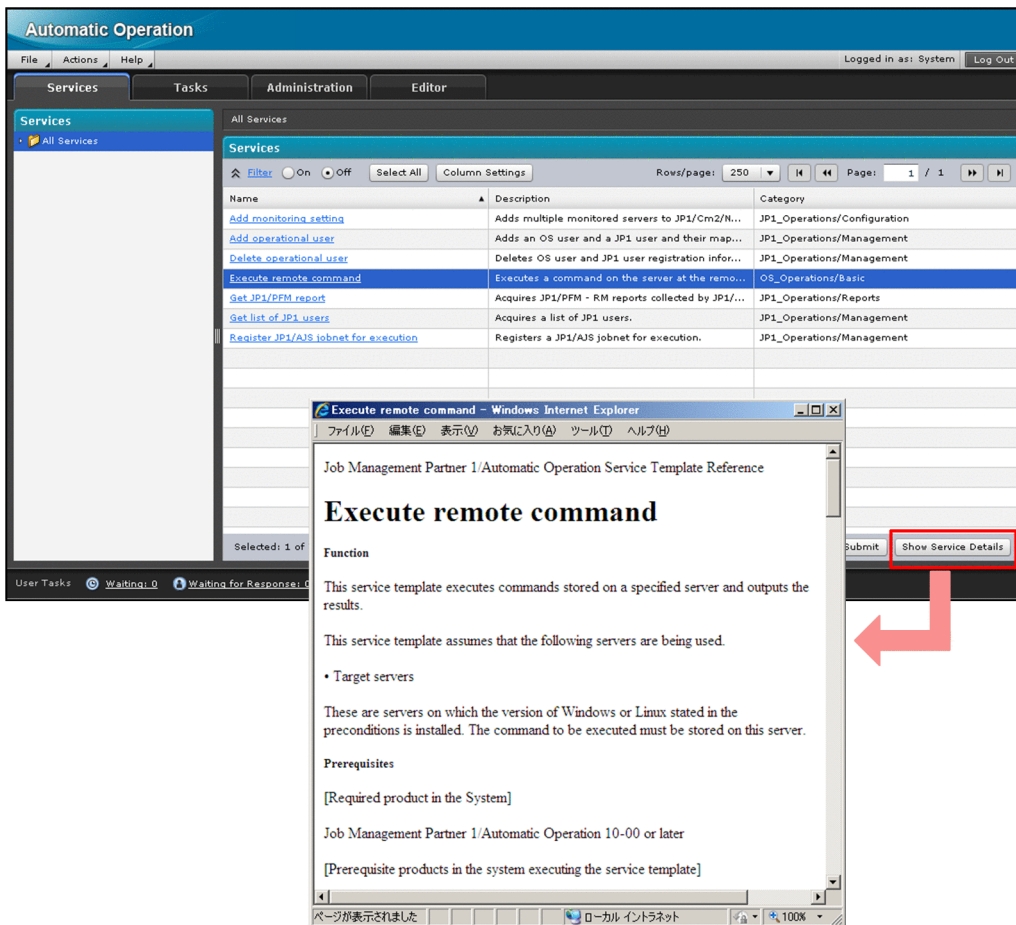
### Related topics

- [2.2.7 Overview of custom files](#)

## 2.2.7 Overview of custom files

A custom file is a file that defines the contents of the **Service Details** window. You can select which information appears in the **Service Details** window by assigning a custom file in the **Edit Service Definition** dialog box. The following figure shows an example of the information displayed in the **Service Details** dialog box:

Figure 2–7: Example of **Service Details** window



## Related topics

- 2.2.6 Procedure for setting custom files
- 2.2.8 Format of custom files

## 2.2.8 Format of custom files

Create custom files as static content to be executed in a Web browser.

### Specifiable extensions

The following are some of the extensions you can specify for custom files:

- .html
- .js
- .css
- .swf
- .jpeg

JP1/AO product support does not extend to dynamic content such as .jsp and .war> files that run on an application server.



## Characters specifiable in file names and paths

Use ASCII characters in the file names and paths of custom files. You cannot use the following characters:

- Multi-byte characters
- Control characters ('\u0000' to '\u001F' and '\u007F' to '\u009F')
- Question marks (?), asterisks (\*), double quotation marks ("), right angle brackets (>), left angle brackets (<), vertical bars (|), and colons (:)

## 2.2.9 Properties defined in services (service properties)

By defining properties for a service, you can specify the parameters a service needs while executing, and collect the execution results of the service. You can define the following types of service property:

### Input properties

Properties that store the input values services need when executing. You can define a custom property or use a Service Share Property. In input properties, in addition to property keys, property names, and other information of this nature, you can specify whether a property is visible. You can define input properties in the following dialog boxes:

- **Create Input Property for Service** dialog box
- **Edit Input Property for Service** dialog box

### Output properties

Properties that store the execution results of a service. You can define output properties in the following dialog boxes:

- **Create Output Property for Service** dialog box
- **Edit Output Property for Service** dialog box

### Variables

A variable temporarily holds a value that passes between plug-ins. You can define variables in the following dialog boxes:

- **Create Variable** dialog box
- **Edit Variable** dialog box

Input properties and output properties are displayed as service properties in the following dialog boxes:

- **Service Definition** dialog box
- **Submit Service** dialog box
- **Task Details** dialog box
- **Service Share Properties** view (for Service Share Properties)

A service input property or output property can store a maximum of 1,024 characters.

## Mapping input and output properties

You can map service properties to plug-in properties in the **Specify Plug-in Input Properties for Mapping** dialog box or the **Specify Plug-in Output Properties for Mapping** dialog box when editing a step. This process allows you to assign a value entered by an operator to a plug-in property, or display the value of a plug-in property on screen.

## Mapping variables

When editing a step, you can use the **Specify Plug-in Output Properties for Mapping** dialog box to map an output property of a plug-in to a variable that can pass the value to another plug-in. In the **Specify Plug-in Input Properties for Mapping** dialog box, you can map a variable to an input property of a plug-in to be executed as a later step.

### Related topics

- [2.2.11 Overview of Service Share Properties](#)
- [2.2.13 Overview of shared built-in service properties](#)
- [2.2.14 Property visibility](#)

## 2.2.10 Adding Service Share Properties

You can share properties between services by adding a Service Share Property as a property of a service template. You can also use Service Share Properties defined in JP1/AO in advance (called shared built-in service properties), or add shared properties associated with service templates you have imported.

### To add a new Service Share Property:

1. In the **Edit Service Definition** dialog box, in the **Input Properties** area, click **Add**.
2. In the **Create Input Property for Service** dialog box, in the **Scope** area, select the **Service Share Property** check box.

### To add an existing Service Share Property:

1. In the **Edit Service Definition** dialog box, in the **Input Properties** area, click the **Select Service Share Property** button.
2. In the **Select Service Share Property** dialog box, select the Service Share Property you want to add, and then click **OK**.

### Related topics

- [2.2.11 Overview of Service Share Properties](#)
- [2.2.12 Notes on defining Service Share Properties](#)
- [2.2.13 Overview of shared built-in service properties](#)

## 2.2.11 Overview of Service Share Properties

Service Share Properties are properties that are shared by more than one service, each of which can view and update its value.

You can define a service property as a Service Share Property by selecting the **Service Share Properties** check box in the **Scope** area of the **Create Input Property for Service** dialog box or the **Edit Input Property for Service** dialog box. Any service can use the same property key to reference and update the Shared Service Property.

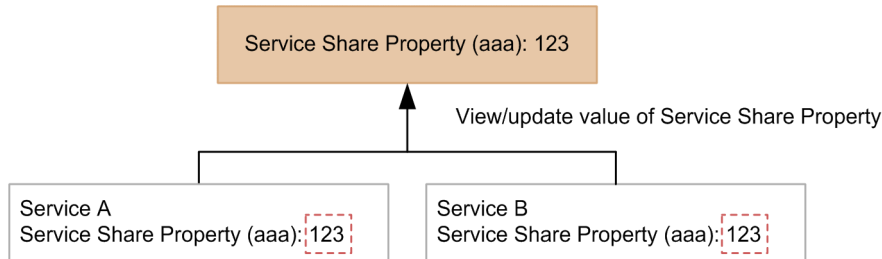
A Service Share Property referenced by a service or task returns a value maintained by the system.

Because the value of a Service Share Property is the same system-wide, you cannot assign a different value for different resource groups. Even if you add multiple services from a service template and assign each to a different resource group, the value of the Service Share Property is shared among the resource groups.

You can assign values to properties in the **Service Definition** dialog box, the **Submit Service** dialog box, and in the **Service Share Properties** view. Note that the value assigned to a property in the **Submit Service** dialog box only applies to tasks generated from that service. For this reason, the value you specify in the **Submit Service** dialog box does not affect the value of the Service Share Property as referenced by other services. Also, after a service is submitted for execution, changing the property value in the **Service Share Properties** view or other dialog boxes does not affect the property value in the running service.

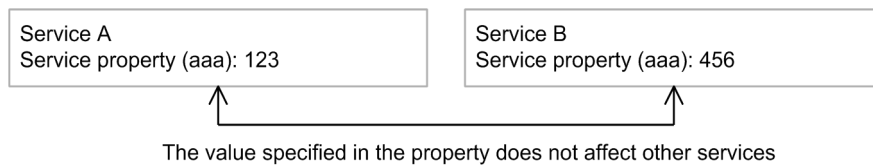
The following figure shows the valid range of a Service Share Property:

Figure 2–8: Valid range of Service Share Property



The following figure shows the valid range of an ordinary (non-shared) service property:

Figure 2–9: Valid range of ordinary (non-shared) service property



If you do not select the **Service Share Properties** check box in the **Scope** area, the value specified for the property is only valid in the context of that service. You can assign property values in the **Service Definition** dialog box and the **Submit Service** dialog box.

Shared service properties that are predefined by JP1/AO are called shared built-in service properties.

## Related topics

- [2.2.13 Overview of shared built-in service properties](#)

## 2.2.12 Notes on defining Service Share Properties

Note the following when defining Service Share Properties:

### Assigning values to Service Share Properties

- The name and description you specify for properties in the **Edit Service Definition** dialog box appear in the **Service Definition** dialog box and the **Submit Service** dialog box.
- Create a property key that is unique within the system by specifying the domain name in reverse order from the top level as a period-separated value. If you choose not to use domain names as property keys, make sure that the property key you specify is not being used for another property whose value you do not want shared.

- If you need to change a parameter of a Service Share Property other than the property name or description, use the following procedure:
  1. Delete from JP1/AO all service templates that include the Service Share Property you want to modify.
  2. Add the service templates to JP1/AO with the new values specified for the Service Share Property.
 The value of the Service Share Property is now changed.
- The following describes what happens when a Service Share Property with the same key as an existing Service Share Property is assigned to a service template:
  - Initially, the Service Share Property is assigned the value specified in the **Default Value** field for the Service Share Property when the service template was built, released, or imported by the `importservicetemplate` command. You can then specify a value for the property in the **Service Share Properties** view, the **Service Definition** dialog box, or the **Submit Service** dialog box.
  - The property name, description, and default value can differ from that of the existing Service Share Property. However, if a parameter other than the property name, description, or default value differs, an error occurs when you attempt to build, release, or import the service template.
  - When a value differs from that of the existing Service Share Property, the specified value only appears in windows and dialog boxes that display the property name or description as a property of the service template.
- Service Share Properties selected in the **Select Service Share Property** dialog box are added with no default value specified. Set the default value in the **Edit Input Property for Service** dialog box.

### Assigning a Service Share Property to a property group

- You can assign Service Share Properties to any property group when defining a service template. The same Service Share Property can belong to a different property group in different service templates.

## 2.2.13 Overview of shared built-in service properties

Shared built-in service properties are properties that are defined in advance in the JP1/AO system. Unlike other properties, shared built-in service properties are not tied to a specific service or task. JP1/AO functions that reference shared built-in service properties use the value assigned to the property when they execute.

You can also define a shared built-in service property as a Service Share Property of a service, and reference it from within a task. Services can reference the values of these properties in the same manner as other Service Share Properties.

You can define a shared built-in service property as a Service Share Property of a service in the **Edit Service Definition** dialog box. When you click the **Select Service Share Property** button in the **Edit Service Definition** dialog box, a list of shared built-in service properties appears.

The following table lists the shared built-in service properties provided in JP1/AO:

Table 2–9: List of shared built-in service properties

No.	Property key	Property name <sup>#</sup>	Description <sup>#</sup>
1	com.hitachi.software.dna.sys.mail.notify	Email notification	<b>Enables or disables the email notification functionality. (Built-in shared service property)</b>
2	com.hitachi.software.dna.sys.mail.smtp.server	SMTP server address	<b>Specifies the SMTP server address. The address can be specified as an IPv4 or IPv6 address, or as a host name. Only one of the above can be specified. Multiple addresses cannot be specified by separating them with commas. (Built-in shared service property)</b>
3	com.hitachi.software.dna.sys.mail.smtp.port	SMTP server port number	<b>Specifies the SMTP server port number. (Built-in shared service property)</b>

No.	Property key	Property name <sup>#</sup>	Description <sup>#</sup>
4	com.hitachi.software.dna.sys.mail.smtp.userid	SMTP server user ID	<b>Specifies the user ID of the user who logs in to the SMTP server. (Built-in shared service property)</b>
5	com.hitachi.software.dna.sys.mail.smtp.password	SMTP server password	<b>Specifies the password of the user who logs in to the SMTP server. (Built-in shared service property)</b>
6	com.hitachi.software.dna.sys.mail.from	Notification email sender	<b>Specifies the sender of notification emails. (Built-in shared service property)</b>
7	com.hitachi.software.dna.sys.mail.to	Notification email recipients (To)	<b>Specifies the "To" recipients of notification emails. Multiple email addresses can be specified by separating them with commas. (Built-in shared service property)</b>
8	com.hitachi.software.dna.sys.mail.cc	Notification email recipients (Cc)	<b>Specifies the "Cc" recipients of notification emails. Multiple email addresses can be specified by separating them with commas. (Built-in shared service property)</b>
9	com.hitachi.software.dna.sys.mail.bcc	Notification email recipients (Bcc)	<b>Specifies the "Bcc" recipients of notification emails. Multiple email addresses can be specified by separating them with commas. (Built-in shared service property)</b>
10	com.hitachi.software.dna.sys.jp1.username	JP1 user name	<b>Specifies the name of the JP1 user who executes services. (Built-in shared service property)</b>
11	com.hitachi.software.dna.sys.jp1.password	JP1 user password	<b>Specifies the password of the JP1 user who executes services. (Built-in shared service property)</b>
12	com.hitachi.software.dna.sys.task.log.level	Task log output level	<b>Specifies the level of messages output to the task log. (Built-in shared service property)</b>
13	com.hitachi.software.dna.sys.ssh.privatekey.passphrase	Pass phrase of the private key (for SSH public key authentication)	<b>Specifies the pass phrase of the private key used for SSH public key authentication. (Built-in shared service property)</b>

#

You can change the contents of the **Service Definition** dialog box and the **Submit Service** dialog box by entering a property name and description of your choice for the built-in shared service property. However, you cannot change the information displayed in the **Service Share Properties** view from its initial state at installation.

The following table shows detailed information about each property:

Table 2–10: Detailed information about shared built-in service properties

No.	Property key	Data type	Default value	Required?	Length		Specifiable characters	Values in list
					Minimum	Maximum		
1	com.hitachi.software.dna.sys.mail.notify	boolean	false	true	--	--	--	--
2	com.hitachi.software.dna.sys.mail.smtp.server	string	--	false	0	255	No restrictions	--
3	com.hitachi.software.dna.sys.mail.smtp.port	integer	25	false	--	--	--	--

No.	Property key	Data type	Default value	Required?	Length		Specifiable characters	Values in list
					Minimum	Maximum		
4	com.hitachi.software.dna.sys.mail.smtp.userid	string	--	false	0	255	No restrictions	--
5	com.hitachi.software.dna.sys.mail.smtp.password	password	--	false	0	1,024	No restrictions	--
6	com.hitachi.software.dna.sys.mail.from	string	--	false	0	255	No restrictions	--
7	com.hitachi.software.dna.sys.mail.to	string	--	false	0	255	No restrictions	--
8	com.hitachi.software.dna.sys.mail.cc	string	--	false	0	255	No restrictions	--
9	com.hitachi.software.dna.sys.mail.bcc	string	--	false	0	255	No restrictions	--
10	com.hitachi.software.dna.sys.jp1.username	string	Value specified by user at installation	true	1	31	^[a-zA-Z0-9!#\$%&-\.\@_~]+\$	--
11	com.hitachi.software.dna.sys.jp1.password	password	Value specified by user at installation	true	6	32	^[a-zA-Z0-9!#\$%&'()\*!+,;=<=>@\[\]^_`{\ }~]+\$	--
12	com.hitachi.software.dna.sys.task.log.level	list	10	true	--	--	--	0,10,20,30,40
13	com.hitachi.software.dna.sys.ssh.privatekey.passphrase	password	--	false	0	1,024	No restrictions	--

Legend:

--: Not applicable.

## 2.2.14 Property visibility

Visibility is a parameter of input service properties.

By specifying the visibility of a service property, you can control whether the property is displayed in the JP1/AO interface. The following windows and dialog boxes display properties in JP1/AO:

- **Service Definition** dialog box
- **Submit Service** dialog box
- **Task Details** dialog box
- **Service Share Properties** view

For example, you can prevent the **Submit Service** dialog box from displaying properties that are not relevant to the user.

You can set property visibility in the **Edit Input Property for Service** dialog box and the **Create Input Property for the Service** dialog box. You can specify **config** or **exec** as the visibility setting.

**config**

Specify this setting for properties that you want to appear as input items in the **Service Definition** dialog box.

**exec**

Specify this setting for properties that you want to appear as input items in the **Service Definition** dialog box and the **Submit Service** dialog box.

You can check the property values specified at service execution in the **Task Details** dialog box, and set property values in the **Service Share Properties** view if the property is defined as a Service Share Property.

Which properties are displayed in each JP1/AO window and dialog box depends on the visibility setting and the permission of the logged-in user.

Table 2–11: Appearance in the **Service Definition** dialog box

Service property type	Visibility	Visibility by role of logged-in user		
		Admin role, Develop role	Modify role	Submit role
Not a Service Share Property	config	Y	Y	N
	exec	Y	Y	N
Service Share Property	config	Y	Y	N
	exec	Y	Y	N

Legend:

Y: Displayed. N: Not displayed.

Table 2–12: Appearance in the **Submit Service** dialog box

Service property type	Visibility	Visibility by role of logged-in user		
		Admin role, Develop role	Modify role	Submit role
Not a Service Share Property	config	N	N	N
	exec	Y	Y	Y
Service Share Property	config	N	N	N
	exec	Y	Y	Y

Legend:

Y: Displayed. N: Not displayed.

Table 2–13: Appearance in the **Task Details** dialog box

Service property type	Visibility	Visibility by role of logged-in user		
		Admin role, Develop role	Modify role	Submit role
Not a Service Share Property	config	Y	Y	N
	exec	Y	Y	Y

Service property type	Visibility	Visibility by role of logged-in user		
		Admin role, Develop role	Modify role	Submit role
Service Share Property	config	Y	Y	N
	exec	Y	Y	Y

Legend:

Y: Displayed. N: Not displayed.

**Table 2–14: Appearance in the Service Share Properties view**

Service property type	Visibility	Visibility by role of logged-in user		
		Admin role, Develop role	Modify role	Submit role
Not a Service Share Property	config	N	N	N
	exec	N	N	N
Service Share Property	config	Y	N	N
	exec	Y	N	N

Legend:

Y: Displayed. N: Not displayed.

## 2.2.15 Viewing another service template during editing

You can view the settings of a development service template or release service template in another window while you edit a service template. Because the reference window is opened as read-only, there is no risk of the user inadvertently changing the other template.

### To open a service template for reference purposes from the Editor window:

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **View**.
2. In the **Service Template List** dialog box, select the service template you want to view, and then click **View**.  
The service template you selected appears in a new window.

### To open a service template in another window during editing:

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Edit**.
2. In the **Service Template List** dialog box, select the service template you want to edit, and then click **Edit**.
3. From the **Actions** pull-down menu, choose **View**.
4. In the **Service Template List** dialog box, select the service template you want to view, and then click **View**.  
The service template you selected appears in a new window.



## 2.3 Displaying a list of service templates

---

### 2.3.1 Procedure for displaying a list of service templates

When you edit, copy, view, or delete a service template, you can choose from a list of service templates.

#### To display a list of service templates:

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Edit**, **Copy**, **View**, or **Delete**.

The **Service Template List** dialog box appears.

A list of development service templates and release service templates appears in a new tab. Note that if you click **Edit** or **Delete**, the window that appears only displays development service templates.

Operations you can perform in the **Service Template List** dialog box include specifying how many columns to display, and whether to only display service templates created with the latest version of JP1/AO.

#### Related topics

- [2.4.2 Uniqueness of service templates and plug-ins](#)

## 2.4 Copying service templates

---

### 2.4.1 Procedure for copying service templates

You can copy a development service template or release service template and create a new development service template that retains the settings of the original. You can use this procedure when developing a new service template based on an existing service template, or when creating an upgraded version of an existing service template.

#### To copy a service template:

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Copy**.
2. In the **Service Template List** dialog box, select the service template you want to copy, and then click **Copy**.
3. In the **Copy Service Template** dialog box, set the definition information for the service template, and then click **Save**.
4. In the **Information** dialog box, click **OK**.

The service template is copied, and the service template editing view appears.

You cannot copy a service template without changing at least one of the vendor ID, service template ID, and service template version. These three parameters together ensure that the service template can be uniquely identified within the JP1/AO system. You cannot specify a vendor ID that begins with `com.hitachi.software.dna`. This string is reserved in JP1/AO. If the vendor ID of the service template you are copying begins with `com.hitachi.software.dna`, the vendor ID and vendor name are deleted when you copy the template.

#### Related topics

- [2.2.2 Parameters to set when creating or copying service templates](#)
- [2.4.2 Uniqueness of service templates and plug-ins](#)
- Adding preset properties in the *Job Management Partner 1/Automatic Operation Administration Guide*

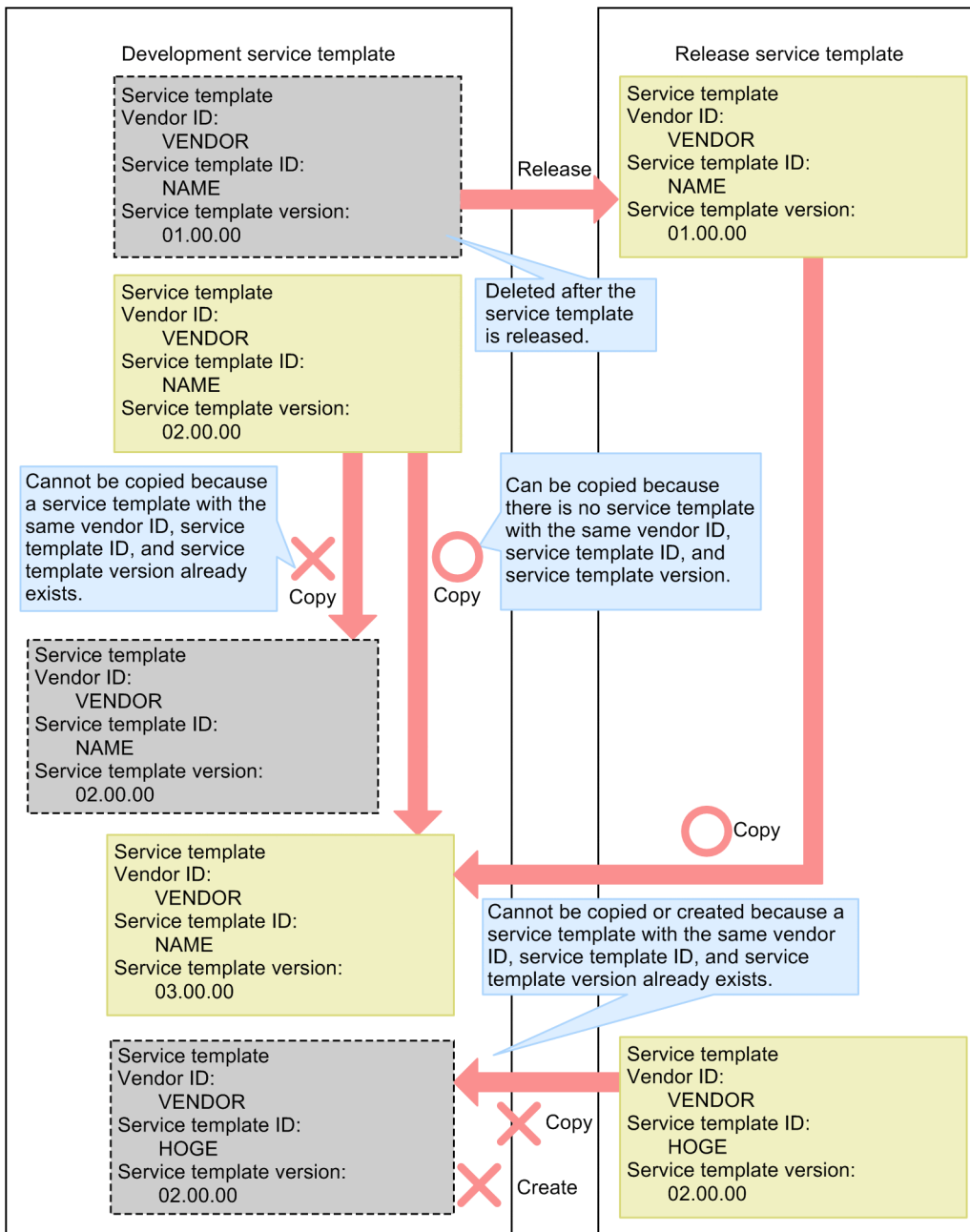
### 2.4.2 Uniqueness of service templates and plug-ins

The following three parameters ensure that service templates and plug-ins can be uniquely identified within the JP1/AO system:

- Vendor ID
- Service template ID (or plug-in ID)
- Service template version (or plug-in version)

These three parameters are centrally managed with no distinction made between development service templates (development plug-ins) and release service templates (release plug-ins). For this reason, you cannot create or copy a service template or plug-in whose vendor ID, service template ID (plug-in ID), and service template version (plug-in version) all match those of an existing release service template (or release plug-in).

Figure 2–10: Managing service template versions



## 2.5 Deleting service templates

---

### 2.5.1 Procedure for deleting development service templates

When you delete a development service template, the definition of the service template is deleted from the JP1/AO server and from the list under **Service Template List**.

#### Important note

You can delete a service template that another user is editing. Make sure that the service template is not being edited by another user before you delete it.

#### To delete a development service template:

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Delete**.
2. In the **Service Template** dialog box, select the service template you want to delete, and then click **Delete**.
3. In the confirmation dialog box, click **OK**.

The system does not delete the development plug-ins and release plug-ins used by the development service template you are deleting, regardless of whether they are used by another service template. If a development plug-in or release plug-in is no longer needed, you can delete it individually.

JP1/AO deletes any services that were created by building the development service template you are deleting, and archives any tasks generated from those services. If a task is in progress, an error occurs and the deletion process fails. If any debug services or debug tasks created while debugging the development service template remain in the system, those services and tasks are deleted. If a debug task is still in progress, an error occurs and the deletion process fails.

If the development service template you are deleting has already been released by another user, an error occurs and the deletion process fails.

#### Related topics

- [2.5.2 Procedure for deleting release service templates](#)
- [4.4.1 Procedure for deleting plug-ins](#)
- [2.1.4 Behavior when an intervening action occurs in the \*\*Editor\*\* window](#)

### 2.5.2 Procedure for deleting release service templates

To delete a release service template, execute the `deleteservicetemplate` command.

#### Related topics

- [Deleting service templates in the \*Job Management Partner 1/Automatic Operation Administration Guide\*](#)

## 2.6 Setting display information for service templates in resource files

### 2.6.1 Procedure for setting service resource files

You can define the information displayed in a service template by assigning a service resource file. You can set display information for service templates and steps. You can edit a service resource file by downloading the file and overwriting its contents.

Note that you cannot change the information displayed for a service template that has already been released unless you copy the release service template and edit it as a development service template.

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Edit**.
2. In the **Service Template List** dialog box, select a service template and click **Edit**.
3. From the **Actions** pull-down menu, choose **Set Resources**.
4. In the **Set the Service Resources** dialog box, click the link for the service resource file and download the file.
5. Edit the definitions in the service resource file you downloaded.  
Do not change the file name from `service_ language-code.properties.txt`. If you change the file name, an error occurs when you attempt to upload the file.  
As *language-code*, you can specify a two-character language code (ja, en, or zh) as defined in ISO-639.
6. Click the **Refresh** button, select the service resource file you edited, and upload the file.
7. In the confirmation dialog box, click **OK**.

#### Important note

When you upload the service resource file, the existing file is overwritten with the contents of the new file. Take care not to upload the wrong file.

#### Related topics

- [2.4.1 Procedure for copying service templates](#)
- [2.6.2 Format of service resource file](#)
- [2.6.3 Definitions in service resource files](#)

### 2.6.2 Format of service resource file

A service resource file defines the information displayed in the JP1/AO user interface. The format of the file is described below. Note that the content of the service resource file depends on the type of display elements you are defining. For details about how to define each type of display element, see [2.6.3 Definitions in service resource files](#).

- The file name of the service resource file is `service_ language-code.properties.txt`.  
As *language-code*, you can specify a two-character language code (ja, en, or zh) as defined in ISO-639.
- Define the file contents in the format *property-keydelimiting-charactersetting-value*. As the delimiting character, you can use an equals sign =, a colon (:), tab characters (\t), or a single-byte space.

- Enter one property key and setting per line.
- Property keys can contain the following characters:
  - Single-byte alphanumeric characters
  - Single-byte hyphens (-)
  - Single-byte underscores (\_)
  - Single-byte periods (.)
- Characters must be encoded in UTF-8.
- If you define the same property key in the file more than once, the value of the last occurrence of the property key applies.
- Lines that begin with a hash mark (#) are handled as comments.
- Property keys are case sensitive.
- To specify a character string that contains a forward slash (\), specify two forward slashes (\\) instead.
- Lines that consist only of single-byte spaces are ignored.
- On each line of the service resource file, the property key is the character string from the first character that is not a single-byte space to the character immediately preceding the first delimiting character.
- The setting value is the string from the first non-delimiting character after the delimiting character following the property key to the last character in the line.  
 For example, the following line in the service resource file represents the property key abc with the setting value =\tc:
 

```
abc\t=\tc
```

 However, if the character immediately following the first delimiting character is = or :, the setting value is the character string from the next character that is not a single-byte space or tab character (\t), to the end of the line.  
 For example, the following line in the service resource file represents the property key abc with the setting value =\tc.
 

```
abc\t=\t=\tc
```
- You cannot use surrogate pair characters.

## 2.6.3 Definitions in service resource files

The definitions in the service resource file depend on the type of display items you are setting. You can specify the following definitions in the service resource file:

Setting items displayed in the **Edit Service Definition** dialog box

Define entries in the following format to set the property names, descriptions, and other information displayed in the **Edit Service Definition** dialog box.

*property-keydelimiting-charactersetting-value*

Property keys can be 1 to 128 characters long.

As the delimiting character, you can use an equals sign =, a colon (:), a tab character (\t), or a single-byte space.

For example, enter a definition in the format `service.displayName=TestService`.

When specifying display elements for a step on the highest hierarchical level of a flow

Define entries as follows to set the step name and comment displayed in the **Step Details** area of the **Task Details** dialog box.

### Setting a step name

To set a step name in a service template, enter a definition in the following format:

`dnajob.step-ID.displayNamesetting-value`

For example, enter a definition in the format `dnajob.teststep.displayName=TestStep`.

### Setting a step description

To set a description of a step, enter a definition in the following format:

`dnajob.step-ID.commentsetting-value`

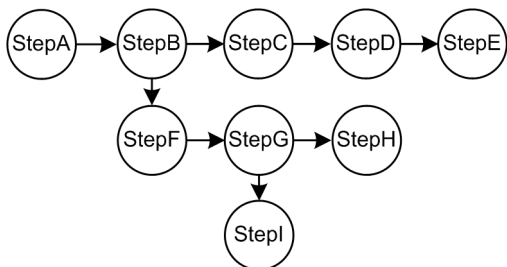
For example, enter a definition in the format `dnajob.teststep.comment=This is a test step..`

When specifying display elements for a step that is not on the highest hierarchical level of a flow

Specify the step IDs from the step at the highest hierarchical level to the target step, connecting them with slashes (/).

For example, to specify a resource in Step F in the figure below, connect the step ID for Step B at the highest hierarchical level to the step ID for Step F. To specify a resource in Step I, connect the step IDs of Step B, Step G, and Step I.

Figure 2–11: Example of specifying resources in Step F and Step I



The following shows how to specify resources in Step F and Step I in a service resource file.

```
dnajob.StepB/StepF.displayName=Step F
dnajob.StepB/StepF.comment=Step F description
dnajob.StepB/StepG/StepI.displayName=Step I
dnajob.StepB/StepG/StepI.comment=Step I description
```

### Related topics

- [2.6.4 Correspondence between information displayed in service templates and properties in service resource files](#)

## 2.6.4 Correspondence between information displayed in service templates and properties in service resource files

The display information for service templates managed in the **Editor** window can be set in the **Editor** window itself. The display information you set in the Editor window is also defined in the resource file for the service. The following table lists the correspondence between the information displayed for a service template and the properties in the service resource file.

Table 2–15: Correspondence between information displayed for service templates and properties in service resource files

Information displayed for service template	Property in service resource file
Vendor name	service.vendorDisplayName

Information displayed for service template	Property in service resource file
Service template name	service.displayName
Service template description	service.shortDescription
Property group name	propertyGroup. <i>property-group-ID</i> .displayName
Property group description	propertyGroup. <i>property-group-ID</i> .description
Name of input property, output property, or variable	property. <i>property-key</i> .displayName
Description of input property, output property, or variable	property. <i>property-key</i> .description
Step name	dnajob. <i>step-ID</i> .displayName#
Step description	dnajob. <i>step-ID</i> .comment#

#

To set a resource for a step that is not at the highest hierarchical level of a flow, specify the step IDs from the step at the highest hierarchical level to the target step, connecting them with slashes (/).

The step name and description both appear in the **Step Details** area of the **Task Details** dialog box. However, this information is only displayed for steps in the top-level flow. It does not appear in the **Step Details** area of the **Task Details** dialog box for steps that are outside the top-level flow or not at the top-level in the hierarchy.

## 2.6.5 Service resource files automatically generated when a service template is created

When a service template is created, JP1/AO automatically generates two service resource files. One is for the same language as the Web browser locale, and the other is for English.

The following table lists the values set in these automatically generated files.

Table 2–16: Default values for display information in service resource files (at service template creation)

Defined display information	Value set by default	
	Same language as Web browser locale	Automatically generated English language resource file#
Vendor name	The value specified in the <b>Editor</b> window	Vendor ID
Service template name		Service template ID
Service template description		Blank

#

For the contents of the service resource file generated when the Web browser locale is English, see the "Same language as the Web browser locale" column.

Examples of service resource files automatically generated when creating a service template are shown below.

### Values specified in JP1/AO interface

```
Vendor ID: test.vendor
```



```
Service template ID: test.service
Service template version: 10.00.00
Service template name: test.template
Vendor name: test.vendor
Description: This service template is for testing purposes.
```

### Generated service resource file (Japanese)

```
service.vendorDisplayName=テスト用
service.displayName=テストサービステンプレート
service.shortDescription=テスト用サービステンプレートです
```

### Generated service resource file (English)

```
service.vendorDisplayName=test.vendor
service.displayName=test.service
service.shortDescription=
```

## 2.6.6 Service resource files updated when a service template is edited

When you edit and save a service template, JP1/AO updates the service resource file for the same language as the Web browser locale.

If you add or delete a definition of display information or update a property group ID, step ID, or property key, corresponding changes are made to the other service resource file for the non-Web browser locale language to ensure consistency.

The table below shows the values added to the service resource file for locales other than the Web browser locale when you add a definition of a display item. When you update the definition of a display item or update a property group ID, step ID, or property key, the values in the file are automatically overwritten with the values in the table. If you want to reference the existing value, create a backup of the service resource file for locales other than the Web browser locale.

### Important note

When the step ID of a layering step or repeated step is updated, the names and descriptions of subordinate steps are automatically overwritten with the values in the table below. Note that the overwritten values are those defined in the service resource file for locales other than the Web browser locale.

Table 2–17: Display item values set in service resource files

Defined display information	Assigned value
Property group name <sup>#1</sup>	Property group ID
Property group description <sup>#1</sup>	Blank
Name of service input property, output property, or variable <sup>#2</sup>	Property key
Description of service input property, output property, or variable <sup>#2</sup>	Blank
Step name <sup>#3</sup>	Step ID

Defined display information	Assigned value
Step description <sup>#3</sup>	Blank

#1

The value of this display item is overwritten when the property group ID is updated.

#2

The value of this display item is overwritten when the property key is updated.

#3

The value of this display item is overwritten when the step ID is updated.

When you delete the definition of a display item, the definition is also deleted from the service resource file for languages other than that of the Web browser locale.

To set display information for a language other than that of the Web browser locale, you need to manually create or edit a service resource file and then upload the file. When manually creating a service resource file, we recommend that you download and use a service resource file for a locale in which display information is already defined.

## 2.6.7 Displaying a service template in a Web browser that is set to a locale for which no service resource file is available

When you display a service template in a Web browser that is set to a locale for which no service resource file is available, the system uses the service resource file for English language locales to display the template.

# 3

## Creating and editing flows

This chapter describes how to create and edit flows. By creating and editing flows, you can change the order in which the steps are performed in an operating procedure, or add steps to an existing procedure.

## 3.1 Displaying the Flow view

### 3.1.1 Procedure for displaying the Flow view

Flows are created and edited in the **Flow** view.

#### Creating a flow with a new service template

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Create**.
2. In the **Create Service Template** dialog box, set the definition information and then click **Save**.  
When you click **OK** in the **Information** dialog box, the **Flow** view appears.

#### Editing the flow of an existing service template

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Edit**.
2. In the **Service Template List** window, select the service template you want to edit and then click **Edit**.  
The **Flow** view appears.

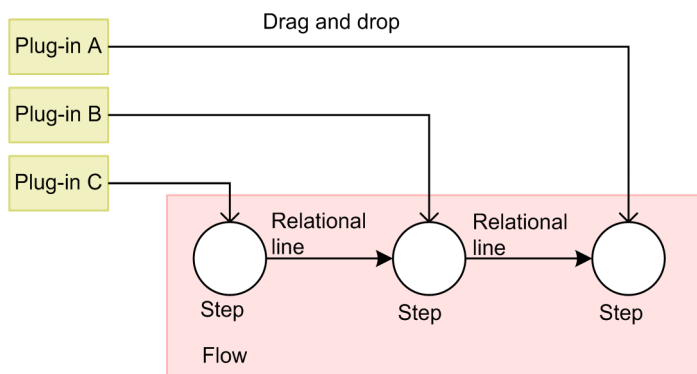
#### Related topics

- [2.2.4 Parameters to set in service definition information](#)

### 3.1.2 Relationship between flow and steps

The user creates each unit of processing in a flow by dragging plug-ins from the **Plug-in** view to the **Flow** view. Each plug-in dropped into the **Flow** view is called a *step*. A flow is created by placing the steps required to execute a task and connecting them with relational lines. The following figure shows the relationship between a flow and steps.

Figure 3–1: Relationship between flow and steps



You can also use flow plug-ins and repeated-execution plug-ins to define a flow within another flow.

#### Related topics

- [3.1.3 Creating flow hierarchies](#)

### 3.1.3 Creating flow hierarchies

A flow hierarchy is created when you define a flow within another flow. You can define a maximum of 25 hierarchical levels, with the top-level flow being level 1.

You can create a flow hierarchy by deploying flow plug-ins, and repeat a unit of processing that consists of several steps by deploying repeated-execution plug-ins.

Figure 3–2: Creating flow hierarchies

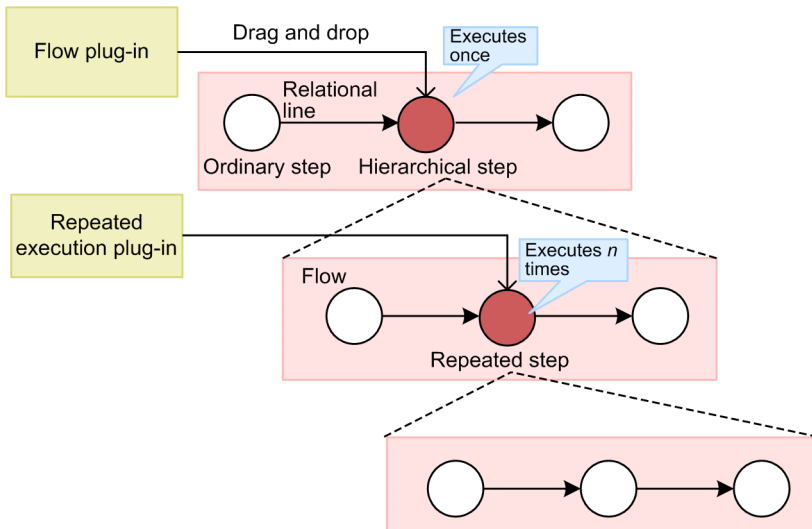


Table 3–1: Plug-in roles and their relationship to steps

Dragged & dropped plug-in	Type of step	Role
Flow plug-in	Hierarchical step	Created a flow hierarchy.
Repeated-execution plug-in	Repeated step	The system repeats execution of the specified flow. However, if you want to impose a hierarchy on a flow subordinate to the repeated step, you must use a flow plug-in. You cannot place a repeated-execution plug-in in a flow that is subordinate to a repeated step. Also, if you attempt to impose a hierarchy on a flow you have copied and pasted that includes a repeated step, an error occurs.
Other plug-ins	Ordinary step	The system executes the plug-in.

When you execute a service template that includes a hierarchical flow, only the top-level steps in the flow appear in the **Task Details** dialog box. Steps in subordinate flows and in flow plug-ins and repeated-execution plug-ins are not displayed.

#### Tip

You can view a list of the steps in a flow in the **Flow Tree** view on the right side of the **Editor** window. In the **Flow Tree** view, the name of the service template appears as the first step of the flow. Lower levels are represented by the step name associated with the step that executes the flow plug-in or repeated-execution plug-in.

#### Related topics

- Task log details in the *Job Management Partner 1/Automatic Operation Administration Guide*

## 3.2 Adding steps

---

### 3.2.1 Procedure for adding steps

You can add processing to a flow by adding steps. A step is a plug-in that the user has dropped into a flow.

The following limits apply to the number of steps you can add:

- Maximum number of steps in one service template: 320
- Maximum number of steps at a given hierarchical level: 80

#### If the service template editing view is not displayed:

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Edit**.
2. In the **Service Template List** window, select a service template and then click **Edit**.

#### To add a step:

1. In the **Plug-in** view on the left side of the service template editing view, select the plug-in you want to add as a step.  
You can use any type of plug-in (basic plug-ins, development plug-ins, and release plug-ins) as steps.
2. Drag the plug-in you selected to the **Flow** view.  
**Create Step** dialog box appears.
3. Enter the definition information for the step in the **Create Step** dialog box, and then click **OK**.  
You must enter values in the **ID** and **Name** fields. Other information and property mapping can be set at a later stage.  
The step is added to the **Flow** view.

#### Related topics

- [3.2.3 Settings in step definition information](#)
- [4.1.2 Overview of basic plug-ins, release plug-ins, and development plug-ins](#)

### 3.2.2 Procedure for changing step definition information

The definition information users set in the **Create Step** dialog box when creating a step can be changed in the **Edit Step** dialog box.

#### If the service template editing view is not displayed:

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Edit**.
2. In the **Service Template List** window, select a service template and then click **Edit**.

#### To change step definition information:

1. In the **Steps** view of the service template editing view, click the step whose definition information you want to change, and then click **Edit Step**.

2. In the **Edit Step** dialog box, change the definition information as needed and then click **OK**.

## Tip

You can also change the definition information for a step by right-clicking the step in the **Flow** view and choosing **Edit** from the right-click menu.

### Related topics

- [3.2.3 Settings in step definition information](#)

## 3.2.3 Settings in step definition information

In the **Edit Step** dialog box, you can set information for a step, such as the ID, name, input properties, and output properties.

Table 3–2: Items that can be set in the **Edit Step** dialog box

Item		Description
Step	ID	A value that uniquely identifies the step. As the step ID, specify a value that is unique within the flow (not including the hierarchy flow). By default, this field displays the plug-in ID. If there is more than one step with the same step ID in a given flow, <code>_n</code> is appended to the end of the step ID (where <code>n</code> is a unique integer starting from 2), which is displayed in the format <code>step-ID_n</code> . Note that if <code>step-ID_n</code> is longer than 30 characters, the excess characters are truncated at the end of the step ID, and the shortened step ID is displayed with the <code>_n</code> suffix.
	Name	The value you specify in this field appears with the icon for the step in the <b>Flow</b> view, and as the name of the step in the <b>Steps</b> view. For the top-level flow, the step ID also appears in the <b>Task Details</b> dialog box. The default is the plug-in name. More than one step can have the same name within a given flow. If there is more than one step with the same name, <code>_n</code> is appended to the end of the step name (where <code>n</code> is a unique integer starting from 2), which is displayed in the format <code>step-name_n</code> . Note that if <code>step-name_n</code> is longer than 64 characters, the excess characters are truncated at the end of the step name, and the shortened step name is displayed with the <code>_n</code> suffix.
	Description	For a top-level flow, the description you specify in this field is displayed as the description of the step in the <b>Task Details</b> dialog box. This field is blank by default.
Plug-in		Displays information about the plug-in. As the plug-in version, the version of the plug-in dragged and dropped from the <b>Plug-in</b> view is displayed. You cannot change this information when editing a step.
Subsequent Step Conditions		Set whether to execute the subsequent step according to the return value of the plug-in executed by the current step.
Input properties		When you set a service property, its value is assigned as an input property of the plug-in. You can also enter a value directly.
Output properties		You can set a service property whose value is updated by the plug-in using an output property.

### Related topics

- [3.2.4 Overview of subsequent step conditions](#)
- [3.2.5 Procedure for setting input property mapping](#)
- [3.2.6 Procedure for setting output property mapping](#)
- [List of limit values in the Job Management Partner 1/Automatic Operation GUI and Command Reference](#)

## 3.2.4 Overview of subsequent step conditions

You can set whether to execute a subsequent step based on the return value of the previous step.

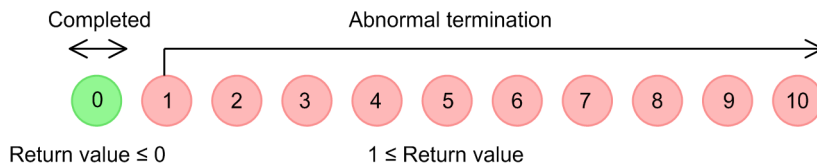
In most situations, the return value of the step is the same as the return value of the plug-in. For details about the relationship between the return values of steps and plug-ins, see [4.3.12 Relationship of command and script return values to the return values of plug-ins and steps](#).

### Types of subsequent step conditions:

#### When the return value of the plug-in is equal to or less than the judgment level, execute subsequent steps

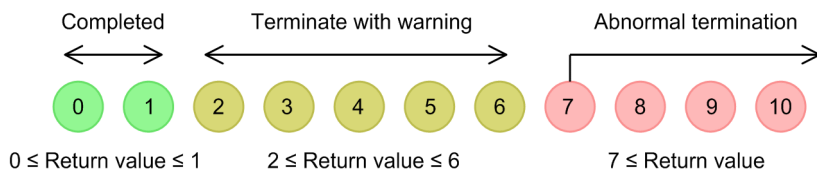
The following shows examples of setting judgment values and warning values:

Figure 3–3: When judgment value is 0 and no warning value is set (default)



- If the return value is equal to or less than the judgment value, the subsequent step is executed (normal termination).
- If the return value is greater than the judgment value, the subsequent step is not executed (abnormal termination).

Figure 3–4: When judgment value is 6 and warning value is 2



- If the return value is equal to or less than the judgment value, the subsequent step is executed (normal termination). However, if the return value is equal to or greater than the warning value and equal to or less than the judgment value, the subsequent step is executed and the task terminates with a warning. In this case, the task status appears as **Failed** (after the task has finished) or **Abnormal Detection** (while the task is still running) to indicate that the warning value was exceeded. In the **Task Details** dialog box, the step is recorded as having terminated with a warning. When you specify a warning value, specify a value that is equal to or less than the judgment value. You do not need to specify a warning value.
- If the return value is larger than the judgment value, the subsequent step is not executed (abnormal termination).

#### Always execute subsequent steps

Subsequent steps are always executed, regardless of the return value of the plug-in.

#### Never execute subsequent steps

The step ends abnormally regardless of the return value of the plug-in. Subsequent steps are not executed.

#### Plug-ins for which subsequent step conditions cannot be set

You cannot specify subsequent step conditions for plug-ins such as flow plug-ins and test value plug-ins that control a flow.

- Flow plug-in
- Judge returncode plug-in
- test value plug-in



- Interval plug-in
- abnormal-end plug-in
- judge value plug-in

### 3.2.5 Procedure for setting input property mapping

You can set a service property and assign its value to an input property of a plug-in. For example, if you map Service property 1 to an input property, the input property stores the value of Service Property 1.

You can also enter a value directly into a plug-in property.

#### If the service template editing view is not displayed:

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Edit**.
2. In the **Service Template List** window, select a service template and then click **Edit**.

#### To assign the value of a service property to an input property:

1. In the **Steps** view of the service template editing view, click the step whose definition information you want to change.
2. Click the **Edit Step** button.
3. In the **Input Properties** area of the **Edit Step** dialog box, select a plug-in property and then click **Edit**.
4. In the **Specify Plug-in Input Properties for Mapping** window, select the **Select a Property from Service Properties** option for the **Input Method** setting.
5. Select a service property in the **Service Properties** list, and then click **OK**.  
The value of the selected service property can now be applied to the plug-in property.

#### To enter a property value directly:

1. In the **Steps** view of the service template editing view, click the step whose definition information you want to change.
2. Click the **Edit Step** button.
3. In the **Input Properties** area of the **Edit Step** dialog box, select a plug-in property and then click **Edit**.
4. In the **Specify Plug-in Input Properties for Mapping** window, select the **Direct Input** option for the **Input Method** setting.
5. Enter a value in the **Mapping Parameter** field, and then click **OK**.

When using direct input, you can specify a value that includes linefeed characters.

You can also specify a value that combines a number of reserved properties and service properties with literal characters. You can reference the value of a property in the format `?dna_property-key?`.

#### Related topics

- [3.2.7 List of reserved properties](#)

## 3.2.6 Procedure for setting output property mapping

You can set an output property that assigns a value to a plug-in property. For example, if you map Service property 1 to an output property of a plug-in, Service Property 1 stores the value of the output property.

### If the service template editing view is not displayed:

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Edit**.
2. In the **Service Template List** window, select a service template and then click **Edit**.

### To apply the value of a plug-in property to a service property:

1. In the **Steps** view of the service template editing view, click the step whose definition information you want to change.
2. Click the **Edit Step** button.
3. In the **Edit Step** dialog box, select a plug-in property in the **Output Properties** area, and then click **Edit**.
4. Select a property in the **Service Properties** area of the **Specify Plug-in Output Properties for Mapping** window. You can also add a property by clicking the **Add Output Property** button or **Add Variable** button.
5. Click **OK**.

The value of the plug-in property can now be applied to the service property.

## 3.2.7 List of reserved properties

A reserved property is a special service property whose property key has a specific definition or purpose in JP1/AO. The property key of a reserved property begins with `reserved.`. You can use reserved properties by mapping them to plug-in properties in the **Specify Plug-in Input Properties for Mapping** dialog box or the **Specify Plug-in Output Properties for Mapping** dialog box. Users cannot define or assign values to reserved properties.

When you map a reserved property to an input property, the value of the reserved property is assigned to a plug-in property when the plug-in is executed.

To use a reserved property, select the **Select a Property from Service Properties** option in the **Specify Plug-in Input Properties for Mapping** dialog box.

Alternatively, select the **Direct Input** option, and in the **Mapping Parameter** field, specify the reserved property in the format `?dna_reserved-property-key?`. In this case, the value of the reserved property supplies part of the value of the plug-in properties at plug-in execution.

When you use a reserved property as an output property, the reserved property stores the value of a designated plug-in property. By selecting the **Select a Property from Service Properties** option in the **Specify Plug-in Output Properties for Mapping** dialog box, you can specify a reserved property to which the value of the output property is passed.

Table 3–3: List of reserved properties

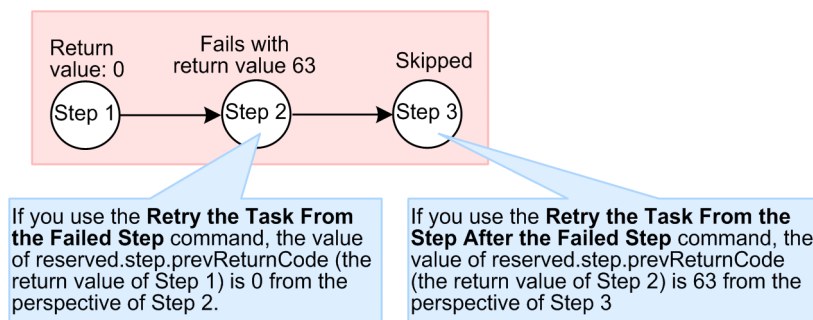
Reserved property key	Description
<code>reserved.loop.index</code>	References a numerical value from 1 to 99 that indicates how many times a repeated execution plug-in has repeated.

Reserved property key	Description
<code>reserved.loop.index</code>	Of the comma-delimited values specified in the <code>inputProperties</code> property of the repeated execution plug-in, this property stores the position of the parameter to which the current execution applies. You can also reference this value when <code>parallel</code> is set as the execution method of the repeated execution plug-in. To reference this reserved property, specify the property key in the format <code>? dna_reserved.loop.index?</code> . Like the <code>reserved.loop.input</code> property, you can use this property in any plug-in within the flow of the repeated execution plug-in, provided that service properties can be mapped to the plug-in.
<code>reserved.loop.input</code>	References the value of the <code>inputProperties</code> input property of a repeated execution plug-in. Of the comma-delimited values specified in the input properties of the repeated execution plug-in, this property stores the value of the element that corresponds to the current iteration of the flow. For example, if the input property is <code>A, B, C</code> , the values <code>A, B, and C</code> are input in the order corresponding to the repetition count of the flow. The repeated execution plug-in can be executed a maximum of 99 times.
<code>reserved.loop.output</code>	Passes values to the <code>outputProperties</code> output property of a repeated execution plug-in. The values output to this property are assigned to the output property as a comma-separated value. For example, if the values of the output property of the plug-in are <code>X, Y, and Z</code> for successive iterations, the value <code>X, Y, Z</code> is assigned to the output property.
<code>reserved.service.category</code>	References the category assigned to the service from which a task was generated. To reference this reserved property, specify the property key in the format <code>? dna_reserved.service.category?</code> . You can use this property in any plug-in to which service properties can be mapped.
<code>reserved.service.name</code>	References the name of the service from which a task was generated. To reference this reserved property, specify the property key in the format <code>? dna_reserved.service.name?</code> . You can use this property in any plug-in to which service properties can be mapped.
<code>reserved.service.resourceGroupName</code>	References the resource group in which the service from which a task was generated is registered. To reference this reserved property, specify the property key in the format <code>? dna_reserved.service.resourceGroupName?</code> . You can use this property in any plug-in to which service properties can be mapped.
<code>reserved.step.path</code>	References the ID of the step that is currently being executed. To reference this reserved property, specify the property key in the format <code>? dna_reserved.step.path?</code> . The value of this property is the same as the step ID displayed in the messages output to the task log when plug-in execution begins and ends. You can use this property in any plug-in to which service properties can be mapped.
<code>reserved.step.prevReturnCode</code>	Supplies the return value of the preceding step (the step that is the origin of the relational line connected to the plug-in). To reference this reserved property, specify the property key in the format <code>? dna_reserved.step.prevReturnCode?</code> . If there are multiple preceding steps, the property is assigned the logical sum of all the return values. If there is no preceding step, 0 is assigned. You can use this property in any plug-in to which service properties can be mapped. If you retry a task from a step that references this reserved property without executing the preceding step, the return value from the last time the preceding step was executed is set in this reserved property as the return value of the preceding step. <sup>#</sup>
<code>reserved.task.description</code>	Supplies the description of a task. To reference this reserved property, specify the property key in the format <code>? dna_reserved.task.description?</code> . You can use this property in any plug-in to which service properties can be mapped.
<code>reserved.task.dir</code>	Supplies the path of the temporary data folder created during task execution. This property provides a unique folder path at execution of each task. The folder referenced by this property is created on the JPI/AO server when the task is executed, and deleted when the task is archived.

Reserved property key	Description
<code>reserved.task.dir</code>	Note that files and folders that start with <code>task</code> are reserved in JPI/AO, and cannot be created by the user.
<code>reserved.task.id</code>	Supplies the task ID. To reference this reserved property, specify the property key in the format ? <code>dna_reserved.task.id?</code> . You can use this property in any plug-in to which service properties can be mapped.
<code>reserved.task.name</code>	Supplies the task name. To reference this reserved property, specify the property key in the format ? <code>dna_reserved.task.name?</code> . You can use this property in any plug-in to which service properties can be mapped.
<code>reserved.task.submitter</code>	Supplies the user ID of the user who submitted the task for execution. If the task was retried, this property references the user ID of the user who submitted the task, not the user who retried the task. To reference this reserved property, specify the property key in the format ? <code>dna_reserved.task.submitter?</code> . You can use this property in any plug-in to which service properties can be mapped.
<code>reserved.task.url</code>	Supplies the URL for accessing the <b>Task Details</b> dialog box. To reference this reserved property, specify the property key in the format ? <code>dna_reserved.task.url?</code> . You can use this property in any plug-in to which service properties can be mapped.
<code>reserved.terminal.account</code>	References the user ID used by a terminal connect plug-in. This property is used by the <code>commandLine</code> input property of a terminal command plug-in. It stores the login name of the user account used to connect to the terminal.
<code>reserved.terminal.password</code>	References the password used by a terminal connect plug-in. This property is used by the <code>commandLine</code> input property of a terminal command plug-in. It stores the password of the user account used to connect to the terminal.
<code>reserved.terminal.suPassword</code>	References the administrator password used by the terminal connect plug-in. This property is used by the <code>commandLine</code> input property of a terminal command plug-in. It stores the password of the superuser used to connect to the terminal.

#

The following shows examples of the values assigned to the reserved property when a task is retried:



The illustrated flow consists of Step 1, Step 2, and Step 3. The `reserved.step.prevReturnCode` property is defined for Step 2 and Step 3. The return value of Step 1 is 0, and the return value of Step 2 is 63, indicating an error. The task fails with Step 3 in *Not Executed* status.

In this scenario, if you use the **Retry the Task From the Failed Step** option, the value of `reserved.step.prevReturnCode` (the return value of Step 1) is 0 from the perspective of Step 2. If you use the **Retry the Task From the Step After the Failed Step** option, the value of `reserved.step.prevReturnCode` (the return value of Step 2) is 63 from the perspective of Step 3.

### 3.2.8 Warning icon displayed for steps

If a mandatory mapping parameter is omitted, a warning icon is displayed for the step.

Figure 3–5: Example of steps with warning icon displayed



The icon is updated when you close the **Create Step** dialog box or the **Edit Step** dialog box and when you edit a mapping parameter.

When you copy and paste a step with a warning icon, the pasted step retains the warning icon.

## 3.3 Connecting steps with relational lines

---

### 3.3.1 Procedure for connecting steps with relational lines

After adding steps to a flow, you can arrange the steps in the appropriate order for the task being automated by connecting them using relational lines.

#### If the service template editing view is not displayed:

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Edit**.
2. In the **Service Template List** window, select a service template and then click **Edit**.

#### To connect steps with relational lines:

1. In the **Flow** view of the service template editing view, click the circle beside the icon for the step you want to execute first, and drag it to the icon of the step you want to execute next. Release the mouse button when the icon changes color.

The steps are connected by relational lines.

#### Related topics

- [3.3.8 Behavior when relational lines connect to multiple steps](#)
- [3.3.9 Scenarios where relational lines cannot be drawn](#)
- [3.3.10 Drawing relational lines when processing branches](#)

### 3.3.2 Procedure for deleting steps and relational lines

You can delete a step or a relational line. When you delete a step, the relational steps that connect that step are also deleted. When you delete a relational line, the steps connected by that line are disassociated.

#### If the service template editing view is not displayed:

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Edit**.
2. In the **Service Template List** window, select a service template and then click **Edit**.

#### To delete a step or relational line:

1. In the **Flow** view of the service template editing view, right-click the step or relational line you want to delete, and then click **Delete**.

You can also delete a step or relational line by pressing the **Delete** key.

#### Related topics

- [3.3.5 Procedure for selecting multiple steps](#)

### 3.3.3 Procedure for copying steps and relational lines

You can copy steps and relational lines and paste them into the **Flow** view for the same service template or a different service template.

The system copies any steps and relational lines in the selected range. If the selected items include a hierarchical flow, the copy operation includes all subordinate flows. The clipboard only holds data from a single copy operation.

#### If the service template editing view is not displayed:

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Edit**.
2. In the **Service Template List** window, select a service template and then click **Edit**.

#### To copy a step or relational line:

1. In the **Flow** view of the service template editing view, right-click the step or relational line you want to copy, and choose **Copy** from the right-click menu.

#### Related topics

- [3.3.5 Procedure for selecting multiple steps](#)
- [3.3.7 Information inherited when pasting steps or relational lines](#)

### 3.3.4 Procedure for cutting steps and relational lines

You can cut steps and relational lines and paste them into the **Flow** view for the same service template or a different service template.

Any steps and relational lines within the selection are included in the cut operation. If the selected items include a hierarchical flow, the cut operation includes all subordinate flows. The clipboard only holds data from a single copy operation.

Any relational lines connected to steps outside the selected range disappear from the **Flow** view as soon as you cut the selection.

#### If the service template editing view is not displayed:

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Edit**.
2. In the **Service Template List** window, select a service template and then click **Edit**.

#### To cut steps or relational lines:

1. In the **Flow** view of the service template editing view, right-click the step or relational line you want to cut, and choose **Cut** from the right-click menu.

#### Related topics

- [3.3.5 Procedure for selecting multiple steps](#)
- [3.3.7 Information inherited when pasting steps or relational lines](#)

### 3.3.5 Procedure for selecting multiple steps

You can select multiple steps at once.

#### If the service template editing view is not displayed:

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Edit**.
2. In the **Service Template List** window, select a service template and then click **Edit**.

#### To select multiple steps:

1. In the **Flow** view of the service template editing view, select steps by clicking and dragging with your mouse. You can also select multiple steps by clicking steps while pressing the **Ctrl** key.

#### Tip

You can select all steps or relational lines by right-clicking an area in the **Flow** view where there are no steps or relational lines, and choosing **Select All** from the right-click menu.

### 3.3.6 Procedure for pasting steps and relational lines

After copying or cutting steps or relational lines, you can paste the data from the clipboard into the desired location in the **Flow** view for a development service template.

The information inherited when you paste the data depends on whether a step with the same ID or name is present at the destination of the paste operation.

#### If the service template editing view is not displayed:

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Edit**.
2. In the **Service Template List** window, select a service template and then click **Edit**.

#### To paste steps or relational lines:

1. Right-click an empty part of the **Flow** view of the service template editing view, and choose **Paste** from the right-click menu.

#### Related topics

- [3.3.7 Information inherited when pasting steps or relational lines](#)

### 3.3.7 Information inherited when pasting steps or relational lines

When you paste a step or relational line you have copied or cut, some information associated with the item is inherited at the destination. After pasting a step or relational line, review the definition information for the destination development service template as needed.

The following table shows the information related to steps or relational lines that is inherited when you paste the item.



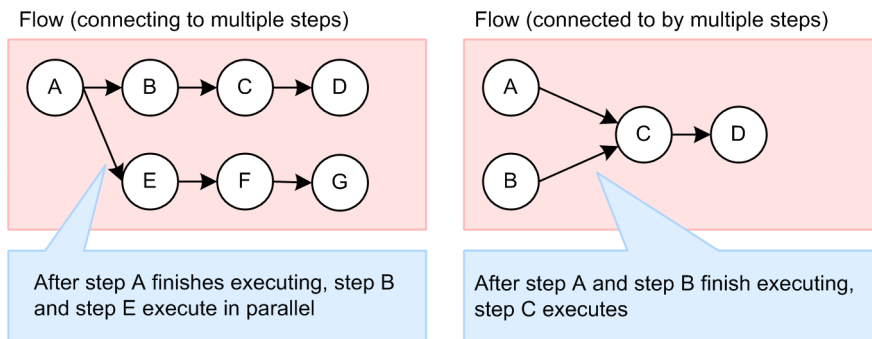
Table 3–4: Information inherited when pasting steps or relational lines

Copied or cut item	Information for copied or cut item	Information at paste destination
Step	Step ID	A step ID must be unique within the flow (not including subordinate hierarchy flows). If there is no step with the same step ID at the paste destination, the information of the copy or cut source is inherited as-is. If there is a step with the same step ID, the step ID of the pasted step changes to <i>step-ID_n</i> , where <i>n</i> is a unique integer of 2 or higher. If <i>step-ID_n</i> is longer than 30 characters, the excess characters are truncated from the end of the step ID.
	Step name	If there is no step with the same name at the paste destination, the information of the copy or cut source is inherited as-is. If there is a step with the same name, the step name of the pasted step is changed to <i>step-name_n</i> , where <i>n</i> is a unique integer of 2 or higher. If <i>step-name_n</i> is longer than 64 characters, the excess characters are truncated from the end of the step ID.
	Description	The information for the copy or cut source is pasted to the destination as-is.
	References to plug-in information	
	Subsequent-step execution conditions	
	Input property mapping parameters	
	Output property mapping parameters	
Relational lines	Direction	The system copies information about the direction of relational lines. However, lines that connect to steps outside the selected range are not copied.

### 3.3.8 Behavior when relational lines connect to multiple steps

A flow can contain relational lines that connect one step to several, and several steps to one. When lines are drawn in these ways, the next step is executed only after every connected step has finished executing.

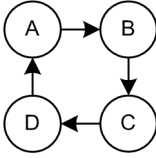
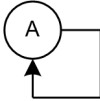
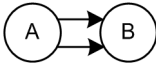
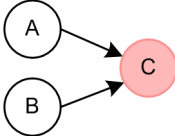
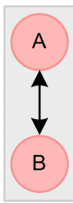
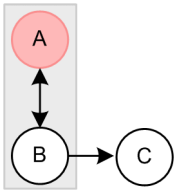
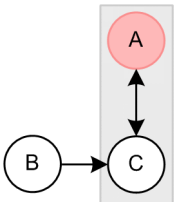
Figure 3–6: Example of connecting multiple steps





### 3.3.9 Scenarios where relational lines cannot be drawn

You cannot connect relational lines in certain configurations. For example, you cannot draw a relational line that would result in a recurring loop within a flow, or whose source and destination are the same step.

Table 3–5: Restrictions on relational lines

Restricted relational line connection	Details	Example
Relational lines that form a loop	You cannot use relational lines in a way that creates a loop within a flow.	
Relational lines that connect a step to itself	You cannot draw a relational line if the source and destination steps of the line are the same.	
Identical relational lines	You cannot create parallel relational lines with the same source and destination step.	
Multiple inputs to a judge returncode plug-in	Because a judge returncode plug-in judges the return value of the preceding step, you cannot draw relational lines to such a plug-in from multiple steps. Only one step can serve as the preceding step of a judge returncode plug-in.	
Connecting a judge returncode plug-in or judge value plug-in to a judge returncode plug-in or judge value plug-in	You cannot draw a relational line that connects a judge returncode plug-in or judge value plug-in to a judge returncode plug-in or judge value plug-in. However, you can connect such a plug-in as a succeeding step.	
Relational lines from a branch destination step of a judge returncode plug-in or judge value plug-in	A judge returncode plug-in or judge value plug-in can only have one branch destination step, which cannot be connected further by a relational line. If you need to use several steps, draw a connection line to a flow plug-in.	
Relational lines to a branch destination step of a judge returncode plug-in or judge value plug-in	You cannot draw a relational line to the branch destination step of a judge returncode plug-in or judge value plug-in.	

Legend:  : Judge returncode plug-in or judge value plug-in  : Other plug-in

#### Related topics

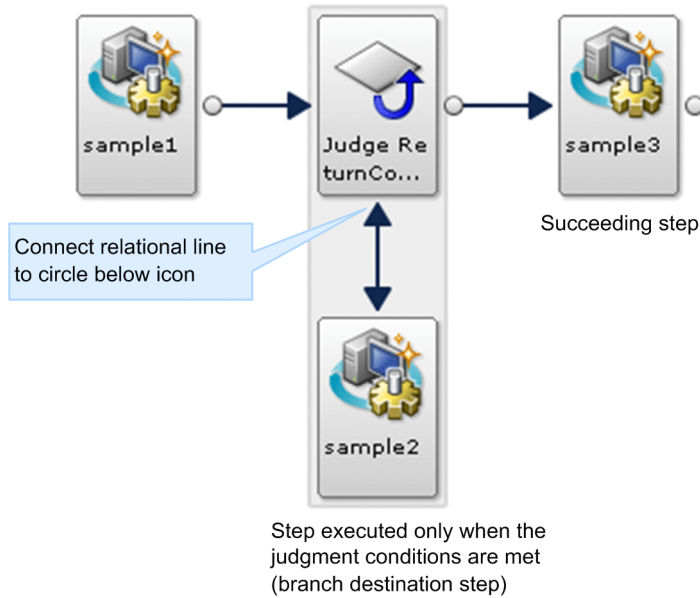
- [3.3.10 Drawing relational lines when processing branches](#)
- The description of judge returncode plug-ins in the manual *Job Management Partner 1/Automatic Operation Service Template Reference*

### 3.3.10 Drawing relational lines when processing branches

If you want to execute a particular step only when judgment conditions are met, you can use a judge returncode plug-in or judge value plug-in. When using such a plug-in, draw the relational line from the circle below the icon of the plug-in.

When the conditions are met, the branch destination step is executed first, followed by the succeeding step. If the conditions are not met, the succeeding step is executed without first executing the branch destination step.

Figure 3–7: Example of relational lines connecting a judge returncode plug-in



- The connection with the succeeding step is represented by a single-headed arrow.
- The relational line that connects a step executed only when the judgment conditions are met is represented by a double-headed arrow. There cannot be more than one step executed when judgment conditions are met.
- If the judgment conditions are met, the steps are executed in the following order: `sample1`, judge returncode plug-in, `sample2`, and then `sample3`.
- If the conditions are not met, the steps are executed in the following order: `sample1`, judge returncode plug-in, and then `sample3`.

# 4

## Creating and editing plug-ins

This chapter describes how to create and edit plug-ins. You can use the plug-ins provided by JP1/AO in an unmodified state, or create and edit plug-ins to define processing that meets a specific need.

## 4.1 Displaying a list of plug-ins

### 4.1.1 Procedure for displaying a list of plug-ins

You can view a list of the plug-ins that have been imported into JP1/AO.

The list of plug-ins appears in the **Plug-in** view on the left side of the **Editor** window.

#### Tip

You can display the name, version, and detailed information of a plug-in by clicking it in the list.

#### Related topics

- [4.1.2 Overview of basic plug-ins, release plug-ins, and development plug-ins](#)

### 4.1.2 Overview of basic plug-ins, release plug-ins, and development plug-ins

A plug-in defines processing that executes a task.

There are three types of plug-in in JP1/AO: basic plug-ins, release plug-ins, and development plug-ins. Each type is displayed in its own tab in the **Plug-in** view.

For the sake of expedience, plug-ins are separated into *basic plug-ins* and *content plug-ins* according to their origin. For details, see Types of Service Templates and Plug-ins in the manual *Job Management Partner 1/Automatic Operation Service Template Reference*.

Table 4–1: Types of plug-in

Type	Description
Basic plug-in	<ul style="list-style-type: none"><li>• Displayed in the <b>Basic</b> tab.</li><li>• A plug-in provided by JP1/AO. A basic plug-in defines generic processing like email notification and flow repetition.</li></ul>
Content plug-ins	Release plug-ins
	Development plug-ins
	<ul style="list-style-type: none"><li>• Displayed in the <b>Release</b> tab.</li><li>• A plug-in that was imported into JP1/AO by a user releasing a service he or she created in the <b>Editor</b> window.</li><li>• A plug-in in a service templates provided by JP1/AO.</li><li>• A plug-in that was imported into the JP1/AO server by the <code>importservicetemplate</code> command and has the <i>released</i> configuration type is also handled as a release plug-in.</li></ul> <ul style="list-style-type: none"><li>• Displayed in the <b>Under Development</b> tab.</li><li>• A plug-in that a user created as a new plug-in, which has not yet been released. A plug-in that is being created based on a copy of an existing plug-in is also classified as a development plug-in.</li><li>• When you build a development service template that includes a development plug-in, the development plug-in is imported into the JP1/AO server and can be executed for testing purposes.</li></ul>

Type		Description
Content plug-ins	Development plug-ins	<ul style="list-style-type: none"> <li>A plug-in that was imported into the JP1/AO server by the <code>importservicetemplate</code> command and has the <code>debug</code> configuration type is also handled as a development plug-in.</li> </ul>

By using plug-ins, you can perform actions like the following:

- Sending notification emails and controlling flow repetition
- Transfer files and folders between the JP1/AO server and a remote host
- Connect to a remote host and execute commands and scripts

In JP1/AO, a user can create a custom plug-in as a content plug-in. Users can also create plug-ins that connect to a remote host and execute commands and scripts, and incorporate these plug-ins into a service template.

When JP1/AO executes a content plug-in, it uses WMI to connect to operation target devices that are running Windows, and SSH to connect to UNIX devices. For details about basic plug-ins, see the description of basic plug-ins in the manual *Job Management Partner 1/Automatic Operation Service Template Reference*.

## Related topics

- [4.1.3 Plug-in executing users](#)
- [4.1.4 Available operations by plug-in type](#)
- [4.1.5 Files transferred to Windows systems](#)
- [4.1.6 Files transferred to UNIX systems](#)
- [4.1.7 Locale set for operation target devices during plug-in execution](#)
- [4.1.8 Character set used for communication by JP1/AO during plug-in execution](#)
- [4.1.9 Setting a specific character set during plug-in execution](#)
- [A.1\(2\) Configuration types assigned to service templates and plug-ins by build and release operations](#)

### 4.1.3 Plug-in executing users

The execution user of a plug-in is as follows:

- When the OS of the operation target device is Windows

Commands and scripts are executed by the user who connects to the operation target device.

When the operation target device is running Windows, user profiles are not inherited. This means a plug-in can produce different execution results from a command or script executed on the desktop.

To avoid this issue, do not reference settings in user profiles, such as user environment variables, registry entries, and Internet Explorer settings, when executing a plug-in. If a command or script references an element of a user profile, the command or script might not behave as expected. For example, when you execute a command or script that references Internet Explorer proxy settings, the command or script might fail with a communication error. This might occur in scenarios such as implementing a Windows Update using a script.

- When the OS of the operation target device is UNIX

Generally, the user who connects to the operation target device is the execution user of commands and scripts. JP1/AO also provides a function that allows you to elevate the execution user of a command or script to root privilege.

Note that when a user connects to an operation target device as a user with root privilege, the connection of the root privilege user must be permitted on the operation target device side.

The following table lists the execution users for plug-ins.

Table 4–2: Execution users for plug-ins

Plug-in	Elevation to root privilege <sup>#1</sup>	User who connected to operation target device	Execution user of command or script <sup>#2</sup>
<ul style="list-style-type: none"> <li>Basic plug-in (general command plug-in, file-forwarding plug-in, or terminal connect plug-in)</li> <li>Content plug-in</li> </ul>	Enabled	User with root privilege	User with root privilege
		User without root privilege	User with root privilege
	Not enabled	User with root privilege	User with root privilege
		User without root privilege	User without root privilege

#1

The process by which the user is elevated to root privilege depends on the plug-in.

- For basic plug-ins:
  - General command plug-in and file-forwarding plug-in  
You can specify whether to elevate the user to root privilege in the plug-in properties.
  - For the terminal connect plug-in:  
You cannot configure JP1/AO to elevate users of the terminal connect plug-in to root privilege. To achieve this, you need to execute the command that elevates the user to root privilege in a terminal command plug-in.

For details about the elevation of users to root privilege, see the section describing basic plug-ins in the manual *Job Management Partner 1/Automatic Operation Service Template Reference*.

- For content plug-ins:  
You can specify the permissions of the execution user by using the **Execute by using root privileges (for SSH connections)** check box in the **Create Plug-in** dialog box or the **Edit Plug-in** dialog box. If you select this check box, commands and scripts are executed as a user with root privilege. If the check box is cleared, commands and scripts are executed with the permissions of the user who connected to the operation target device.

### Tip

When the **Execute by using root privileges (for SSH connections)** is selected, the way in which you specify the superuser password depends on the credential type selected in the **Create Plug-in** or **Edit Plug-in** dialog box.

- If **Destination** is selected as the credential type for the plug-in  
JP1/AO uses the superuser password specified in the definition of the connection destination.
- If **Property** is selected as the credential type for the plug-in  
JP1/AO uses the superuser password specified in the plugin.suPassword plug-in property.

#2

- In the case of a file-forwarding plug-in, the user who transfers the file.
- In the case of a terminal connect plug-in, the command is actually executed by a terminal command plug-in.

## 4.1.4 Available operations by plug-in type

The operations you can perform in the **Editor** window depend on the type of plug-in selected.

Table 4–3: Available operations by plug-in type

Type	Plug-in operation				
	Show in list	Create	Edit	Delete	Copy
Basic plug-in	Y	N	N	N	N
Development plug-in	Y	Y	Y	Y #1	Y
Release plug-in	Y	N	N	Y #1#2	Y

Legend:

Y: Can be performed. N: Cannot be performed.

#1

Cannot be deleted when being used in a development service template.

Cannot be deleted if a development service template has been built that incorporates the plug-in you want to delete. In this case, delete the step that uses the plug-in and build the template again. You can then delete the plug-in.

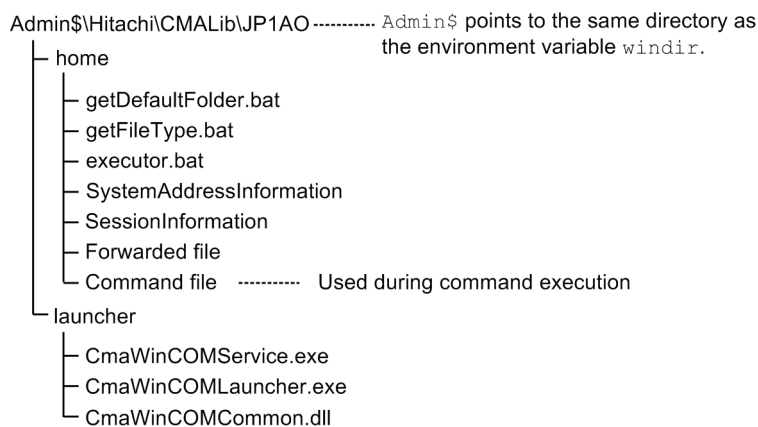
#2

Cannot be deleted when being used in a release service template.

## 4.1.5 Files transferred to Windows systems

When a general command plug-in, file-forwarding plug-in, or content plug-in executes an operation on a Windows device, JP1/AO transfers the following files to the device. The files are deleted when the plug-in finishes executing.

Figure 4–1: Files transferred to Windows systems



Files are transferred when either of the following conditions are met:

- You use a file-forwarding plug-in
- A script is executed in a content plug-in

### Related topics

- [4.3.9 Procedure for setting scripts](#)



## 4.1.6 Files transferred to UNIX systems

When a file-forwarding plug-in or content plug-in executes an operation on a UNIX device, JP1/AO transfers the following files to the device. The files are deleted when the plug-in finishes executing.

Figure 4–2: Files transferred to UNIX systems

*working-folder*  
└ Forwarded file

Files are transferred in the following circumstances:

- You use a file-forwarding plug-in
- A script is executed in a content plug-in

You can set *working-folder* in the `plugin.remoteCommand.workDirectory.ssh` property in the property file (`config_user.properties`). The default is `/tmp/Hitachi_AO`.

### Related topics

- [4.3.9 Procedure for setting scripts](#)
- Property file (`config_user.properties`) in the *Job Management Partner 1/Automatic Operation Configuration Guide*

## 4.1.7 Locale set for operation target devices during plug-in execution

The locale setting that applies to a device on which an operation is performed by a plug-in depends on the operating system. The following describes the locale settings applied when plug-ins are executed in each operating system.

### In Windows

When JP1/AO executes a script or command on an operation target device, make sure that the locale and character set of the operation target device match those of the JP1/AO server. The locale and character set are determined by the settings in the **Region and Language** area of the Windows **Control Panel** that govern date and time formats, user-level display languages, system-level display languages, and system locale settings.

For details about the character set JP1/AO uses for communication, see [4.1.8 Character set used for communication by JP1/AO during plug-in execution](#).

### In UNIX

The locale setting applied during plug-in execution depends on the **Character Set Auto Judgment (SSH)** setting in the **Create Plug-in** dialog box or the **Edit Plug-in** dialog box.

- If the **Enabled** check box is cleared in the **Character Set Auto Judgment (SSH)** area:  
Scripts are executed with the `LC_ALL=C` locale. Make sure that commands and command parameters consist only of ASCII characters. If a command parameter, standard output, or standard error output contains non-ASCII characters, the characters might become garbled and prevent the command from executing normally.
- If the **Enabled** check box is selected in the **Character Set Auto Judgment (SSH)** area:  
JP1/AO references the default locale of the connecting user and executes the script accordingly.  
When executing a script or command on an operation target device, JP1/AO sets the environment variable `LC_ALL` and `LANG` to the default locale of the connecting user. It does not change the settings of `LC_XXXXX` environment variables other than `LC_ALL`.

The locale assigned when executing a script or command is referenced in the following order of priority:

Table 4–4: Priority of locale settings referenced during plug-in execution

Priority	Environment variable
1	Value of LC_ALL
2	Value of LC_CTYPE
3	Value of LANG

If the script or command is encoded in a different character set from the one assigned at plug-in execution, the characters might become garbled. Note that the character set you can use in commands and command parameters depends on the operating system. For details, see [4.1.8 Character set used for communication by JP1/AO during plug-in execution](#).

## 4.1.8 Character set used for communication by JP1/AO during plug-in execution

The character set that is assigned at plug-in execution and used by JP1/AO for communication depends on the operating system of the device on which the operation is being performed.

Entries output to the task log and public log by the JP1/AO server are output in the default character set of the operating system of the JP1/AO server. For this reason, characters taken from a character set that is incompatible with the operation target device, machine-dependent characters, and Unicode-dependent characters might become garbled when output to a log file.

The following describes the character sets assigned at plug-in execution according to the operating system of the operation target device.

In Windows:

When executing a script or command on an operation target device, make sure that the locale and character set of the operation target device match those of the JP1/AO server. The locale and character set are determined by the settings in the **Region and Language** area of the Windows **Control Panel**.

In UNIX:

When executing a script or command on an operation target device, the character sets the JP1/AO server can use for communication are limited to the following character sets output by the `locale charmap` command. Note that the output of the `locale charmap` command is not case sensitive.

- EUC-JP
- eucjp
- ibm-943C
- ISO-8859-1
- MS932
- PCK
- Shift\_JIS
- UTF-8
- windows-31j

If the command returns a character set that is not one of those listed here, UTF-8 is assigned as the character set. Note that if the output of the `locale charmap` command is IBM-943, JP1/AO uses the `ibm-943C` character set for communication when executing the plug-in.

To find out which character set JP1/AO is using, use an SSH client or the `ssh` command to log in as the connection user, and then execute the `locale charmap` command. If you want to automatically change the character set when the connection user logs in, use a login script or other means to assign values to environment variables at login. You can change the character set at login by assigning a value to the `LC_ALL` or `LANG` environment variable. If you want to assign a specific character set, see [4.1.9 Setting a specific character set during plug-in execution](#).

## 4.1.9 Setting a specific character set during plug-in execution

If you want JP1/AO to use a specific character set for communication when performing an operation on a UNIX device, enter the appropriate setting in a character-set mapping file, or in the `terminal.charset` key of a connection-destination property file.

If you specify a character set in a character-set mapping file and in the `terminal.charset` key of a connection-destination property file, the character set is assigned in the following order of priority:

Table 4–5: Priority of character set settings during plug-in execution

Priority	Setting
1	Character set specified in the <code>terminal.charset</code> key of the connection-destination property file
2	Character set specified in the character-set mapping file
3	Character set returned by the <code>locale charmap</code> command on the operation target device
4	UTF-8

### Related topics

- Connection-destination property file (`connection-destination.properties`) and Character-set mapping file (`charsetMapping_user.properties` in the *Job Management Partner 1/Automatic Operation Configuration Guide*)

## 4.1.10 Commands required for plug-in execution

Certain commands must be installed in the operating system of the operation target device before you can execute plug-ins. For details, see the Release Notes.

## 4.2 Creating plug-ins

### 4.2.1 Procedure for creating plug-ins

Users can create new plug-ins to meet a specific need.

#### To create a plug-in:

1. In the **Plug-in** view of the **Editor** window, display the **Under Development** tab.
2. Click the **Create** button.
3. In the **Create Plug-in** dialog box, enter the definition information for the plug-in and then click **Save**.

#### Related topics

- [4.2.2 Parameters to set when creating or copying plug-ins](#)
- [4.3.2 Parameters to set in plug-in definition information](#)

### 4.2.2 Parameters to set when creating or copying plug-ins

The following table lists the parameters you can set in the **Create Plug-in** dialog box and the **Copy Plug-in** dialog box.

Table 4–6: Parameters to set in the **Create Plug-in** dialog box and the **Copy Plug-in** dialog box

Parameter	Description
<b>ID</b> <sup>#1</sup>	Specify an ID that identifies the plug-in.
<b>Version</b> <sup>#1</sup>	Specify the version number of the plug-in. Specify the version number in the format <i>aa.bb.cc</i> .
<b>Vendor ID</b> <sup>#1</sup>	Specify the ID of the vendor who created the plug-in. Create a unique vendor ID by specifying the domain name in reverse order from the top level as a period-separated value. For example, specify vendor IDs in the format <i>com.xxx</i> or <i>jp.co.yyy</i> . If you choose not to use domain names as vendor IDs, make sure that the vendor ID you specify is not being used for another vendor. You cannot specify a vendor ID that starts with <i>com.hitachi.software.dna</i> .
<b>Name</b>	Specify a name for the plug-in.
<b>Vendor Name</b> <sup>#2</sup>	Specify the name of the vendor who created the plug-in.
<b>Description</b>	Specify a description of the plug-in.
<b>Category</b>	Specify the name of the plug-in category.

#1

You cannot change the values you specify in the **ID**, **Version**, and **Vendor ID** fields after you create or copy the plug-in.

#2

If you omit this parameter, the value specified in **Vendor ID** is set as the **Vendor Name**. For a development plug-in, the **Vendor Name** field remains blank in the **Editor** window.

In the **Create Plug-in** dialog box, you can also define input properties, output properties, and remote commands. For details about input properties, output properties, and remote commands, see [4.3.2 Parameters to set in plug-in definition information](#).

## 4.3 Editing plug-ins

### 4.3.1 Procedure for editing plug-in definition information

You can edit the definition information for plug-ins displayed in the **Under Development** tab.

**To edit the definition information for a plug-in:**

1. In the **Plug-in** view of the **Editor** window, display the **Under Development** tab.
2. Select the plug-in you want to edit, and then click **Edit**.
3. In the **Edit Plug-in** dialog box, set the definition information for the plug-in, and then click **Save**.

#### Important note

If you are editing the properties of a plug-in that has been placed as a step in a flow, make sure that the mapping parameters in the step are defined appropriately. If the mapping parameters defined in the step do not match the properties of the plug-in, an error occurs when you build or release the service template.

#### Related topics

- [4.3.2 Parameters to set in plug-in definition information](#)

### 4.3.2 Parameters to set in plug-in definition information

In the **Create Plug-in** dialog box or the **Edit Plug-in** dialog box, set the plug-in information, input properties, output properties, and remote commands.

Table 4–7: Parameters set in the **Create Plug-in** or the **Edit Plug-in** dialog box

Parameter	Description
Plug-in information	You can set the plug-in name, vendor name, description, category, icon, credential type, character set auto-judgment setting (for SSH connections), and whether to execute the plug-in as root.
Input properties <sup>#</sup>	Properties that store the input values the plug-in needs while executing.
Output properties <sup>#</sup>	Properties that store the execution results of the plug-in.
Remote commands	Commands or scripts to execute on the remote host. You can set a different set of remote commands for each operating system.

<sup>#</sup>

In total, you can define a maximum of 100 input properties and output properties per plug-in.

#### Related topics for plug-in information

- [4.2.2 Parameters to set when creating or copying plug-ins](#)
- [4.3.3 Image files usable as plug-in icons](#)
- [4.3.4 plug-in credential types](#)

## Related topics for Input properties and Output properties

- [4.3.5 Properties defined in plug-ins \(plug-in properties\)](#)

## Related topics for Input properties

- [4.3.6 Reserved plug-in properties for specifying execution-target hosts and credential information](#)

## Related topics for Remote commands

- [4.3.9 Procedure for setting scripts](#)
- [4.3.15 Differences between script settings methods](#)
- [4.3.17 Procedure for setting commands](#)
- [4.3.7 Procedure for mapping standard output and standard error output to output properties](#)

## 4.3.3 Image files usable as plug-in icons

You can display a custom image as the icon of a plug-in in the Plug-ins list and **Flow** view. Each plug-in can have a different icon. If you do not assign an icon, the standard icon is displayed for the plug-in.

Figure 4–3: Standard plug-in icon



An image file used as the icon for a plug-in must meet the conditions below. If you specify a file that does not meet these conditions, an error occurs when you attempt to register the file.

File format:

PNG

Image size:

48 x 48 pixels

When you register an image file as an icon, the file is renamed `icon.png`.

You can register another image file for a plug-in that already has an icon by selecting the new file. The existing icon is replaced with the image in the new file.

## 4.3.4 plug-in credential types

The following properties are set automatically according to the option selected for **Credential Type** in the **Create Plug-in** dialog box or the **Edit Plug-in** dialog box:

When **Destination** is selected for **Credential Type**

- `plugin.destinationHost`

When **Property** is selected or **Credential Type**

- `plugin.destinationHost`
- `plugin.account`

- `plugin.password`
- `plugin.suPassword`
- `plugin.publicKeyAuthentication`

## Related topics

- [4.3.6 Reserved plug-in properties for specifying execution-target hosts and credential information](#)

### 4.3.5 Properties defined in plug-ins (plug-in properties)

By defining properties for a plug-in, you can specify the parameters a plug-in requires during execution, and collect the execution results of the plug-in. You can define the following plug-in property types:

#### Input properties

You can define properties that store the input values plug-ins require during execution, such as the arguments for remote commands or the target host of the operation. You can define input properties in the following dialog boxes:

- **Create Input Property for Plug-in** dialog box
- **Edit Input Property for Plug-in** dialog box

#### Output properties

You can define properties that store the execution results of the plug-in, such as the results of a remote command (standard output and standard error output). You can define output properties in the following dialog boxes:

- **Create Output Property for Plug-in** dialog box
- **Edit Output Property for Plug-in** dialog box

Plug-in properties are only valid in the context of the plug-in for which they are defined.

Input and output properties can store a maximum of 1,024 characters. If you specify a value that is longer than 1,024 characters, the first 1,024 characters are stored as the property value and the remainder are discarded. If you reference the value of a property key in the format `?dna_property-key?`, the referenced value is truncated if it is longer than 1,024 characters.

A plug-in property whose property key and purpose are determined in advance is called a *reserved plug-in property*. These properties specify credential information and the execution-target hosts of remote commands.

## Related topics

- [4.3.6 Reserved plug-in properties for specifying execution-target hosts and credential information](#)

### 4.3.6 Reserved plug-in properties for specifying execution-target hosts and credential information

A reserved plug-in property is a plug-in property whose property key and intended use are predefined in the JP1/AO system.

The names of reserved plug-in properties start with `plugin..` These properties are created automatically in the **Input Properties** area of the following dialog boxes:

- **Create Plug-in** dialog box



- **Edit Plug-in** dialog box

Plug-in properties are only valid in the context of the plug-in for which they are defined.

Although you can edit some aspects of a reserved plug-in property, other aspects such as the property key and whether certain parameters are mandatory cannot be changed. You cannot delete a reserved plug-in property.

### Reserved plug-in property for specifying execution-target hosts

The following reserved plug-in property is automatically created to specify the execution-target host.

Table 4–8: Reserved plug-in property for specifying execution-target hosts

Property key	Description
<code>plugin.destinationHost</code>	Specify the target of an operation by IPv4 address, IPv6 address, or host name. You must specify a target host in a network configuration in which the JP1/AO server and the command execution environment can communicate directly with each other. You can specify a value from 1 to 256 characters long.

### Reserved plug-in property for specifying credential information

As the credential type of the plug-in, select the **Destination** or the **Property** option in the **Credential Type** area of the **Create Plug-in** or the **Edit Plug-in** dialog box.

#### Destination

Select this option to use the credential information set in the **Connection Destinations** view. When you select this option, JP1/AO uses the credential information specified in the connection destination definition for WMI, SSH, or Telnet connections, depending on the IP address of the logged-in user.

#### Property

Select this option to use the credential information specified in a property.

When you select **Property**, the following reserved plug-in property is automatically created in the **Edit Input Property for Plug-in** dialog box.

Table 4–9: Reserved plug-in properties for specifying credential information

Property key	Description
<code>plugin.account</code>	When the <b>Property</b> option is selected for <b>Credential Type</b> , specify the user ID for logging in to the target host in 1 to 256 characters.
<code>plugin.password</code>	When the <b>Property</b> option is selected for <b>Credential Type</b> , specify the password for logging in to the target host in 1 to 256 characters. If true is specified for the <code>plugin.publicKeyAuthentication</code> reserved plug-in property, JP1/AO ignores the value set for the <code>plugin.password</code> property.
<code>plugin.suPassword</code>	If <b>Property</b> is selected for <b>Credential Type</b> , specify the password of the root account used to log in to a target host in a UNIX environment, using 1 to 256 characters. The root password you specify is ignored in the following circumstances: <ul style="list-style-type: none"> <li>• The target host is running Windows</li> <li>• The <b>Execute by using root privileges (for SSH connections)</b> check box was cleared in the <b>Create Plug-in</b> dialog box or the <b>Edit Plug-in</b> dialog box.</li> </ul>
<code>plugin.publicKeyAuthentication</code>	When <b>Property</b> is selected for <b>Credential Type</b> , this property specifies whether to use public key authentication for SSH connections to target hosts in UNIX environments. If you do not specify a value, false applies. <ul style="list-style-type: none"> <li>• true</li> </ul> Specify this value to use public key authentication.

Property key	Description
plugin.publicKeyAuthentication	<ul style="list-style-type: none"> <li>• false</li> </ul> Specify this value to use password authentication.

### 4.3.7 Procedure for mapping standard output and standard error output to output properties

You can map the standard output and standard error output of a command or script to an output property.

By default, the output filters are blank for properties in the **Mapping Definition of Output Properties** list. This means that the entire standard output and standard error output of commands and scripts is stored in output properties. To extract values from standard output and standard error output and map them to output properties, define regular expressions in the **Edit Output Filter** dialog box.

#### If the **Create Plug-in dialog box** or the **Edit Plug-in dialog box** is not displayed:

1. In the **Plug-in** view of the **Editor** window, display the **Under Development** tab.
2. To create a new plug-in, click **Create**.  
To edit an existing plug-in, select the plug-in and then click **Edit**.

#### To map standard output or standard error output to an output property:

1. In the **Remote Command** area of the **Create Plug-in** dialog box or the **Edit Plug-in** dialog box, click **Edit**.
2. Select a property in the **Mapping Definition of Output Properties** list, and then click **Edit**.  
Only output properties that you registered when setting the plug-in properties appear in the **Mapping Definition of Output Properties** list.
3. In the **Edit Output Filter** dialog box, enter a PCRE-compliant regular expression in the **Output Filter** area and then click **OK**.
4. You can verify the output filter you entered by entering a sample of standard output and standard error output in the **Standard Output / Standard Error Output:** text box, and clicking **Verification of the Output Filter**.  
The value filtered by the output filter appears in the **Value of the Output Property** text box.

#### Related topics

- [4.3.8 Specifying output filters](#)

### 4.3.8 Specifying output filters

This section describes how to store the standard output of a command or script in an output property.

By defining a PCRE-compliant regular expression in the **Output Filter** field, you can extract character strings from the standard output and standard error output of a command or script, and store them in the output property of a plug-in.

## Important note

- If you specify multiple groups in the regular expression, only values that match the first group are stored in the output property of the plug-in.
- If the regular expression applies to multiple value ranges, only the first range of values is stored in the output property of the plug-in. Multiple value ranges cannot be stored in an output property.

The following describes how to specify a regular expression that stores the standard output of a command or script in an output property.

- **Property key:**output01
- **Output filter:**DATE=(. \*)

When you specify an output filter in this way, the value immediately following DATE= in standard output is stored in the output property output01.

To store the return value of a script in an output property, define the plug-in and create a script as follows:

- In the **Edit Remote Command** dialog box, specify the **Script** option for **Execution Mode**.
- Create a script whose standard output displays the return value of the command or script in a format that is filtered by the regular expression specified in the output filter.

## 4.3.9 Procedure for setting scripts

This section describes how to set the script that executes on the operation-target host. To make sure that the plug-in is compatible with the operating system (Windows or UNIX) on the remote host, choose an appropriate script for each operating system. In a script, you can define scripts and commands that are present on the operation target device and which the operation target device can execute.

You can configure a script by attaching an existing script, or by entering the contents of the script directly.

When defining a script that consists of multiple files, or a script that uses a specific character set or linefeed code, follow the procedure described in *Attaching an existing script file*. When you use the procedure described in *Entering a script directly*, you can define only one script file and the character set and line-feed character are fixed for the intended operating system.

Use ASCII characters in the script file. You cannot use the following characters:

- Control characters ('\u0000' to '\u001F' and '\u007F' to '\u009F')
- Question marks (?), asterisks (\*), double quotation marks ("), right angle brackets (>), left angle brackets (<), vertical bars (|), and colons (:).

You cannot specify a file name that contains multi-byte characters.

The return values of scripts executed as plug-ins can be from 0 to 63.

Plug-ins and service templates must be designed in such a way that standard output and standard error output produce less than 100 KB of data. When the standard output or standard error output of a plug-in exceeds 100 KB, the command is immediately killed and the plug-in terminates with an error. In this scenario, the execution results of the command cannot be guaranteed.

The locale setting that applies when a script is executed depends on the operating system of the target device. For details about the locale assigned at the operation target device, see [4.1.7 Locale set for operation target devices during plug-in execution](#).

### If the **Create Plug-in dialog box** or **Edit Plug-in dialog box** is not displayed:

1. In the **Plug-in** view of the **Editor** window, display the **Under Development** tab.
2. To create a new plug-in, click the **Create** button.  
To edit an existing plug-in, select a plug-in and then click the **Edit** button.

### Attaching an existing script file

1. In the **Remote Command** area of the **Create Plug-in** dialog box or the **Edit Plug-in** dialog box, click **Edit**.
2. In the **Edit Remote Command** dialog box, specify the **Script** option for **Execution Mode**.
3. In the **Command line** text box, enter the command that executes the script.  
Although only one script file can be registered, you can register a script that consists of several files and folders in a hierarchical structure by compressing the files into a zip archive. If the script is a single file, specify the file name in the **Command line** text box. If the script is a zip archive containing multiple files, specify a relative path whose current directory is the location where the archive will be extracted.
4. In the **Setting the Scripts** area, select the **Attachment** option.
5. In the **File** area, click **Select** and register the script.

### Entering a script directly

1. In the **Remote Command** area of the **Create Plug-in** dialog box or the **Edit Plug-in** dialog box, click **Edit**.
2. In the **Execution Mode** area of the **Edit Remote Command** dialog box, select the **Script** option.
3. In the **Command line** text box, enter the command that executes the script.
4. In the **Setting the Scripts** area, select the **Direct Input** option.
5. Enter values in the **File Name** field and the **Script** text box.  
In the **File Name** field, enter the name of the file in which to store the code entered in the **Script** text box. In the **Script** text box, enter the script code.

You can later replace the file you registered on the server by selecting another file.

#### Tip

For details about how to set return values when executing a command or script, see [4.3.11 Return values of content plug-ins](#).

### Related topics

- [4.3.10 Specifying commands in the \*\*Command line\*\* text box](#)
- [4.3.12 Relationship of command and script return values to the return values of plug-ins and steps](#)
- [4.3.15 Differences between script settings methods](#)

## 4.3.10 Specifying commands in the Command line text box

The information you can enter in the **Command line** text box depends on the option specified for **Execution Mode** and whether you choose to specify the value of an input property as an argument of the command you are executing.

Note that special characters specified in the **Command line** text box, such as those used to indicate environment variables, will not be escaped. However, depending on the platform, the following characters are automatically escaped when the values of mapped input properties are passed to the command line:

- In Windows: %
- In UNIX: \$, `, \, "

When mapping an input property as a command line argument, enclose the value of the argument in double quotation marks (for example, "?dna\_*property-key-of-plug-in-property*?"). When executing a PowerShell script, you can enclose the value in double or single quotation marks.

If **Platform** is Windows and the value of an input property contains a double-quotation mark, an error will occur when the plug-in is executed.

### When Script is specified for Execution Mode:

Create the script to be executed on the target device, and enter the command that calls the script in the **Command-line** text box. If the script is a single file, specify the file name. If the script is a zip archive containing multiple files, specify a relative path whose current directory is the location where the archive will be extracted.

The script is copied to a temporary folder under the folder specified in **Execution Directory**.

If **Platform** is **AIX**, **HP-UX**, **Linux**, or **Solaris**, the command line is automatically prefixed with ./ when the command is executed. You do not need to manually add the prefix. If you specify ./ in the command line, the script file appears after ./ but still works normally. Special characters, such as those used to indicate environment variables in the command line, are not escaped.

The following shows how to enter information in the **Command line** text box when the **Script** option is specified for **Execution Mode**.

Example of specifying information in the **Command line** text box

```
cmd.exe /q /c "AAA.bat bbb ccc"
```

Example of script file (AAA.bat) contents

```
@xxx.exe %~1  
@yyy.exe %~2
```

If the value in **Platform** is **Windows**, the command is converted to a batch file and executed on the operation target device. For this reason, the results of the command might differ from those of the same command executed at the command prompt.

### When Command is specified for Execution Mode:

Enter the command to be executed on the operation target device directly in the **Command line** text box. You do not need to create a script.

The following shows how to enter information in the **Command line** text box when the **Command** option is specified for **Execution Mode**

Example of specifying information in the **Command line** text box

```
zzz.exe aaa bbb
```

### When specifying input property values as command arguments

To specify the value of an input property in an argument of a command, specify `?dna_property-key-of-plugin-property?` in the **Command line** text box.

Example of specifying information in the **Command line** text box

```
scriptA.sh -xx ?dna_input01? -yy ?dna_input02?
```

In this example, `?dna_input01?` is replaced with the value of the plug-in property `input01`, and `?dna_input02?` is replaced with the value of the plug-in property `input02`.

### When specifying a non-standard script

JP1/AO executes scripts using `cmd.exe` when the operation target device is running Windows, and the user's login shell when the device is running UNIX. If you want to run a non-standard script, you need to define the instructions required to run the executable file that implements the script.

The following shows an example of running a PowerShell script from the command prompt and establishing a connection with vCenter.

Example of specifying information in the **Command line** text box

```
powershell -executionPolicy RemoteSigned -command ".  
\vSphereConnectChallenge.ps1 '?dna_vCenterServerName?' '?dna_userName?' '?  
dna_password?' '?dna_portNumber?' '?dna_protocol?'; exit $LASTEXITCODE"  
2>&1
```

- PowerShell cannot execute scripts by default. By specifying `powershell -executionPolicy RemoteSigned` in the command line, you can execute a local PowerShell script on an operation target device of JP1/AO.
- Here, `?dna_property-key?` is a variable replaced with the value of a property. Enclose `?dna_property-key?` with double or single-quotation marks.<sup>#</sup>

This allows the properties specified in the **Command line** text box to be passed to the shell even if the property has a null value.

#

If you enclose a property with double-quotation marks (") in PowerShell and that property has a null value, PowerShell skips the property. If you enclose it in single-quotation marks ('), the property is interpreted as a null value and not skipped. By avoiding double-quotation marks ("), you can ensure that the script is executed as originally defined in terms of the order and content of arguments.

### Related topics

- [4.3.11 Return values of content plug-ins](#)
- [4.3.12 Relationship of command and script return values to the return values of plug-ins and steps](#)
- [4.3.16 Specifying Execution Directory](#)

## 4.3.11 Return values of content plug-ins

The following table lists the return values of content plug-ins:

Table 4–10: Return values of content plug-ins

Return value	Description
-1	The plug-in was forcibly terminated during execution.
0 to 63 <sup>#</sup>	The meaning of the return value differs between content plug-ins.
64	A command executed in the content plug-in terminated with a return value outside the 0 to 63 range.
65	The connection to the JP1/AO server failed. For example, the JP1/AO server might have stopped during plug-in execution.
66	The following user is mapped to the JP1 user: <ul style="list-style-type: none"> <li>• A user who does not belong to the Administrators group</li> <li>• A user with UAC enabled who is not the built-in Administrator of the Administrators group</li> </ul>
68	The system cannot find information for the applicablejob execution ID.
69	An attempt to acquire an environment variable for the task processing engine failed.
70	The connection to the remote host failed.
71	An attempt to call a command failed.
72	The status of command execution could not be acquired.
73	File transfer failed.
74	File deletion failed.
76	The connection timed out.
77	The host name of the remote host could not be resolved.
78	Authentication with the remote host failed for one of the following reasons: <ul style="list-style-type: none"> <li>• Password authentication failed.</li> <li>• Public key authentication has not been set up on the operation target device.</li> <li>• In public key authentication, the private key does not match the pass phrase.</li> <li>• In public key authentication, the private key does not correspond to the public key registered in the operation target device.</li> <li>• In public key authentication, an invalid private key was used.</li> </ul>
80	Task execution has stopped.
81	The plug-in was called in an invalid status.
82	The request message from the task-processing engine could not be correctly parsed.
83	The environment of the JP1/AO server is corrupted.
84	Information about the specified plug-in could not be obtained.
86	The specified property value is invalid.
127	An unspecified error has occurred.

#

For plug-ins created in the **Editor** window, the return values are the same as for the command or script the plug-in is executing. If the plug-in is a content plug-in provided by JP1/AO, see the description of the content plug-in in the *Job Management Partner 1/Automatic Operation Service Template Reference*.

## Related topics

- [4.3.12 Relationship of command and script return values to the return values of plug-ins and steps](#)

## 4.3.12 Relationship of command and script return values to the return values of plug-ins and steps

In most circumstances, the return value of a command or script serves as the return value of the plug-in. When content plug-ins and general command plug-ins execute commands and scripts on connection destinations, the return value of the command or script is used as the return value of the plug-in. This does not apply to the terminal command plug-in, which does not set the return value of the command or script as the return value of the plug-in.

You can use values in the range from 0 to 63 as the return value of a command or script. If the command or script returns a value outside this range, the plug-in returns 64, indicating that the returned value was outside the 0 to 63 range. If the command or script could not be executed, the plug-in returns a value of 65 or higher that indicates the cause of the failure.

Although the return value of the plug-in is generally the return value of the step, the values might differ in certain circumstances, such as an error occurring during plug-in execution or the task being forcibly terminated. In this case, take the appropriate action based on the return value of the step.

### Related topics

- [4.3.13 Procedure for using the return value of a command or script as a flow branching condition \(for values outside the 0 to 63 range\)](#)

## 4.3.13 Procedure for using the return value of a command or script as a flow branching condition (for values outside the 0 to 63 range)

When a command or script executed in a plug-in returns a value outside the 0 to 63 range, you can use the return value as the branch condition for a flow by following the procedure below.

### To use the return value of a command or script as the branch condition of a flow:

1. When creating or editing a plug-in, specify the **Script** option for **Execution Mode** in the **Edit Remote Command** dialog box.
2. Create a script that outputs the return value of the command it executes to standard output.
3. In the **Edit Output Filter** dialog box, enter a regular expression that assigns the return value of the command or script output to standard output to an output property of the plug-in.
4. Configure output property mapping so that the value of the plug-in output property assigned in step 3 is assigned to a service property (variable).
5. Configure a judge value plug-in to judge the value of the service property (variable) assigned in step 4.

### Related topics

- Judge value plug-in in the manual *Job Management Partner 1/Automatic Operation Service Template Reference*

## 4.3.14 Information output to standard output by plug-ins

The standard output and standard error output of the commands and scripts specified in the **Command line** text box serve as the standard output of the plug-in.



However, plug-ins and service templates must be designed so that the standard output of the plug-in does not exceed 100 KB. If the standard output of a plug-in exceeds this size, the command is immediately forcibly terminated and the plug-in ends in an error. In this situation, the results of the command cannot be guaranteed.

The size of standard output includes the data added by JP1/AO. For this reason, you need to include some leeway over and above the standard output and standard error output of the plug-in when estimating the standard output of a plug-in.

### Size of plug-in standard output (when the operation target device is running UNIX)

Number of linefeed codes (LF) × bytes

When the operation target device is running UNIX, the carriage return character CR(0x0d) is replaced with the linefeed character LF(0x0a) in standard output and standard error output. LF(0x0a) is appended to the end of standard output and standard output if the last character is not a linefeed character (CR/LF/CR+LF).

- LF(0x0a) is left unchanged
- CR(0x0d) is replaced with LF(0x0a)
- CR+LF(0x0d0a) is replaced with LF+LF(0x0a0a)

## 4.3.15 Differences between script settings methods

There are two ways to configure a script: attaching an existing script, or entering the contents of the script directly. The difference between these methods is described below.

When **Attachment** is selected

You can register a script composed of multiple files and folders by archiving them as a file in zip format.

Archives in zip format are renamed when the plug-in is placed on the JP1/AO server. The file name is changed to windows.zip, aix.zip, hpux.zip, linux.zip, or solaris.zip depending on the operating system. Files registered with Attachment selected can be downloaded to a terminal where JP1/AO is operated through a Web browser. If you enter the script directly, you cannot download the script file.

When **Direct Input** is selected

In the **Edit Remote Command** dialog box, enter the contents of the script to execute on the operation target device directly. You can only define scripts and commands that are present and can be executed on the operation-target device. You can only define one script file. The character set and linefeed code applied to the saved script file are fixed values appropriate to the operating system of the operation-target device.

Therefore, if you want to register a script file that consists of multiple files, or assign a specific character set or linefeed code, select **Attachment**.

The following table shows the differences between file registration methods and available character sets and linefeed codes depending on the setting in the **Setting the Scripts** area.

Table 4–11: Differences between script setting methods

Item	With Attachment selected		With Direct Input selected	
	Windows	AIX, HP-UX, Linux, Solaris	Windows	AIX, HP-UX, Linux, Solaris
Registering single file	Can be registered		Can be registered	
Registering multiple files	Can be registered		Cannot be registered	

Item	With Attachment selected		With Direct Input selected	
	Windows	AIX, HP-UX, Linux, Solaris	Windows	AIX, HP-UX, Linux, Solaris
Character set of saved script	Assigned character set of registered file		Default character set of JP1/AO server OS	UTF-8
Linefeed code of saved scripts	Assigned linefeed code of registered file		CR+LF	LF

## Related topics

- [4.3.9 Procedure for setting scripts](#)
- [4.3.12 Relationship of command and script return values to the return values of plug-ins and steps](#)

## 4.3.16 Specifying Execution Directory

In the **Execution Directory** field, specify the absolute path of the folder in which to execute the script or command. Only use characters that are supported in command lines on the JP1/AO server and the operation target device.

Do not enclose the path of the execution directory in double or single-quotation marks, even if the path contains spaces. Plug-in execution will fail if the path is enclosed in quotation marks. The execution directory must be created in advance on the operation target host. If you do not create the execution directory in advance, plug-in execution might fail.

The behavior of the system depends on the option selected for **Execution Mode**, as described below.

### When Script is specified for Execution Mode

JP1/AO copies the script to a uniquely-named temporary subfolder of the folder specified in **Execution Directory**. The script is executed in this temporary folder. The script and temporary folder are deleted when the script has finished executing.

### When Command is specified for Execution Mode

You can specify the execution directory in the **Execution Directory** field, and in the property file (`config_user.properties`). The execution directory is set in the following order of priority:

### When the operation target host is running Windows

1. The value specified in the Execution Directory field
2. The value specified for the `plugin.remoteCommand.executionDirectory.wmi` property in the property file (`config_user.properties`)
3. The value of the `%TEMP%` environment variable on the operation target host

### When the operation target host is running UNIX

1. The value specified in the Execution Directory field
2. The value specified for the `plugin.remoteCommand.executionDirectory.ssh` property in the property file (`config_user.properties`)
3. `/tmp`

## 4.3.17 Procedure for setting commands

Set the command that is appropriate for the operating system on the operation target host (Windows, AIX, HP-UX, Linux, or Solaris). When creating a plug-in that is compatible with multiple operating systems, set commands for each operating system.

The commands you set must be scripts or commands that are present on and executable by the operation target device.

The commands executed in plug-ins can have return values in the range from 0 to 63.

Plug-ins and service templates must be designed in such a way that standard output and standard error output produce less than 100 KB of data. When the standard output or standard error output of a plug-in exceeds 100 KB, the command is immediately killed and the plug-in terminates with an error. In this scenario, the execution results of the command cannot be guaranteed.

### If the **Create Plug-in dialog box** or **Edit Plug-in dialog box** is not displayed:

1. In the **Plug-in** view of the **Editor** window, display the **Under Development** tab.
2. To create a new plug-in, click **Create**.  
To edit an existing plug-in, select the plug-in and then click **Edit**.

### To set a command:

1. In the **Remote Command** area of the **Create Plug-in** dialog box or the **Edit Plug-in** dialog box, click the **Edit** button for the operating system of the operation target device.
2. In the **Execution Mode** area of the **Edit Remote Command** dialog box, click **Command**.
3. Enter the command line in **Command line**, and then click **OK**.

### Related topics

- [4.3.10 Specifying commands in the \*\*Command line\*\* text box](#)

## 4.4 Deleting plug-ins

---

### 4.4.1 Procedure for deleting plug-ins

You can delete plug-ins that appear in the **Under Development** tab and the **Release** tab. When you delete a plug-in, the definition of the plug-in is deleted from the JP1/AO server, and the plug-in disappears from the **Under Development** and **Release** tabs.

However, you cannot delete a plug-in that is being used by a development service template or a release service template. Nor can you delete a plug-in if a development service template has been built that uses that plug-in as a step. In this scenario, delete the step from the development service template, and build the service template again. You will then be able to delete the plug-in.

#### To delete a plug-in:

1. In the **Plug-in** view of the **Editor** window, display the **Release** tab or the **Under Development** tab.
2. Select the plug-in you want to delete, and from the **Actions** pull-down menu, choose **Delete**.
3. In the confirmation dialog box, click **OK**.  
The plug-in is deleted.

## 4.5 Copying plug-ins

---

### 4.5.1 Procedure for copying plug-ins

You can copy a development plug-in or release plug-in to create a new development plug-in that retains the settings of the original. Use this procedure when you want to develop a new plug-in based on an existing plug-in, or to create a modified version of an existing plug-in.

#### To copy a plug-in:

1. In the **Plug-in** view of the **Editor** window, click the **Release** tab or the **Under Development** tab.
2. Select the plug-in you want to copy, and from the **Actions** pull-down menu, choose **Copy**.
3. In the **Copy Plug-in** dialog box, change at least one of the plug-in ID, plug-in version, and vendor ID, and then click **Save**.

You cannot specify a vendor ID that begins with `com.hitachi.software.dna` because such IDs are reserved by JP1/AO. If the vendor ID of the original plug-in begins with `com.hitachi.software.dna`, the vendor name and vendor ID are removed when the plug-in is copied.

After copying a plug-in, you can then set the definition information for the plug-in.

#### Related topics

- [4.2.2 Parameters to set when creating or copying plug-ins](#)
- [4.3.1 Procedure for editing plug-in definition information](#)

## 4.6 Using resource files to set plug-in display information

### 4.6.1 Procedure for setting plug-in resource files

You can set the information displayed for a plug-in by entering settings in a plug-in resource file. You can edit the plug-in resource file directly by downloading the file and overwriting it.

You cannot set display information for plug-ins in released service templates.

#### To set a plug-in resource file:

1. Select a plug-in in the **Plug-in** view of the **Editor** window.
2. From the **Actions** pull-down menu, choose **Set Resources**.
3. In the **Set the Plug-in Resources** dialog box, download the plug-in resource file by clicking the link.
4. Edit the plug-in resource file you downloaded.
5. Click the **Refresh** button, select the plug-in resource file you edited, and upload the file.  
If the plug-in resource file you uploaded is not named `plugin_<language-code>.properties.txt`, an error occurs.
6. In the conformation dialog box, click **OK**.

#### Important note

When you upload the plug-in resource file, the existing file is overwritten with the contents of the new file. Take care not to upload the wrong file.

The file name of the uploaded file must be `plugin_<language-code>.properties.txt`, or an error occurs.

*language-code* is a two-character language code (ja, en, or zh) as defined in ISO-639.

#### Related topics

- [4.6.2 Format of plug-in resource files](#)
- [4.6.3 Correspondence between properties in plug-in resource files and information displayed for plug-ins](#)

### 4.6.2 Format of plug-in resource files

A plug-in resource file defines the information displayed in the JP1/AO user interface. The file has the following format:

- The file name of the plug-in resource file is `plugin_<language-code>.properties.txt`.  
*language-code* is a two-character language code (ja, en, or zh) as defined in ISO-639.
- Define the file contents in the format *property-key**delimiting-character**setting-value*. As the delimiting character, you can use an equals sign (=), a colon (:), a tab character (\t), or a single-byte space.  
For example, enter a definition in the format `plugin.displayName=TestPlugin`.
- Enter one property key and setting per line.

- Property keys can be 1 to 128 characters long, and can contain the following characters:
  - Single-byte alphanumeric characters
  - Single-byte hyphens (-)
  - Single-byte underscores (\_)
  - Single-byte periods (.)
- Characters must be encoded in UTF-8.
- If you define the same property key in the file more than once, the value of the last occurrence of the property key applies.
- Lines that begin with a hash mark (#) are handled as comments.
- Property keys are case sensitive.
- To specify a character string that contains a forward slash (\), specify two forward slashes (\\) instead.
- Lines that consist only of single-byte spaces are ignored.
- On each line of the plug-in resource file, the property key is the character string from the first character that is not a single-byte space to the character immediately preceding the first delimiting character.
- The setting value is the string from the first non-delimiting character after the delimiting character following the property key to the last character of the line.  
 For example, the following line represents the property key `abc` with the setting value `=\tc`:  
`abc\t=\tc`  
 However, if the character immediately following the first delimiting character is `=` or `:`, the setting value is the character string from the next character that is not a single-byte space or `\t` to the end of the line.  
 For example, the following line in the service resource file represents the property key `abc` with the setting value `=\tc`.  
`abc\t=\t=\tc`
- Surrogate pair characters are ignored.

### 4.6.3 Correspondence between properties in plug-in resource files and information displayed for plug-ins

You can set the display information for plug-ins managed in the **Editor** window from the user interface. The following table lists the correspondence between the display information of the plug-in and the properties in the plug-in resource file.

Table 4–12: Correspondence between properties in plug-in resource file and display information

Plug-in display information	Property in plug-in resource file
Plug-in name	<code>plugin.displayName</code>
Vendor name	<code>plugin.vendorDisplayName</code>
Description	<code>plugin.shortDescription</code>
Input property name/Output property name	<code>property.<i>property-key</i>.displayName</code>
Input property description/Output property description	<code>property.<i>property-key</i>.description</code>

## 4.6.4 Plug-in resource files automatically generated when a plug-in is created

When a plug-in is created, two different plug-in resource files are automatically generated. One is for the same language as the Web browser locale, and the other is for English.

The following values are set in the generated plug-in resource files.

Table 4–13: Display information values set by default for plug-in resource files (when a plug-in is created)

Defined display information	Values set by default	
	Same language as the Web browser locale	Automatically generated resource file for English <sup>#</sup>
Vendor name	Values specified in the <b>Editor</b> window	Vendor ID
Plug-in name		Plug-in ID
Plug-in description		Blank
Plug-in input property name or output property name		Property key
Plug-in input property description or output property description		Blank

#

For the contents of the plug-in resource file generated when the Web browser locale is English, see the "Same language as the Web browser locale" column.

Examples of plug-in resource files that are automatically generated when a plug-in is created are shown below.

### Values specified in the window

```
plugin.vendorDisplayName=テスト用  
plugin.displayName=テスト部品  
plugin.shortDescription=テスト用部品です
```

### Generated plug-in resource file (Japanese)

```
ベンダー ID : test.vendor  
部品 ID : test.plugin  
部品バージョン : 10.00.00  
部品名 : テスト部品  
ベンダー名 : テスト用  
説明 : テスト用部品です
```

### Generated plug-in resource file (English)

```
plugin.vendorDisplayName=test.vendor  
plugin.displayName=test.plugin  
plugin.shortDescription=
```



## 4.6.5 Plug-in resource files updated when a plug-in is edited

When a plug-in is edited and saved, the plug-in resource file for the same language as the Web browser locale is updated.

If a display information definition is added or deleted or if the property key is updated, the update is also reflected in the other plug-in resource file for the non-Web browser locale language, for consistency.

The table below describes the values that will be added to the plug-in resource files for locales other than the Web browser's locale when the definition of a display item is added. If the definition of a display item is updated, the values in this table will automatically overwrite the existing values in those files. Note, however, that overwriting of values takes place only when the property key is updated.

If you might need to reference the values that existed before being overwritten, back up the plug-in resource files for locales other than the Web browser's locale.

Table 4–14: Values of the display items that will be set in plug-in resource files

Defined display information	Value that will be set
Plug-in input property name or output property name	Property key
Plug-in input property description or output property description	Blank

If a display information definition is deleted, the definition is also deleted from the plug-in resource file for the non-Web browser locale language.

To specify display information for a non-Web browser locale language, the plug-in resource file must be created or edited manually before uploading. To manually create a plug-in resource file, we recommend that you download and use the appropriate plug-in resource file for a locale for which display information is defined.

## 4.6.6 Displaying a plug-in on a Web browser set to a locale for which no plug-in resource file is available

If a plug-in is displayed on a Web browser set to a locale for which no plug-in resource file is available, the plug-in resource file for English will be loaded to display the plug-in.

# 5

## Validating Service Templates

After creating a service template, you can perform a validation process to make sure that it will operate as intended in the active environment.

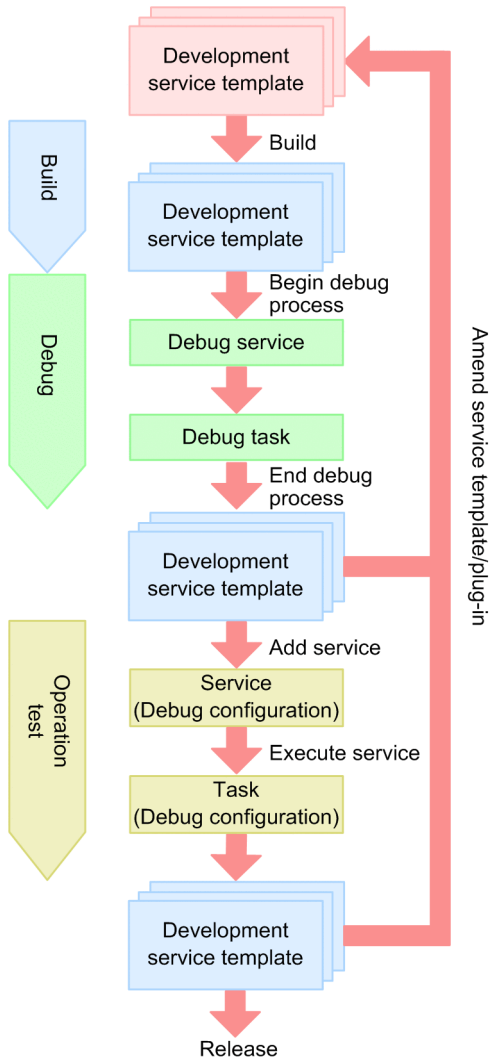
## 5.1 Overview of service template validation

### 5.1.1 Flow of service template validation

After creating or editing a flow, you can check for issues in the flow transitions and plug-in processing by building and debugging a development service template. If the debug process reveals an issue, you can edit the affected service template or plug-in, build the development service template again, and repeat the debug process. When all the issues have been resolved, you then test the operation of the service by executing it in the development environment.

The following figure shows the flow of service template validation:

Figure 5–1: Flow of service template validation



#### Build

1. After creating or editing a flow, prepare it for testing by building a service template.

#### Debug

1. If the build process is successful, debug the service template to identify issues in its flow or plug-ins. The debug process creates a debug service and debug task.

2. Review the execution results of the debug task, and resolve any issues by amending the service template. Then, build the amended service template again and repeat the debug process.

Repeat the process of amending, building, and debugging the service template until all problems are resolved.

## Operation test

1. After all issues have been resolved, conduct an operation test by adding and executing the service in the development environment.

Check the execution results of the task. If any issues are identified, amend the affected service template or plug-in and repeat the debug process.

When there are no further issues, release the service template.

Note that debugging and operation testing can be performed on an as-needed basis.

## Related topics for building service templates

- [5.1.2 Overview of building](#)
- [5.2 Building service templates](#)

## Related topics for debugging service templates

- [5.1.3 Overview of debugging](#)
- [5.3 Debugging service templates](#)
- [5.4 Managing debug tasks](#)

## Related topics for testing service template operation

- [5.1.4 Overview of operation tests](#)
- [5.5 Testing service templates](#)

## 5.1.2 Overview of building

Building is the process of preparing a service template you created or edited in the **Editor** window for testing. When successful, a build operation creates a package of the service template which you can then use to debug the service template or add a service to the server.

### Objective

Perform a build operation to prepare a development service template for validation. The service template you build is packaged as a debug configuration service template and imported to the JP1/AO server.

### Number of executions

You can build a service template any number of times. If the debug process or an operation test reveals an issue with the service template, the user repeats the series of operations from amending the development service template or development plug-in to checking its operation until all issues are resolved.

When you edit and rebuild a service template that has already been through a debug process, JP1/AO deletes the debug service and debug task generated the last time the service template was built. When you rebuild a service template after testing its operation, JP1/AO deletes the services added from the service template and archives tasks. For details about the services and tasks archived and deleted during build operations, see [A.1\(5\) Deletion and archiving of service templates, services, and tasks during build and release operations](#).

## Assigned configuration type

After building, a service template is assigned the Debug configuration type. Only users assigned the Admin or Develop role can view and work with Debug configuration type service templates and the associated services and tasks.

## Output destinations of service templates

When you build a service template, a service template package is created in the following folder with the name *vendor-ID\_name\_version\_d.st*:

In a non-cluster system

*JPI/AO-installation-folder*\develop\output

In a cluster system

*shared-folder-name*\develop\output

## Related topics

- [2.1.2 Overview of development service templates and release service templates](#)
- [5.1.1 Flow of service template validation](#)
- [5.2.1 Procedure for building a service template](#)
- [A.1 Reference information for build and release operations](#)

## 5.1.3 Overview of debugging

Debugging is the process of using the **Debug** view and the service template debugging view to check the operation of a service template you have built, and identify issues in its flow or plug-ins. When you debug a service template, JPI/AO creates a debug service and debug task. The debug process involves executing this debug task.

If the debug process reveals an issue, the user stops debugging and edits the affected service template or plug-in.

### Objective

Perform a debug operation to make sure the flow and plug-ins of a service template are working as intended. For example, you can confirm that property mapping is set correctly and that the conditions for executing subsequent steps branch the flow in the intended way. In the **Debug** view and the service template debugging view, you can:

- Execute debug tasks while checking the flow transitions at all hierarchical levels (including hierarchy flows and repeated flows) and the results of plug-in processing.
- Execute debug tasks while making sure that property values are assigned correctly by the property mapping configured for the service template.
- If you detect an issue with a plug-in, you can assign an arbitrary property value or return value to the plug-in and execute it again. This allows you to see the effect a given property value or return value has on the plug-in processing and flow transitions.

### Number of executions

You can debug a service template any number of times. If the debug process reveals an issue with the service template, the user can repeat the series of operations from amending the development service template or development plug-in to building and debugging until all issues are resolved.

## Debug services

A debug service is a service generated and executed when debugging a service template. One debug service is generated per service template. When you debug a service template that has already been through a debug process, JP1/AO deletes the existing debug service and creates a new one.

Note that debug services appear in the Service Name column in the **Debug-Tasks** view, but do not appear in the **Services** window.

## Debug tasks

A debug task is a task generated for a debug service when debugging a service template. When you debug a service template that has already been through a debug process, JP1/AO deletes the existing debug task and creates a new one.

Debug tasks appear in the service template debugging view and **Debug-Tasks** view. Only users assigned the Admin or Develop role can view and work with debug tasks.

Note that debug tasks do not appear in the task summary.

## Related topics

- [2.1.2 Overview of development service templates and release service templates](#)
- [5.1.1 Flow of service template validation](#)
- [5.3.1 Flow of service template debugging](#)
- [5.3.2 Functions used during debug operations](#)
- [5.3.3 Example of service template debugging](#)

## 5.1.4 Overview of operation tests

An operation test is the process of adding a service from a service template you have built, and executing it in the development environment. This provides final confirmation that the service template is ready for real-world use.

If the operation test reveals a problem, the user can edit the service template or plug-in in the **Editor** window.

## Objectives

Perform an operation test by executing a service generated from the service template to confirm that it will work correctly in the active environment. The user can also test the usability of the service template by adding and executing services from the **Services** window in a way that reflects real-world use. For example, the user can specify a schedule for the service, check whether it operates as intended, and make sure that the appropriate properties are visible in the user interface.

## Number of executions

You can conduct any number of operation tests for a given service template. If the operation test reveals an issue with the service template, the user repeats the series of operations from amending the development service template or development plug-in to building, debugging, editing, and testing it until all issues are resolved.

## Related topics

- [2.1.2 Overview of development service templates and release service templates](#)
- [5.1.1 Flow of service template validation](#)
- [5.5.1 Procedure for testing service templates](#)

- Managing services in the *Job Management Partner 1/Automatic Operation Administration Guide*
- Executing Services in the *Job Management Partner 1/Automatic Operation Administration Guide*
- Managing Tasks in the *Job Management Partner 1/Automatic Operation Administration Guide*

## 5.2 Building service templates

---

### 5.2.1 Procedure for building a service template

When you select and build a service template, a service template package is created and imported to the JP1/AO server. You can then debug and test the service template.

#### Important note

After you edit the definition of a plug-in that is positioned as a step in a flow, the mapping parameters defined for the step might no longer match the properties of the edited plug-in. In this scenario, an error occurs when you build the service template. You can resolve the mismatch by reviewing the property settings and plug-in placement.

#### If the service template editing view is not displayed:

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Edit**.
2. In the **Service Template List** window, select a service template, and then click **Edit**.

#### To build a service template:

1. In the service template editing view, click **Build/Debug**.
2. In the confirmation window, click **OK**.  
The results of the build process appear in the **Build / Release Result** dialog box.
3. If you want to then debug the service template, click **Perform Debugging**. For details about how to start the debug process, see [5.3.4 Procedure for starting the debug process](#).  
If you do not want to debug the service template, click **End Debug**.

If an error occurs during the build process, a message is displayed that describes the cause of the error and instructs the user to fix the flow. The error message remains on screen until the user closes the service template editing view.

If the service template you are working with has already been released or deleted by another user, an error occurs and the build process fails.

#### Related topics

- [2.1.4 Behavior when an intervening action occurs in the Editor window](#)
- [2.1.2 Overview of development service templates and release service templates](#)
- [5.1.1 Flow of service template validation](#)
- [5.1.2 Overview of building](#)
- [A.1 Reference information for build and release operations](#)



## 5.3 Debugging service templates

### 5.3.1 Flow of service template debugging

The following describes the general procedure for debugging a service template:

1. If you do not expect any issues when executing the plug-ins in the service template, the first step of the debug process is to execute the debug task without pausing between steps.  
In the **Debug** view or the service template debugging view, make sure that there are no issues with the flow transitions or the processing of the plug-ins.  
If the service template contains plug-ins that you would prefer not to execute at this time, skip this step and start from step 2 instead.
2. If you identify an issue with a flow transition or the processing of a plug-in, execute the steps in the debug task individually to identify the precise location and nature of the problem. You can also test the behavior of the plug-ins by assigning unexpected values to input and output properties. You can debug the service template any number of times by clicking **Perform Debugging** in the service template debugging view.
3. Amend the service template in the service template editing view.
4. Build and debug the service template again, repeating steps 1 to 4 until all issues are resolved.
5. Make sure that all issues have been resolved, and finish the debug process.

#### Tip

After executing a debug task, you do not need to perform the build operation again if you repeat the debug process without amending the service template.

#### Related topics

- [5.1.1 Flow of service template validation](#)
- [5.1.2 Overview of building](#)
- [5.1.3 Overview of debugging](#)
- [5.3.2 Functions used during debug operations](#)
- [5.3.3 Example of service template debugging](#)

### 5.3.2 Functions used during debug operations

The table below lists the functions used when debugging service templates. When debugging service templates, use the functions that are appropriate for what you want to achieve.

Table 5–1: Functions used during debug operations

Function	Description	Reference
Interruption settings for	There are two ways to execute a debug task: <ul style="list-style-type: none"><li>• Do not break after each step</li></ul>	<a href="#">5.3.4 Procedure for starting the debug process</a> , <a href="#">5.3.7 Flow of debug process without pausing between</a>

Function	Description	Reference
debug operations	<p>Like execution of a normal task, processing of the debug task proceeds without pausing between steps. You can check the execution results of the debug task and identify steps where issues occur.</p> <ul style="list-style-type: none"> <li>• Break after each step</li> </ul> <p>Execution of the debug task pauses between each step. This allows you to check the values of plug-in properties and return values step by step.</p>	steps, 5.3.8 Flow of debug process when pausing between steps
Displaying flow information	<p>There are two ways to display the flow of a debug task:</p> <ul style="list-style-type: none"> <li>• Flow view</li> </ul> <p>In the service template debugging view and <b>Tasks</b> window, you can view the status of steps and the flow transitions at each hierarchical level.</p> <ul style="list-style-type: none"> <li>• Flow Tree view</li> </ul> <p>In the service template debugging view, you can view flow hierarchies in tree format. You can also identify paused steps in each flow hierarchy.</p> <p>You can also view flow hierarchies in tree format from the <b>Tasks</b> window.</p>	5.3.24 Displaying the flow of a debug task, 5.3.25 Displaying the flow tree of a debug task, 5.4 Managing debug tasks
Displaying task logs	<p>The contents of the task log appear in the <b>Task Log</b> tab of the service template debugging view. You can configure JP1/AO to automatically refresh the contents of the task log.</p>	5.4.4 Procedure for checking task log entries for debug tasks
Checking property mapping	<p>You can display the values of plug-in properties and the property mapping settings in the <b>Debug</b> view. By viewing this information together with the service property values displayed in the <b>Properties</b> tab of the service template debugging view, you can check whether property mapping is set up correctly.</p>	5.3.16 Procedure for checking property mapping settings during debugging
Changing information in a step <sup>#</sup>	<p>You can change a property value or return value of a plug-in to any value.</p> <ul style="list-style-type: none"> <li>• You can pause a step before its plug-in processing is executed and change the value of input properties.</li> <li>• You can pause a step after its plug-in processing is executed and change the value of output properties or the return value of the plug-in.</li> </ul>	5.3.17 Procedure for changing the value of a plug-in input property during debugging, 5.3.18 Procedure for changing the value of a plug-in output property during debugging, 5.3.19 Procedure for changing plug-in return values during debugging
Skipping plug-in processing <sup>#</sup>	<p>You can skip plug-in processing and continue processing as if execution of the step had completed.</p>	5.3.12 Procedure for skipping plug-in processing during the debug process
Retrying tasks	<p>There are two approaches to retrying a failed debug task:</p> <ul style="list-style-type: none"> <li>• Retry the task from the failed step</li> </ul> <p>You can resume task execution from the failed step.</p> <ul style="list-style-type: none"> <li>• Retry the task from the step after the failed step</li> </ul> <p>You can resume task execution from the next step, as if the failed step had finished normally.</p>	5.3.13 Procedure for retrying a task from a failed step during debugging, 5.3.14 Procedure for retrying a task from the step after the failed step during debugging
Managing debug tasks	<p>You can perform the following operations in relation to debug tasks:</p> <ul style="list-style-type: none"> <li>• Display a list of all debug tasks</li> <li>• Stop execution of a debug task</li> <li>• Forcibly stop a debug task</li> <li>• Delete a debug task</li> </ul>	5.4 Managing debug tasks

#

This function is available when Interrupt After Each Step is selected as the break setting for the debug process.

## Related topics

- [5.3.1 Flow of service template debugging](#)
- [5.3.3 Example of service template debugging](#)

### 5.3.3 Example of service template debugging

When debugging a service template, users need to adjust various settings according to the aspects of the service template they want to check.

An example of the operations a user performs when debugging a service template is shown below.

This example describes the procedure for debugging a service template in the following scenario:

#### Issues with service template being debugged

Issue 1:

There is an issue with the mapping between a service property and an input property of Step A.

#### User objectives

Objective 1:

To check the execution results of other steps before amending the service template.

Objective 2:

To check the processing of Plug-in A.

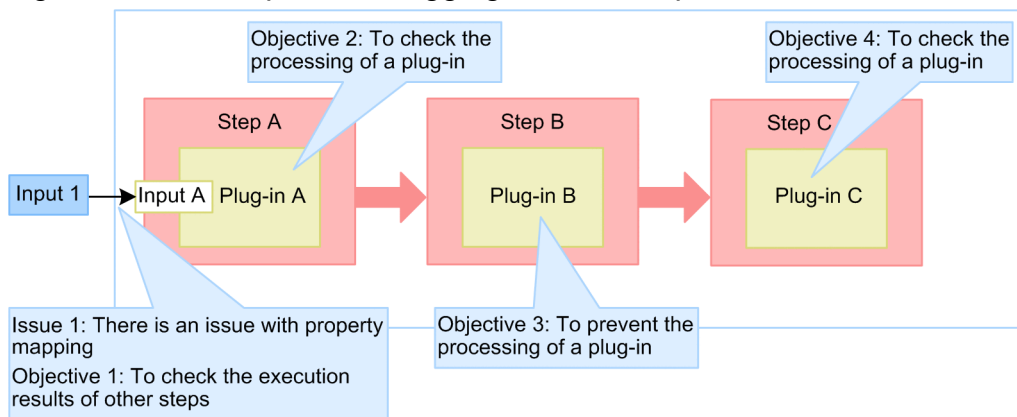
Objective 3:

To prevent the processing of Plug-in B from being executed.

Objective 4:

To check the results of Plug-in C.

Figure 5–2: Example of debugging service templates



1. As indicated by Objective 3, there is a plug-in within the flow that the user does not want to execute. Therefore, the user needs execution of the debug task to pause between steps. To achieve this, select **Interrupt After Each Step** from the **Interruption Settings** list box in the **Perform Debugging** dialog box.
2. Specify the definition information for the service, and click **Run**.  
The **Debug** view and the service template debugging view appear.
3. In the **Debug** view, check the value of Input property A in Step A. Make sure that the value of Input service property 1 has been mapped correctly in the **Properties** tab.  
Through this process, you can identify the problem with the input property mapping of Step A (Issue 1).

4. To check the processing of Plug-in A when the value of the input property is set correctly (Objective 2), change the value of input property A in the **Debug** view.
5. In the **Debug** view, click **Rerun** to execute the processing of Plug-in A.
6. Check the results of Step A in the **Debug** view, **Flow** view, and **Task Log** tab.
7. To check the execution results of the other steps (Objective 1), you need to advance the processing of the debug task. In the **Flow** view, select Step B.
8. To skip the processing of Plug-in B (to meet Objective 3), from the **Execution Settings** list box in the **Debug** view, select **Do not execute the plug-in processing**.
9. In the **Debug** view, click **Rerun** to execute Step B.  
The processing of Plug-in B is skipped and Step B finishes executing.
10. In the **Flow** view, make sure that Step C has begun executing according to the subsequent-step execution condition of Step B.
11. In the **Debug** view, click **Rerun** to execute the processing of Plug-in C.  
The debug tasks ends normally.
12. To meet Objective 4, check the execution results of Step C in the **Debug** view, **Flow** view, and **Task Log** tab.
13. In the service template debugging view, click **End Debug**.  
The service template editing view appears.
14. Amend the property mapping settings of Step A.
15. After saving the service template, build the service template again and begin the debug process.

## Related topics

- [5.1.1 Flow of service template validation](#)
- [5.1.2 Overview of building](#)
- [5.1.3 Overview of debugging](#)
- [5.3.1 Flow of service template debugging](#)
- [5.3.2 Functions used during debug operations](#)

## 5.3.4 Procedure for starting the debug process

When JP1/AO has successfully built the service template, you can begin the debug process. This involves selecting the break setting for debug service execution and setting the definition information for the debug service and debug task.

Note that schedule information is ignored during debug operations. All plug-ins are executed immediately.

### To start the debug process:

1. From the **Interruption Settings** list box in the **Perform Debugging** dialog box, select the execution method for the debug task.  
To pause after each step as you execute the debug task, select **Interrupt After Each Step**. To execute the debug task without pausing between steps, select **Do Not Break**.

2. Set the definition information for the debug service. For details about the parameters you can set in the **Perform Debugging** dialog box, see [5.3.5 Settings used when beginning the debug process](#).
3. Click the **Run** button.  
The **Debug** view and the service template debugging view appear in the **Editor** window, and the debug task is executed.
4. If you select **Interrupt After Each Step** as the break setting, execution of the debug task pauses after each step. You can resume execution of the task by clicking **Rerun** in the **Debug** view.

### Important note

- The break setting governs whether execution of the debug task pauses between steps. If you want to check the execution results of each step as you go, or the task contains plug-ins you do not want to execute at that time, select **Interrupt After Each Step** as the break setting.
- The **Perform Debugging** dialog box shows the service template as it was configured when last built by the current user. If another user edits and builds the same service template after the debugging user, the changes do not apply to the contents of the service template shown in the **Perform Debugging** dialog box.
- A debug task is forcibly terminated in the following circumstances:
  - The JP1/AO server stops.
  - Failover to another node in a cluster system occurs.
  - The user logs out of JP1/AO.
- Closing your Web browser during the debug process does not stop the debug task. If you close your Web browser, processing of the debug task continues until reaching a point where it would normally stop. If you have configured JP1/AO to break between each step of the debug task, processing of the debug task remains paused indefinitely at the current or succeeding step. If this occurs, the user needs to log in again and stop the debug task from the **Debug-Tasks** view of the **Tasks** window. The **Debug** view and the service template debugging view do not re-appear. To execute the debug task again, you need to rebuild the service template and restart the debug process.
- If another user builds the same service template in the period of time after you build the service template but before you begin the debug process, an error occurs after you click **Run** in the **Perform Debugging** dialog box.
- A single JP1/AO system can execute a maximum of 10 plug-ins concurrently in a debug task. For details about the maximum number of plug-ins that can be executed at one time and what happens when the number exceeds this limit, see *Maximum number of plug-ins contained in a task that can be executed concurrently* in the *Job Management Partner 1/Automatic Operation Administration Guide*.
- The status of debug tasks is subject to JP1 event reporting and email notification.

### Related topics

- [5.3.2 Functions used during debug operations](#)
- [1.1.3 Main windows used to develop service templates](#)
- [5.2.1 Procedure for building a service template](#)
- [5.3.1 Flow of service template debugging](#)
- [5.3.7 Flow of debug process without pausing between steps](#)
- [5.3.8 Flow of debug process when pausing between steps](#)

## 5.3.5 Settings used when beginning the debug process

In the **Perform Debugging** dialog box, you can enter the break setting, service name, category, task name, task description, resource group, task log output level, and property values.

Table 5–2: Items set in **Perform Debugging** dialog box

Item	Description
Interruption Settings	As the break setting for the debug task, select one of the following: <ul style="list-style-type: none"><li>• Do Not Break The debug task is executed without pausing between steps.</li><li>• Interrupt After Each Step Execution of the debug task pauses between each step.</li></ul>
Service Name	Enter the name of the debug service. The default is [DEBUG]name-of-executed-service-template.
Category	Enter the category of the debug service. The default is category-of-executed-service-template.
Task Name	Enter the name of the debug task. The default is [DEBUG]name-of-executed-service-template_current-time. The format of current-time is YYYYMMDDhhmmss.
Task Description	Enter a description of the debug task. This field is empty by default.
Resource Group	Select the resource group in which to register the debug service. The default is All Resources.
Task Log Level	Select the level of messages output to the task log. The default is 40. The log level you select applies only to the debug task. It has no effect on the existing shared built-in service property com.hitachi.software.dna.sys.task.log.level.
Property	Set the values of the input properties of the service. Default values are assigned according to the service property definitions set when creating the service template. The property values set in this area are specific to the debug task, and do not affect service share properties elsewhere in the system. Therefore, the values you set will not affect other services that reference the service share properties.

## 5.3.6 Procedure for debugging a service template again without rebuilding

After executing a debug task, you can debug the same service template again by running the debug task from the service template debugging view that is already displayed. In this case, you do not need to build the service template again.

When you initiate another debug process, JP1/AO deletes the debug service and debug task generated during the last debug process, and generates a new debug service and debug task.

### To execute a new debug task without building the service template again:

1. If the debug task is in completed or failed status, click **Perform Debugging** in the service template debugging view.  
If the debug task is still in progress, forcibly stop it.
2. In the **Perform Debugging** dialog box, enter the interruption settings and definition information for the debug service.
3. Click **Run**.  
The debug task is executed.

## Important note

- The **Perform Debugging** dialog box shows the service template as it was configured when last built by the current user. If another user edits and builds the same service template in the meantime, the changes do not apply to the contents of the service template shown in the **Perform Debugging** dialog box.
- You can only debug a service template again without building it again if the service template debugging view is still displayed. If you close the service template debugging view by clicking the **End Debug** button or logging out of JP1/AO, you need to rebuild the service template before debugging it.

### Related topics

- [5.3.1 Flow of service template debugging](#)
- [5.3.5 Settings used when beginning the debug process](#)
- [5.4.6 Procedure for forcibly stopping debug tasks](#)

## 5.3.7 Flow of debug process without pausing between steps

Like a normal task, processing of the debug task proceeds without pausing between steps. You cannot change the values of plug-in properties or the return value of the plug-in.

You might use this approach when you want to make sure a task still ends normally after you edit the service template or plug-in, or you want to identify problems with the flow transitions.

### Debugging without pausing between steps:

1. In the **Perform Debugging** dialog box, select **Do Not Break** in the **Interruption Settings** list box.
2. Specify the definition information for the debug service.
3. Click **Run**.  
The **Debug** view and the service template debugging view appear, and the debug task is executed.
4. Check the execution results in the **Debug** view and the service template debugging view.
5. After the debug task has finished, edit the service template or plug-ins as needed.

### Related topics

- [5.3.4 Procedure for starting the debug process](#)
- [5.3.5 Settings used when beginning the debug process](#)
- [5.3.16 Procedure for checking property mapping settings during debugging](#)
- [5.3.24 Displaying the flow of a debug task](#)
- [5.4.4 Procedure for checking task log entries for debug tasks](#)

## 5.3.8 Flow of debug process when pausing between steps

You can configure JP1/AO to pause after each step as it executes the debug task. This gives the user the opportunity to perform the actions below. If a flow includes multiple steps executed in parallel, you can perform these actions for each step.

- Check the flow transitions and plug-in processing at the level of individual steps
- Select whether to execute the processing of a particular plug-in
- Change the values of plug-in properties and the return values of plug-ins

You might use this approach when you have a general idea of where a problem is located in the flow, and want to keep a close eye on the task as it executes. You could also use this procedure to see how the task behaves with unexpected property values.

### Important note

If a plug-in returns a value of 65 or higher, the processing of the debug task will not pause after the plug-in, even if configured to do so in the interruption settings. This could occur when the value of an input property is specified incorrectly, or JP1/AO is unable to connect to the OS of the operation target device.

### Debugging with pauses between steps:

1. In the **Perform Debugging** dialog box, select **Interrupt After Each Step** in the **Interruption Settings** list box.
2. Specify the definition information for the debug service.
3. Click **Run**.  
The **Debug** view and the service template debugging view appear, and the debug task is executed.  
JP1/AO executes the debug task from the first step in the flow, which immediately pauses before executing its plug-in processing. For details about the flow of tasks in this scenario, see [5.3.9 Processing of debug process when pausing between steps](#).
4. In the **Debug** view and the **Properties** tab, check the mapping of the input properties, and change the input properties to arbitrary values as needed.
5. Select whether to execute plug-in processing. If you choose not to execute the processing of a particular plug-in, specify the output properties and return value of the plug-in. If you want to change the output properties and return value of a plug-in after executing its processing, configure JP1/AO to pause after processing the plug-in.
6. Click **Rerun** to resume step execution. If the plug-in requires a response, provide the response now.  
Plug-in processing is executed and execution of the step finishes.
7. If you configured JP1/AO to pause after executing the plug-in processing in step 5, change the output properties and return values as needed, and then click **Rerun**.
8. JP1/AO executes the subsequent step according to the subsequent-step execution condition, but pauses before executing the plug-in processing.
9. To execute the subsequent step, repeat steps 4 to 7.
10. After the debug task has finished, edit the service template or plug-ins as needed.



## Related topics

- 5.3.4 Procedure for starting the debug process
- 5.3.10 Plug-ins that cannot be paused during debugging
- 5.3.11 Step information that can be changed while step execution is paused
- 5.3.15 Handling debug tasks that are waiting for a response (response entry)
- 5.3.16 Procedure for checking property mapping settings during debugging
- 5.3.17 Procedure for changing the value of a plug-in input property during debugging
- 5.3.18 Procedure for changing the value of a plug-in output property during debugging
- 5.3.19 Procedure for changing plug-in return values during debugging

## 5.3.9 Processing of debug process when pausing between steps

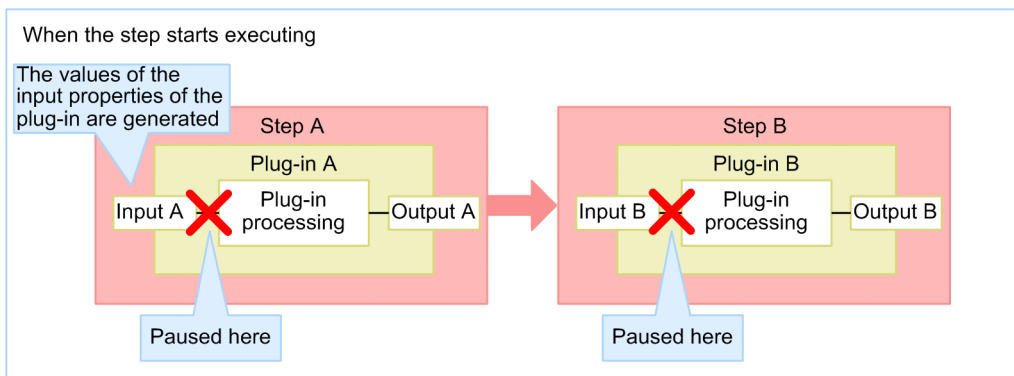
If you execute a debug task with **Interrupt After Each Step** specified as the break setting, the steps in the flow are paused before their plug-in processing is executed. The flow of processing when the step is resumed and the timing with which execution is paused depends on the debugging setting of the plug-in. This section describes the processing for the following scenarios:

- When executing a step
- When skipping plug-in processing
- When changing the output properties and return value of a plug-in

### Processing when executing a step

The following describes the processing of the debug task when a step is executed with **Interrupt After Each Step** specified as the break setting.

Figure 5–3: Flow of processing when executing a step



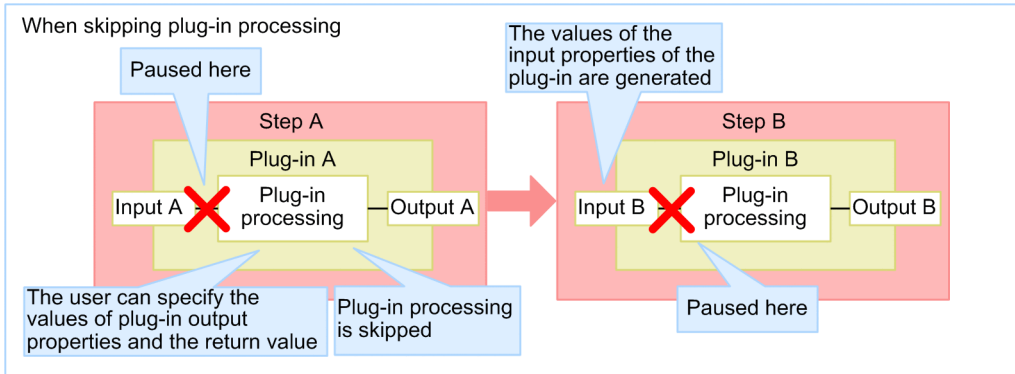
1. When you execute the debug task, JP1/AO begins executing Step A.
2. The value of the input property Input A of Plug-in A is generated, and Step A pauses before executing the processing of Plug-in A.
3. When you resume execution of Step A, JP1/AO executes the processing of Plug-in A.
4. JP1/AO begins executing Step B according to the subsequent-step execution condition.

The value of the input property Input B of Plug-in B is generated, and Step B is paused before executing the processing of Plug-in B.

### Flow of processing when skipping plug-in processing

The following describes the processing of the debug task when a step is executed with **Interrupt After Each Step** specified as the break setting, and **Do not execute the plug-in processing** is specified as the debugging setting.

Figure 5–4: Flow of processing when skipping plug-in processing

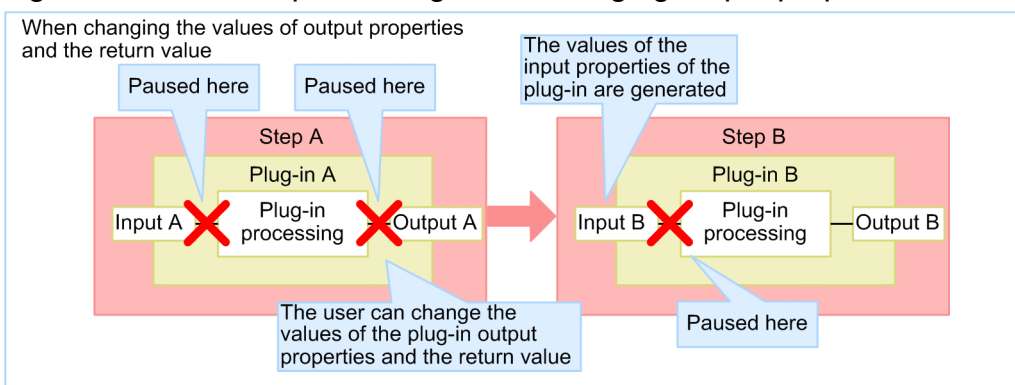


1. With Step A paused before executing the processing of Plug-in A, you select **Do not execute the plug-in processing** as the debugging setting for Plug-in A. You are then able to specify the values of output properties and the return value of the plug-in.
2. When you resume Step A, JP1/AO skips the execution of Plug-in A.
3. Step B begins executing, subject to the output properties and return value you specified. The value of the input property Input B of Plug-in B is generated. Step B is paused before executing the processing of Plug-in B.

### Flow of processing when changing output properties and return values

The following describes the processing of the debug task when **Interrupt After Each Step** is specified as the break setting, and **Interrupt after the plug-in processing** is specified as the debugging setting.

Figure 5–5: Flow of processing when changing output properties and return values



1. With Step A paused before executing the processing of Plug-in A, if you select **Interrupt after the plug-in processing** as the debugging setting for Plug-in A and resume Step A, JP1/AO executes the processing of Plug-in A. After processing Plug-in A, the execution of Step A pauses before the step finishes, allowing you to change output properties and return values.

2. When you resume Step A, Step B starts executing subject to the output property values and return value you specified. The value of the input property Input B of Plug-in B is generated. Step B is paused before executing the processing of Plug-in B.

## Related topics

- [5.3.8 Flow of debug process when pausing between steps](#)
- [5.3.11 Step information that can be changed while step execution is paused](#)
- [5.3.12 Procedure for skipping plug-in processing during the debug process](#)
- [5.3.17 Procedure for changing the value of a plug-in input property during debugging](#)
- [5.3.18 Procedure for changing the value of a plug-in output property during debugging](#)
- [5.3.19 Procedure for changing plug-in return values during debugging](#)

## 5.3.10 Plug-ins that cannot be paused during debugging

Some plug-ins cannot be paused when executing a step, even if **Interrupt After Each Step** is selected as the break setting for the debug task. When one of these plug-ins is encountered in the debug process, the flow automatically advances to the succeeding step.

Table 5–3: Ability to pause plug-ins

No.	Plug-in name	Can be paused	
1	Basic plug-ins	General command plug-in	Y
2		File-forwarding plug-in	Y
3		Repeated execution plug-in	Y
4		Email notification plug-in	Y
5		User-response wait plug-in	Y
6		Standard output plug-in	Y
7		Terminal connect plug-in	Y
8		Terminal command plug-in	Y
9		Terminal disconnect plug-in	Y
10		Flow plug-in	N
11		Interval plug-in	N
12		Judge returncode plug-in	N
13		Test value plug-in	Y
14		Abnormal-end plug-in	N
15		Judge value plug-in	N
16	Content plug-ins	Y	
17	Incompatible steps in service templates created in versions of JPI/AO earlier than 10-10	N	

Legend:

Y: Can be paused. N: Cannot be paused.

### 5.3.11 Step information that can be changed while step execution is paused

You can change the information of a step in a debug task while execution of the step is paused during the debug process. When there is an issue with a plug-in, this allows you to change the values of input properties and output properties and see how it affects plug-in processing.

The information you can change depends on when the step was paused.

Table 5–4: Step information that can be changed while a step is paused

Item	If the step was paused before executing plug-in processing		If the step was paused after executing plug-in processing
	When not skipping plug-in processing	When skipping plug-in processing	
Values of plug-in input properties	Y	N	N
Values of plug-in output properties	N	Y	Y
Return value of plug-in	N	Y	Y

Legend:

Y: Can be changed. N: Cannot be changed.

### 5.3.12 Procedure for skipping plug-in processing during the debug process

If there are plug-ins whose processing you do not want to execute during the debug process, you can skip the processing of those plug-ins. You can specify the values of output properties and the return value of these plug-ins so that it appears to the next step as if the plug-in has executed. This allows you to assess the effect the subsequent-step execution conditions have on the status transitions of steps and tasks, flow transitions, and the processing of subsequent steps.

For details on plug-ins whose processing cannot be skipped, see [5.3.10 Plug-ins that cannot be paused during debugging](#).

Note that you can only skip plug-in processing when **Interrupt After Each Step** is selected as the break setting for the debug process.

#### To skip plug-in processing:

1. With the step whose plug-in you do not want to execute paused, select **Do not execute the plug-in processing** from the **Execution Settings** list box in the **Debug** view.
2. In the **Return Value** text box, specify the value you want to use as the return value of the plug-in.  
The return value you specify determines the status transitions of steps and tasks and the flow transitions, subject to the subsequent-step execution conditions.
3. Specify values for output properties as needed.

#### Tip

If you have configured output property mapping, the value you specify for an output property is passed to the service property (output property or variable) to which it is mapped. If you do not specify a value, the service property (output property or variable) to which the output property is mapped will have a null value.

4. Click **Rerun**.

## Related topics

- [5.3.4 Procedure for starting the debug process](#)
- [5.3.9 Processing of debug process when pausing between steps](#)
- [5.3.18 Procedure for changing the value of a plug-in output property during debugging](#)
- [5.3.19 Procedure for changing plug-in return values during debugging](#)
- [3.2.4 Overview of subsequent step conditions](#)

### 5.3.13 Procedure for retrying a task from a failed step during debugging

When a debug task fails partway through, you can retry the task from the failed step.

By retrying from a failed step, you can resume the debug task with the same task ID and the original property values. You can use this approach when the cause of the failure has been resolved. For example, a step that fails due to a temporary problem with the network can be retried when the network connection is available again.

For details about retrying tasks, see *Retrying tasks in the Job Management Partner 1/Automatic Operation Administration Guide*. This describes situations in which property values are not inherited, whether tasks in a particular status can be retried, and other considerations.

#### To retry a task from a failed step during debugging:

1. With the debug task in Failed status, from the **Retry** list box in the **Debug** view, select **Retry the Task From the Failed Step**.

A dialog box appears in which you can confirm that you want to retry the task from the failed step.

2. Click the **OK** button.

## Related topics

- [5.3.14 Procedure for retrying a task from the step after the failed step during debugging](#)
- [Retrying tasks in the Job Management Partner 1/Automatic Operation Administration Guide](#)

### 5.3.14 Procedure for retrying a task from the step after the failed step during debugging

When a debug task fails partway through, you can retry the task from the step after the failed step.

By retrying from the step after the failed step, you can resume the debug task with the same task ID and the original property values. This approach is appropriate in situations where there is no need to execute the failed step. When you retry a task from the step after the failed step, processing of the task continues as if the failed step had ended normally. You can use this approach when you encounter an issue in a step, but want to continue executing the debug task and deal with the issue later.

For details about retrying tasks, see *Retrying tasks in the Job Management Partner 1/Automatic Operation Administration Guide*. This describes situations in which property values are not inherited, whether tasks in a particular status can be retried, and other aspects of retrying tasks.

### To retry a task from the step after a failed step during debugging:

1. With the debug task in Failed status, from the **Retry** list box in the **Debug** view, select **Retry the Task From the Step After the Failed Step**.  
A dialog box appears in which you can confirm that you want to retry the task from the step after the failed step.
2. Click the **OK** button.

### Related topics

- [5.3.13 Procedure for retrying a task from a failed step during debugging](#)
- [Retrying tasks in the \*Job Management Partner 1/Automatic Operation Administration Guide\*](#)

## 5.3.15 Handling debug tasks that are waiting for a response (response entry)

If a debug task requires a user response during execution, you can provide the response in the **Debug** view.

### To respond to a debug task that is waiting for a user response:

1. With the debug task in Waiting for Response status, click **Enter Response** in the **Debug** view.
2. Read the message in the **Respond** dialog box, and click the button associated with the action you want to perform.
3. In the **Information** dialog box, click the **OK** button.  
Depending on the response you entered, the plug-in processing of the step resumes or stops.

## 5.3.16 Procedure for checking property mapping settings during debugging

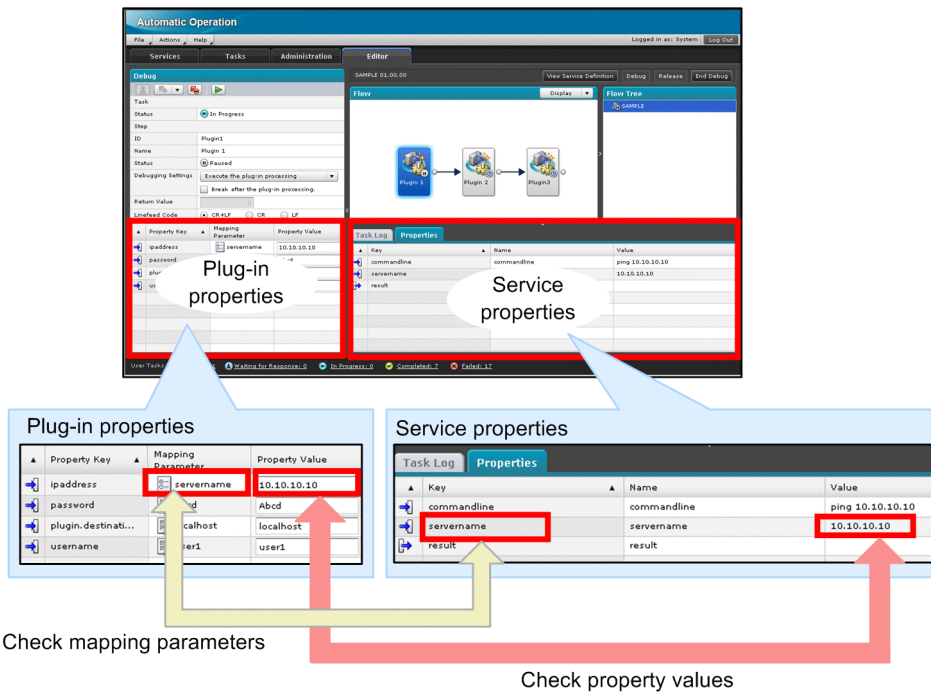
The following describes how to check whether the values of service properties and plug-in properties are mapped as intended during the debug process.

By viewing the **Properties** tab of the **Debug** view and the service template debugging view, you can make sure that the same value is assigned to plug-in properties and the service properties to which they are mapped.

Use the following timing to check the mapping settings during debugging:

- For mapping between plug-in input properties and service input properties or variables  
While the step is paused before executing the plug-in processing.
- For mapping between plug-in output properties and service output properties or variables
  - If skipping plug-in processing, when the step is resumed after you specify the plug-in output properties and return value.
  - After resuming a step that was paused after executing plug-in processing.

Figure 5–6: Window for checking property mapping



### To check the property mapping of a plug-in:

1. In the **Flow** view, select the step whose plug-in property values you want to check.  
The **Debug** view displays the input properties and output properties of the step you selected.
2. Click the **Properties** tab at the bottom of the service template debugging view.  
The values of the service properties are displayed.
3. In the **Debug** view, review the contents of the Mapping Parameters column for the plug-in property you want to check, and identify the service property to which it is mapped.
4. In the Property Key column of the **Properties** tab, find the service property you identified in step 3.
5. In the **Debug** view and the **Property List** view, make sure that the same value appears in the Property Value columns for the plug-in property and the mapped service property.

If a service property is not mapped to the intended plug-in property or the values of the plug-in property and the service property differ, fix the problem in the service template editing view.

You can also change the values of the plug-in properties. By doing so, you can test the plug-in processing when property mapping is configured correctly, and see how the processing of subsequent steps and the flow transitions change with an assortment of values.

### Tip

Property types are represented by the following icons in the **Debug** view and the **Properties** tab:



Indicates an input property.



icon

Indicates an output property.



icon

Indicates a variable.

### Important note

When `parallel` is specified for the `foreachMode` property of a repeated execution plug-in, values assigned to service properties in a repeated flow executed in parallel do not appear in the **Properties** tab.

### Related topics

- [5.3.9 Processing of debug process when pausing between steps](#)
- [5.3.17 Procedure for changing the value of a plug-in input property during debugging](#)
- [5.3.18 Procedure for changing the value of a plug-in output property during debugging](#)

## 5.3.17 Procedure for changing the value of a plug-in input property during debugging

You can change the value of plug-in input properties during debugging. By changing the input property to an arbitrary value before executing a plug-in, you can see how different property values affect the processing of the plug-in.

Note that you can only change the values of input properties if **Interrupt After Each Step** is selected as the break setting.

### To change the value of an input property during debugging:

1. In the **Flow** view, select a step that is in Interrupted status.
2. In the **Debug** view, select **Execute the plug-in processing** from the **Execution Settings** list box.
3. In the **Debug** view, specify the desired values for the input properties.
4. Click **Rerun**.

### Related topics

- [5.3.4 Procedure for starting the debug process](#)
- [5.3.8 Flow of debug process when pausing between steps](#)
- [5.3.9 Processing of debug process when pausing between steps](#)
- [5.3.16 Procedure for checking property mapping settings during debugging](#)
- [5.3.21 Effect of changing the values of plug-in properties during debugging](#)
- [5.3.22 When the value of a plug-in property includes surrogate pair characters or control characters](#)
- [5.3.23 Linefeed codes in values of plug-in properties \(when step execution is paused\)](#)



## 5.3.18 Procedure for changing the value of a plug-in output property during debugging

You can change the values of plug-in output properties during debugging. By changing an output property to an arbitrary value after executing or skipping a plug-in, you can see how different property values affect the processing of subsequent steps and the flow transitions.

Note that you can only change the values of output properties if **Interrupt After Each Step** is selected as the break setting.

The method of changing the output property value differs according to whether a step skips or executes plug-in processing.

### To change the value of an output property during debugging (when executing plug-in processing):

1. In the **Flow** view, select a step that is in Interrupted status.
2. In the **Debug** view, select **Execute the plug-in processing** from the **Execution Settings** list box.
3. Select the **Interrupt after the plug-in processing** check box.
4. Click **Rerun**.  
The step is paused after the plug-in processing is executed.
5. In the **Debug** view, specify the desired values for the output properties.
6. Click **Rerun**.

### To change the value of an output property during debugging (when skipping plug-in processing):

1. In the **Flow** view, select a step that is in **Interrupted** status.
2. In the **Debug** view, select **Do not execute the plug-in processing** from the **Execution Settings** list box.
3. In the **Debug** view, specify the desired values for the output properties.
4. Click **Rerun**.

### Related topics

- [5.3.4 Procedure for starting the debug process](#)
- [5.3.8 Flow of debug process when pausing between steps](#)
- [5.3.9 Processing of debug process when pausing between steps](#)
- [5.3.12 Procedure for skipping plug-in processing during the debug process](#)
- [5.3.16 Procedure for checking property mapping settings during debugging](#)
- [5.3.21 Effect of changing the values of plug-in properties during debugging](#)
- [5.3.22 When the value of a plug-in property includes surrogate pair characters or control characters](#)
- [5.3.23 Linefeed codes in values of plug-in properties \(when step execution is paused\)](#)

## 5.3.19 Procedure for changing plug-in return values during debugging

You can change the return value of a plug-in during debugging. By changing the return value to an arbitrary value after executing or skipping plug-in processing, you can see how different return values affect aspects of the task. This includes the status transitions of steps and tasks, processing of subsequent steps, and flow transitions influenced by the subsequent-step execution condition.

Note that you can only change return values if **Interrupt After Each Step** is selected as the break setting.

The method of changing the return value differs according to whether plug-in processing is executed or skipped.

### To change the return value during debugging (when executing plug-in processing):

1. In the **Flow** view, select a step that is in Interrupted status.
2. In the **Debug** view, select **Execute the plug-in processing** from the **Execution Settings** list box.
3. Select the **Interrupt after the plug-in processing** check box.
4. Click **Rerun**.  
The step is paused after the plug-in processing is executed.
5. In the **Debug** view, specify the desired return value in the **Return Value** text box.
6. Click **Rerun**.

### To change the return value during debugging (when skipping plug-in processing):

1. In the **Flow** view, select a step that is in Interrupted status.
2. In the **Debug** view, select **Do not execute the plug-in processing** from the **Execution Settings** list box.
3. In the **Debug** view, specify the desired return value in the **Return Value** text box.
4. Click **Rerun**.

### Related topics

- [5.3.4 Procedure for starting the debug process](#)
- [5.3.8 Flow of debug process when pausing between steps](#)
- [5.3.9 Processing of debug process when pausing between steps](#)
- [5.3.12 Procedure for skipping plug-in processing during the debug process](#)
- [3.2.4 Overview of subsequent step conditions](#)

## 5.3.20 Displaying the values of plug-in properties during debugging

The following describes the conditions under which the values of plug-in input and output properties appear in the user interface, and the location where these values appear.

Table 5–5: Conditions for displaying plug-in properties and their location

Type of plug-in property	Displayed plug-ins	Display timing	Display location
Input property	Plug-ins that can be paused, and judge value plug-ins	Timing with which plug-in execution begins	Property Value column in <b>Debug</b> view
Output property	Plug-ins that can be paused <sup>#</sup>	Timing with which plug-in processing ends	Property Value column in <b>Debug</b> view

#

If a plug-in returns a value of 65 or higher, the processing of the debug task will not pause after the plug-in, even if configured to do so in the interruption settings. This might occur when the value of an input property is specified incorrectly, or JP1/AO is unable to connect to the OS of the operation target device.

**Tip**

For plug-ins that cannot be paused (except judge value plug-ins), the values of input properties are those specified directly in the input property mapping settings when editing the step. Therefore, the input property values of these plug-ins appear in the Mapping Parameters column in the **Debug** view.

**Related topics**

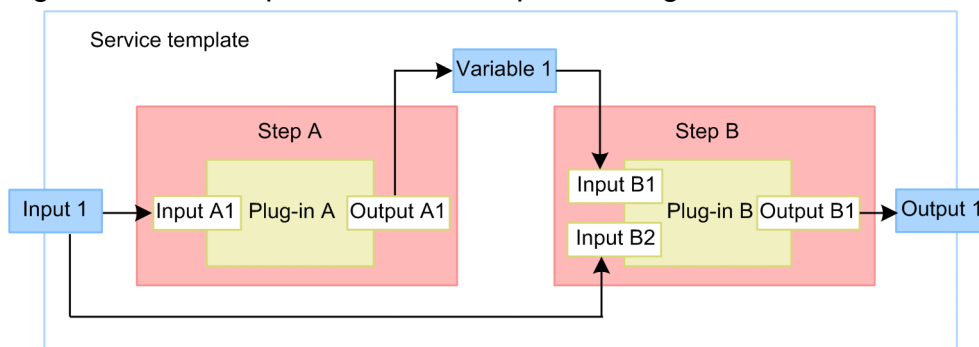
- [5.3.10 Plug-ins that cannot be paused during debugging](#)

### 5.3.21 Effect of changing the values of plug-in properties during debugging

During the debug process, the information you can change in relation to a step are the values of plug-in properties and the return values of plug-ins. Changing the value of a plug-in property does not automatically change the values of the service properties to which it is mapped. However, changing an output property of a plug-in can indirectly change the value of a service property (output property or variable) depending on how output property mapping is configured.

The following shows an example of how changing the values of input and output properties of a plug-in while executing a step affects the behavior of a service template. This example assumes a service template with the following definition:

Figure 5–7: Example of service template configuration



## Property mapping definition

### Step A

- The input property Input 1 of the service is mapped to the input property Input A1 of Plug-in A
- The output property Output A1 of Plug-in A is mapped to the variable Variable 1 of the service.

### Step B

- The variable Variable 1 of the service is mapped to the input property Input B1 of Plug-in B.
- The input property Input 1 of the service is mapped to the input property Input B2 of Plug-in B.
- The output property Output B1 of Plug-in B is mapped to the output property Output 1 of the service.

## Behavior when a property value is changed

### Behavior when changing an input property of a plug-in

If you change the value of the input property Input A1 of Plug-in A before executing the plug-in, Plug-in A uses the new value when it runs. This does not change the value of the service input property Input 1 to which the plug-in input property Input A1 is mapped. Therefore, the input property Input B2 of the plug-in is assigned the original value specified for the input property of the service.

### Behavior when changing an output property of a plug-in

If you change the value of the output property Output A1 after executing the plug-in, the new value is assigned to the service variable Variable 1 to which Output A1 is mapped. The input property Input B1 of the plug-in also takes the new value.

## 5.3.22 When the value of a plug-in property includes surrogate pair characters or control characters

If the value of a plug-in property contains surrogate pair characters or control characters (excluding line breaks and tab characters) at the point when a step is paused during debugging, these characters do not appear on screen. When you resume the step, a value with the surrogate pair characters and control characters (excluding line breaks and tab characters) omitted is set as the value of the plug-in property.

When the break setting of the debug task is Do Not Break, the surrogate pair characters and control characters (excluding line breaks and tab characters) in property values do not appear on screen. However, the value of the plug-in property retains these characters.

## 5.3.23 Linefeed codes in values of plug-in properties (when step execution is paused)

When a step is paused during debugging, you can change the linefeed code used in the values of the plug-in properties for that step. Specify the linefeed code used in the operating system of the connection destination device of the plug-in.

Service and plug-in properties handle CR+LF as two characters. As such, changing the linefeed code might cause the output of the plug-in to exceed the maximum number of characters (1,024 characters), altering the value that is ultimately assigned to the property. In this scenario, you can prevent the property value from changing by specifying the same linefeed code as the environment on the operation target device.

In the **Debug** view, you can select CR, LF, or CRLF as the linefeed code. When you resume the step, the linefeed codes in all plug-in properties in the step change to the linefeed code you selected. The default is the linefeed code in the existing property values. If the property values contain more than one linefeed code, the default is the first linefeed code specified when defining the plug-in properties. If the values of plug-in properties do not contain linefeed codes, the default is CRLF.

Note that you do not need to specify a linefeed code if no property values contain line feeds.

### 5.3.24 Displaying the flow of a debug task

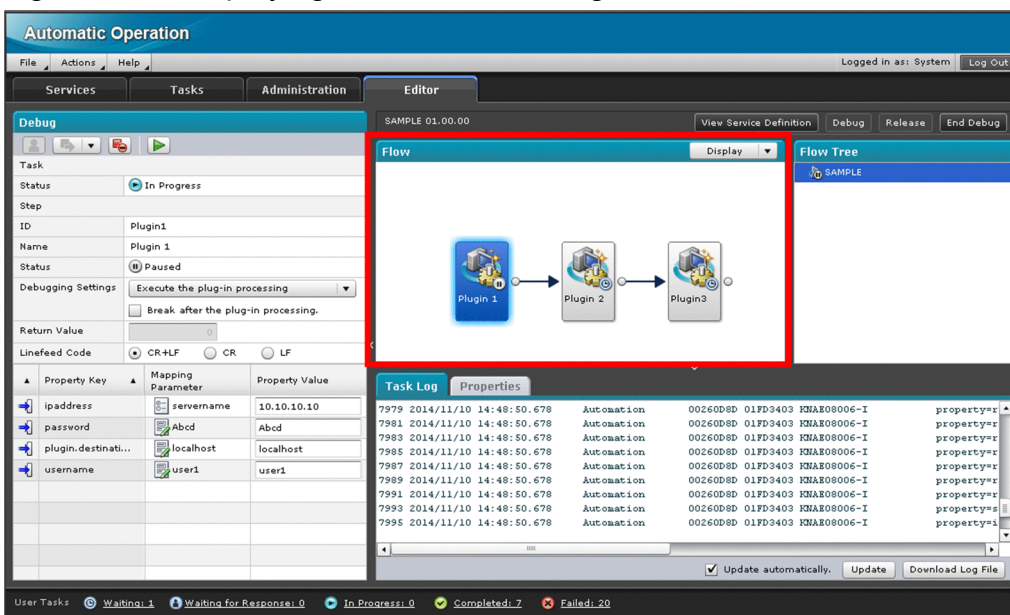
You can display a flow that represents the debug task you are executing.

The steps in the debug task appear in the **Flow** view of the service template debugging view, in the order in which they are executed. The icon of the step indicates the status of the step.

When you rest your mouse pointer on the step icon, the step name, the status icon, and the status of the step are displayed.

**Tip**  
For details about step statuses, see Step statuses in the *Job Management Partner 1/Automatic Operation Administration Guide*.

Figure 5–8: Displaying the flow of a debug task



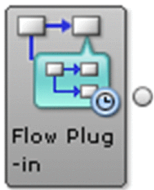
In the **Flow** view, the border around the icon of a paused step is highlighted as follows:

Figure 5–9: Icon of paused step (example)



Steps with plug-ins that cannot be paused have an icon with a darker background in the **Flow** view, as shown below. The icons of steps in a subordinate flow of a repeated execution plug-in also have a darker background until the repeated execution plug-in is executed.

Figure 5–10: Icon (example) of steps with unpausable plug-ins and steps in subordinate flows of repeated execution plug-ins (before execution)



### Related topics

- [5.3.26 Displaying a repeated execution flow during debugging](#)
- [5.3.28 Information displayed for repeated execution plug-ins and repeated flows during debugging](#)
- Step status icons in the *Job Management Partner 1/Automatic Operation GUI, Command, and API Reference*

## 5.3.25 Displaying the flow tree of a debug task

The name of the service template appears at the top level of the flow tree. Lower levels are represented by the name of the step that executes the flow plug-in or repeated execution plug-in.

During debugging, a pause icon is displayed for levels whose steps are paused (in a status requiring user intervention).

Figure 5–11: Example of pause icon (when a step in Flow plug-in 2 is paused)



You can find paused steps during debugging by looking for hierarchical levels with pause icons in the flow tree. By selecting a level with a pause icon, you can identify the paused step from the flow that appears.

The pause icon is not displayed for levels that are above the level containing the paused step in the hierarchy. The status of the step that represents the flow plug-in or repeated execution plug-in itself and the status of the repeated execution flow are not represented in the flow tree.

### Related topics

- [5.3.27 Flow tree view for repeated flows during debugging](#)

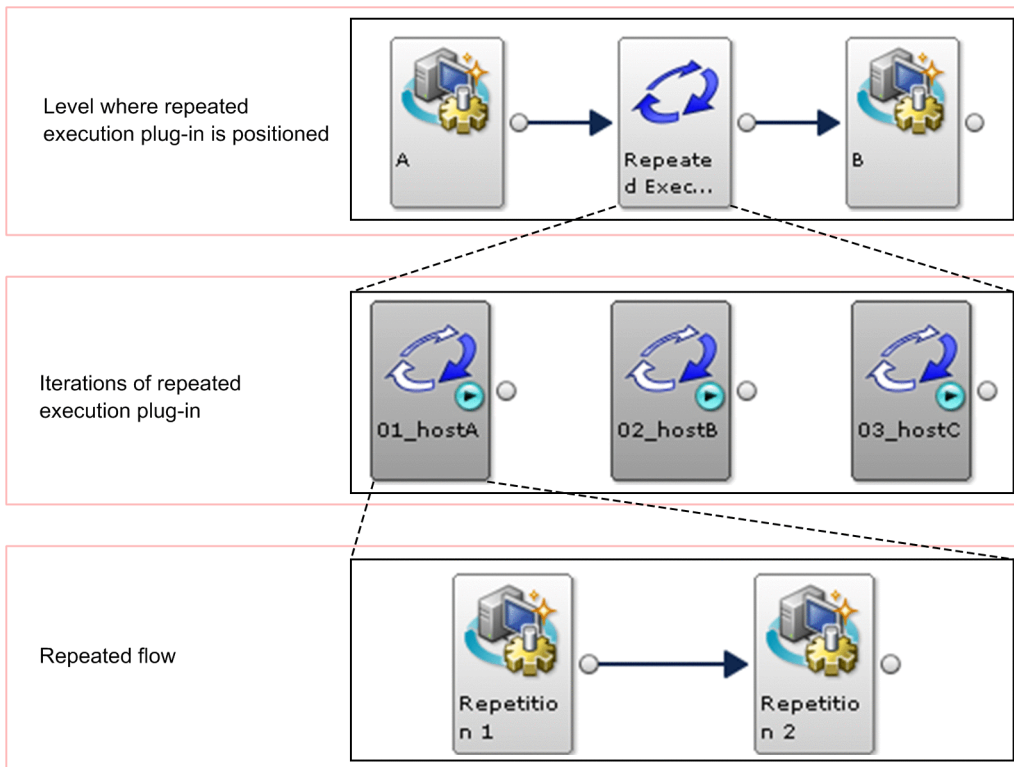
- [5.3.28 Information displayed for repeated execution plug-ins and repeated flows during debugging](#)

### 5.3.26 Displaying a repeated execution flow during debugging

During debugging, for steps that execute a repeated execution plug-in, the user interface displays each iteration of the flow in the levels below the repeated execution plug-in. This is called a repeated flow. A repeated flow appears as soon as a repeated execution plug-in is executed (during and after execution of the plug-in).

The following shows an example of a repeated execution plug-in executed with hostA,hostB,hostC specified in the Input Properties (inputProperties).

Figure 5–12: Flow displayed for repeated execution flow



The name of a repeated flow is displayed under the icon in the format *iteration-number\_input-value-(reserved.loop.input)*. Repeated flow names that are 65 characters or longer are truncated after the 64th character. If the Input Properties value contains control characters, names are truncated after the 64th character after removing the control characters. Note that *iteration-number* is a two-digit number. In the example in the figure above, the repeated flow names will be 01\_hostA, 02\_hostB, and 03\_hostC.

#### Related topics

- [5.3.26 Displaying a repeated execution flow during debugging](#)
- [5.3.28 Information displayed for repeated execution plug-ins and repeated flows during debugging](#)

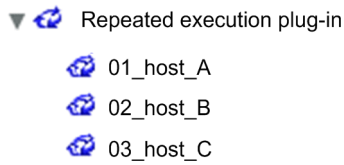
### 5.3.27 Flow tree view for repeated flows during debugging

The flow tree of the debug task you are debugging appears in the **Flow Tree** view of the service template debugging view. A repeated flow appears as soon as a repeated execution plug-in is executed (and remains after plug-in execution).

Repeated flow names appear in the format *iteration-number\_input-value-(reserved.loop.input)*. Repeated flow names that are 65 characters or longer are truncated after the 64th character. If the Input Properties value contains control characters, names are truncated after the 64th character after removing the control characters. Note that *iteration-number* is a two-digit number.

The following shows an example of the window contents when executing a repeated execution plug-in with hostA,hostB,hostC specified in the Input Properties (inputProperties).

Figure 5–13: Flow tree of repeated flow



Note that all iterations of the repeated flow appear in the window regardless of whether parallel or serial is specified for the foreachMode property.

If you log out during the debug process, information about iterations of the flow that are in Waiting to Repeat status might not appear in the flow tree.

### Related topics

- [5.3.25 Displaying the flow tree of a debug task](#)
- [5.3.28 Information displayed for repeated execution plug-ins and repeated flows during debugging](#)

## 5.3.28 Information displayed for repeated execution plug-ins and repeated flows during debugging

The table below describes the information displayed for repeated execution plug-ins and repeated flows when a step that includes a repeated execution plug-in is executed during a debug process. This information does not appear until the step has started executing.

Table 5–6: Information displayed for repeated flows

Item	Description
Execution method (parallel or serial)	The value of the foreachMode property of the repeated execution plug-in is displayed.
Iteration number (reserved.loop.index)	Combines with the applicable Input Properties value to give the name of the repeated flow. This information indicates which of the parameters in the comma-separated data specified in the inputProperties property of the plug-in applies to this instance of the flow.
Input Properties (reserved.loop.input)	Combines with the iteration number to give the name of the repeated flow. <sup>#</sup> This value is the comma-separated value in the inputProperties property of the repeated flow plug-in that corresponds to the particular iteration of the flow.
Repeated execution result (true or false)	This information appears as the value of the outputResult property of the repeated execution plug-in.
Output value (reserved.loop.output)	This information appears as the outputProperties output property of the repeated execution plug-in.



Item	Description
Output value (reserved.loop.output)	Its value is the value of a plug-in property mapped to reserved.loop.output by output property mapping, output during execution of the particular iteration of the repeated flow.

#

Repeated flow names that are 65 characters or longer are truncated after the 64th character. If the Input Properties value contains control characters, names are truncated after the 64th character after removing the control characters.

### Related topics

- [5.3.22 When the value of a plug-in property includes surrogate pair characters or control characters](#)

## 5.4 Managing debug tasks

---

### 5.4.1 Procedure for checking the status of all debug tasks

In the **Tasks** window, you can view the status of all debug tasks in a list. This feature is useful when you want to view the status of several debug tasks, or check the status of a debug task that was executed by another user.

#### To view the status of all debug tasks:

1. In the left pane of the **Tasks** window, click the **Debug Executed Tasks** menu command.
2. In the right pane (**Debug-Tasks** view) of the **Tasks** window, view the statuses of the debug tasks.

#### Tip

You can view detailed information about a task in the **Debug-Tasks** view by clicking the link in the **Task** column.

#### Important note

JP1/AO automatically deletes debug tasks (and archives tasks) whose retention period has expired and tasks exceeding the total number of tasks (including normal tasks) that can be retained. This takes place daily and is performed at the same time as the automatic archiving of normal tasks. Tasks are deleted or archived in order from the task with the oldest end date. You can change the retention period for debug tasks, the total number of tasks that can be retained, and the timing of automatic deletion in the property file (`config_user.properties`). For details about the automatic archiving of tasks, see *Archiving tasks and removing histories automatically* in the *Job Management Partner 1/Automatic Operation Administration Guide*.

#### Related topics

- Property file (`config_user.properties`) in the *Job Management Partner 1/Automatic Operation Configuration Guide*

### 5.4.2 Procedure for checking the progress of debug tasks from the Tasks window

From the **Tasks** window, you can view the progress of debug tasks as a flow. You might use this approach when you want to check the status of steps in a debug task after closing the **Debug** view and the service template debugging view. You could also use this procedure to check the status of a step in a debug task executed by another user.

#### To view the status of steps in a debug task from the Tasks window:

1. In the left pane of the **Tasks** window, click the **Debug Executed Tasks** menu command.
2. In the right pane (**Debug-Tasks** view) of the **Tasks** window, select the task whose step status you want to check.
3. In the **Task Monitor** view, check the progress of the task. The progress of a task is indicated by an icon superimposed on the step icon.

## Related topics

- [5.3.25 Displaying the flow tree of a debug task](#)
- [5.3.26 Displaying a repeated execution flow during debugging](#)
- [Checking the progress of tasks in flow format in the \*Job Management Partner 1/Automatic Operation Administration Guide\*](#)
- [Step statuses in the \*Job Management Partner 1/Automatic Operation Administration Guide\*](#)
- [Step status icons in the \*Job Management Partner 1/Automatic Operation GUI, Command, and API Reference\*](#)

### 5.4.3 Procedure for checking detailed progress of debug tasks in list format

The following describes how to view detailed information about the progress of a debug task. You can view the progress of a debug task from a general perspective, or check the progress of individual steps. Note that only the steps defined at the highest level of the flow appear in the list. Steps in flows at lower hierarchical levels and steps in flow plug-ins and repeated execution plug-ins do not appear.

#### To display detailed progress information for debug tasks in list format:

1. In the left pane of the **Tasks** window, click the **Debug Executed Tasks** menu command.
2. In the right pane (**Tasks List** view) of the **Tasks** window, click the *task-name*.

#### Tip

You can achieve the same result by selecting the debug task whose status you want to check in the table, and then clicking **Show Task Details**.

3. Click **Step Details** at the bottom of the **Task Details** dialog box.
4. View the overall progress of the debug task in **Step Information**.
5. View the progress of individual steps in the **Steps** area.

### 5.4.4 Procedure for checking task log entries for debug tasks

The following describes how to view the information output to the task log by a debug task. You cannot view the task log for a debug task that is in *Waiting* status. You can also download the task log to a folder of your choice, under any file name.

You can view the task log for a running debug task from the **Editor** window.

To view the task log for a finished debug task or a debug task that was executed by another user, use the **Tasks** window.

For details about the information output to the task log, see *Task log details* in the *Job Management Partner 1/Automatic Operation Administration Guide*.

#### To view the task log for a debug task from the Editor window:

1. In the service template debugging view, click the **Task Log** tab.

The **Task Log** tab shows the contents of the task log of the debug task you are debugging.

In the **Task Log** tab, you can perform actions such as setting the timing with which the contents of the task log are refreshed, and downloading the task log information to a file. The following describes how to perform these actions.

- To automatically update the task log at regular intervals:

Select the **Update automatically** check box.

When you select this check box, the task log is automatically updated each time the status of a step or task changes while the debug task is running.

- To manually update the task log:

Click **Update**.

- To download the task log:

Click **Download Log File**.

### Tip

- When you debug a debug task that has already undergone a debug process, the **Update automatically** check box is selected automatically and the task log is refreshed. This also happens when you retry a debug task.
- When you select the **Update automatically** check box, regardless of the task status, the task log is refreshed and the scroll box in the **Task Log** tab scrolls to the bottom of the table.

### To view the task log for a debug task from the Tasks window:

1. In the left pane of the **Tasks** window, click the **Debug Executed Tasks** menu command.

2. View the task log in the right pane (**Debug-Tasks** view) of the **Tasks** window.

For details about how to view the task log from the **Tasks** window, see *Displaying task logs in the Job Management Partner 1/Automatic Operation Administration Guide*.

### Important note

- The task log size specified (in KB) in the property file (`config_user.properties`) determines the amount of information displayed in the **Task Log** dialog box and in the **Task Log** tab of the service template debugging view. This amount of information is taken from the end of the task log data. The dialog boxes do not display the entire contents of the task log.
- You can set the maximum log file size for debug tasks (in KB) in the property file (`config_user.properties`). When the maximum file size is exceeded, JP1/AO begins overwriting the oldest information in the task log.

### Related topics

- Property file (`config_user.properties`) in the *Job Management Partner 1/Automatic Operation Configuration Guide*

## 5.4.5 Procedure for stopping debug tasks

The following describes how to stop a debug task that is still in progress. You can stop a debug task that is in In Progress, Waiting for Response, or Abnormal Detection status. Stopped tasks enter Failed status, unless the final step was already in progress in which case the debug task enters Completed status if the final step ends normally.

If the debug task contains a paused step, the step is automatically resumed and the task ends when the step finishes executing.

### Tip

For details about the difference between stopping and forcibly stopping tasks, see *Managing Tasks* in the *Job Management Partner 1/Automatic Operation Administration Guide*.

#### To stop a debug task:

1. In the left pane of the **Tasks** window, click the **Debug Executed Tasks** menu command.
2. In the right pane (**Debug-Tasks** view) of the **Tasks** window, select the debug tasks that you want to stop. You can select multiple debug tasks by holding the **Ctrl** key as you click. To select all the debug tasks in the list, click **Select All**.

### Important note

Clicking a *task-name* link displays the **Task Details** dialog box for that task. To prevent this from happening, select each debug task by clicking the empty space in the row for the task.

3. In the **More Actions** menu, click **Stop Tasks**.
4. In the **Stop Tasks** dialog box, review the debug task or tasks you are stopping, and then click **OK**.
5. In the **Information** dialog box, click **OK**.

The selected debug task or tasks enter Failed status.

## 5.4.6 Procedure for forcibly stopping debug tasks

The following describes how to forcibly stop a debug task that is still in progress. You can forcibly stop a debug task that is in In Progress, Waiting for Response, Abnormal Detection, or Terminated status.

When you want to stop a running debug task and perform the debug process again, you can forcibly terminate the debug task from the **Editor** window.

You can forcibly stop a task from the **Tasks** window. Situations in which you might need to forcibly stop a task include inadvertently closing the Web browser from which you are using JP1/AO while a debug task is in progress, or stopping a debug task executed by another user.

Forcibly stopped debug tasks enter Failed status.

If the debug task contains a paused step, the debug task stops after the step is automatically resumed, without executing the plug-in processing.

### Tip

For details about the difference between stopping and forcibly stopping tasks, see *Managing Tasks* in the *Job Management Partner 1/Automatic Operation Administration Guide*.

### To forcibly stop debug tasks from the Editor window:

1. In the **Debug** view of the **Editor** window, click **Forcibly Stop Tasks**.
2. In the **Forcibly Stop Tasks** dialog box, review the debug task you are forcibly stopping, and then click **OK**.
3. In the **Information** dialog box, click **OK**.

The debug task enters Failed status.

### To forcibly stop debug tasks from the Tasks window:

1. In the left pane of the **Tasks** window, click the **Debug Executed Tasks** menu command.
2. In the right pane (**Debug-Tasks** view) of the **Tasks** window, select the debug tasks you want to forcibly stop. You can select multiple debug tasks by holding the **Ctrl** key as you click. To select all the debug tasks in the list, click **Select All**.

#### Important note

Clicking a *task-name* link displays the **Task Details** dialog box for that task. To prevent this from happening, select each debug task by clicking the empty space in the row for the task.

3. In the **More Actions** menu, click **Forcibly Stop Tasks**.
4. In the **Forcibly Stop Tasks** dialog box, review the debug tasks you are forcibly stopping, and then click **OK**.
5. In the **Information** dialog box, click **OK**.

The selected debug tasks enter Failed status.

## 5.4.7 Procedure for deleting debug tasks

The following describes how to manually delete debug tasks that are no longer required. You can delete debug tasks that are in Completed or Failed status.

If you want to delete debug tasks automatically, set a retention period for finished debug tasks in the property file (config\_user.properties).

### To manually delete debug tasks:

1. In the left pane of the **Tasks** window, click the **Debug Executed Tasks** menu command.
2. In the right pane (**Debug-Tasks** view) of the **Tasks** window, select the debug tasks you want to delete. You can select multiple debug tasks by holding the **Ctrl** key as you click. To select all the debug tasks in the list, click **Select All**.

#### Important note

Clicking a *task-name* link displays the **Task Details** dialog box for that task. To prevent this from happening, select each debug task by clicking the empty space in the row for the task.

3. In the **More Actions** menu, click **Delete Tasks**.

4. In the **Delete Tasks** dialog box, review the debug tasks you are deleting and then click **OK**.

5. In the **Information** dialog box, click **OK**.

The debug tasks you selected are removed from the **Debug-Tasks** view.

### **Related topics**

- Property file (config\_user.properties) in the *Job Management Partner 1/Automatic Operation Configuration Guide*

## 5.5 Testing service templates

---

### 5.5.1 Procedure for testing service templates

When you have finished debugging the service template and are sure that no issues remain, you can perform an operation test in the development environment.

#### To test the operation of a service template:

1. Build the service template and then add it as a service.  
For details about how to add a service, see *Adding services in the Job Management Partner 1/Automatic Operation Administration Guide*.
2. Execute the service.  
For details about how to execute a service, see *Submitting services for execution in the Job Management Partner 1/Automatic Operation Administration Guide*.
3. Check the execution results in the **Tasks** window.  
For details about how to check the execution results, see *Managing Tasks in the Job Management Partner 1/Automatic Operation Administration Guide*.
4. If the results of the operation test reveal an issue with the service template, edit the affected service template or plugin.

#### Related topics

- [5.1.1 Flow of service template validation](#)
- [5.1.2 Overview of building](#)
- [5.1.4 Overview of operation tests](#)
- [5.2.1 Procedure for building a service template](#)



# 6

## Releasing Service Templates

This chapter describes how to release service templates. Releasing an edited service template creates a package that is imported to the JP1/AO server.

## 6.1 Releasing service templates

---

### 6.1.1 Procedure for releasing a service template

When you select and release a service template, a package is created based on the service template and imported to the JP1/AO server. If the development environment is the same as the active environment, you are then able to add the service.

#### Important note

Once a service template is released, it cannot be re-edited. If you want to edit a Release service template, copy it so that it can be edited.

#### If the service template editing view is not displayed:

1. In the **Life Cycle of a Service Template** area of the **Service Template Editor Home** view, click **Edit**.
2. In the **Service Template List** window, select a service template and then click **Edit**.

#### To release a service template:

1. In the service template editing view, click **Release**.  
You can also release a service template by clicking the **Release** button in the service template debugging view.
2. In the confirmation window, click **OK**.
3. Check the results in the **Build / Release Result** dialog box, and then click **Close**.
4. In the **Information** dialog box, click **OK**.

If the service template has been deleted or released by another user, an error occurs and the release process fails.

If an error occurs during the release process, a message is displayed to the operator that describes the cause of the error and instructs the operator to fix the flow. The error message remains on screen until the operator closes the service template editing view.

#### Important note

When you release a service template from the service template debugging view, the content of the released service template is that of the last build you performed as part of the debug process. If another user edits and builds the service template in the meantime, the changes made by this user do not apply to this release.

#### Related topics

- [6.2.1 Overview of service template release](#)
- [A.1\(5\) Deletion and archiving of service templates, services, and tasks during build and release operations](#)

## 6.2 Overview of service template release

---

### 6.2.1 Overview of service template release

Release is the process of making a service template available to other users after it has undergone the validation process. When successfully released, a service template is packaged and can be added as a service.

Note that you cannot edit a service template or its plug-ins after the service template has been released. To edit a released service template or its plug-ins, you need to copy the service template or plug-in and edit the copy.

#### Objective

Perform a release operation when you want to use a service template in the active environment. A released service template is packaged as a service template of the Release configuration type, and imported to the JP1/AO server.

#### Number of releases

You can release a service template only once. When you release a service template, the pre-release development service template is deleted. Any services that were added from the development service template prior to release are deleted, and associated tasks are archived. Debug services and debug tasks are deleted.

For details about the specific service templates, services, tasks, debug services, and debug tasks that are deleted and archived during a release operation, see [A.1\(5\) Deletion and archiving of service templates, services, and tasks during build and release operations](#).

#### Assigned configuration type

After its release, a service template is assigned the Release configuration type. In addition to users in the Admin and Develop roles, users in the Modify and Submit roles can configure and execute release service templates and the associated services and tasks. Note that you cannot edit a service template or its plug-ins after the service template has been released. To edit a released service template or its plug-ins, you need to copy the service template or plug-in and edit the copy. For details about the configuration types assigned to service templates and plug-ins by the release process, see [A.1\(2\) Configuration types assigned to service templates and plug-ins by build and release operations](#).

#### Output destinations of service templates

When you release a service template, a service template package is created in the following folder with the name *vendor-ID\_name\_version.st*:

In a non-cluster system

*JP1/AO-installation-folder*\develop\output

In a cluster system

*shared-folder-name*\develop\output

#### Related topics

- [2.1.2 Overview of development service templates and release service templates](#)
- [2.4.1 Procedure for copying service templates](#)
- [A.1\(1\) Structure of build and release processes and how they differ](#)
- [A.1\(2\) Configuration types assigned to service templates and plug-ins by build and release operations](#)
- [A.1\(5\) Deletion and archiving of service templates, services, and tasks during build and release operations](#)

## 6.3 Importing service templates into the active environment (when the active and development environments are separate)

---

### 6.3.1 Procedure for importing service templates into the active environment (when the active and development environments are separate)

If your JP1/AO system maintains separate development and active environments, service templates created in the development environment must be imported into the active environment.

When the development and active environments are separate, we recommend that you check the operation of the service template in the development environment before proceeding. After checking operation in the development environment, apply the service template to the active environment. The following describes how to bring a service template from the development environment to the active environment.

#### To bring a service template from the development environment to the active environment:

1. Copy the service template package from the JP1/AO server in the development environment to the local disk of the JP1/AO server in the active environment.
2. On the JP1/AO server in the active environment, execute the `importservicetemplate` command to import the service template package.
3. Create the service on the JP1/AO server in the active environment.

#### Tip

You can import several service template packages in a single operation by archiving the packages as a zip file and using the `importservicetemplate` command to import the file.

#### Related topics

- [6.3.2 Reason for maintaining separate development and active environments](#)

### 6.3.2 Reason for maintaining separate development and active environments

When you develop a service template in a separate environment from the active environment, each environment has its own set of service share properties.

When the development and active environments are the same

A given service share property has one value within the JP1/AO system. When you change the value of a service share property in the service template editor, the change affects release service templates in addition to development service templates.

When the development and active environments are different

The values of service share properties are managed separately in the active and development environments. When you change the value of a service share property in the service template editor, the change only applies to the service

share property associated with the development service template. It does not affect the service share property associated with the release service template.

Therefore, we recommend that you keep the development and active environments of the service template separate. This allows you to prevent changes made during development from affecting the service share properties of the service template after release.

# Appendix

## A. Reference Information

---

This appendix provides reference information for users of JP1/AO.

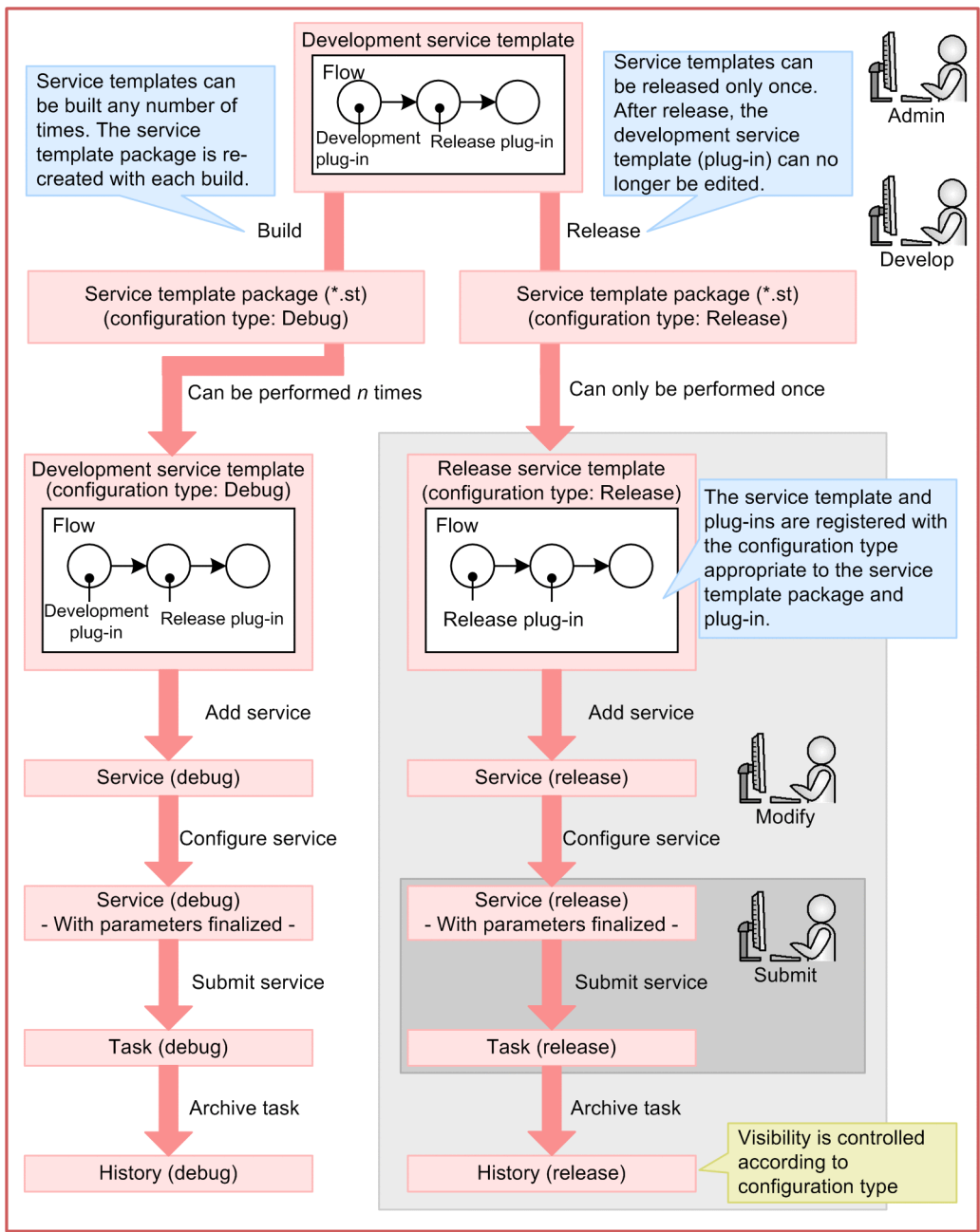
### A.1 Reference information for build and release operations

#### (1) Structure of build and release processes and how they differ

The build and release processes create a package of a service template. Perform a build operation when you want to validate a service template, and a release operation when you want to make the service template available in the active environment.

The following figure shows the structure of the build and release processes.

Figure A-1: Structure of build and release processes



The following table lists the differences between build and release processes.

Table A-1: Differences between build and release

Operation	Objective	Number of executions	Configuration type assigned to service template package	File name of service template package	Can development service template be edited after operation?
Build	To validate a service template	Any number of times	Debug configuration	<i>vendor-ID_name_version_d.st</i>	Yes
Release	To make a service template available in the active environment	Once	Release configuration	<i>vendor-ID_name_version.st</i>	No



Legend:

Yes: Can be edited. No: Cannot be edited.

## Related topics

- [A.1\(2\) Configuration types assigned to service templates and plug-ins by build and release operations](#)
- [A.1\(4\) Ability to display window items by service template configuration type and user role](#)
- [A.1\(5\) Deletion and archiving of service templates, services, and tasks during build and release operations](#)

## (2) Configuration types assigned to service templates and plug-ins by build and release operations

JP1/AO assigns a configuration type to the service templates and plug-ins in a service template package (\*.st) according to its stage in the development process (build or release). The following table lists the configuration types assigned to service templates and plug-ins.

Table A–2: Configuration types assigned by build and release operations

Service template or plug-in		Assigned configuration type	
		When built	When released
Development service template		Debug configuration	Release configuration
Plug-in <sup>#</sup>	Basic plug-in or release plug-in (release configuration)	Release configuration	Release configuration
	Development plug-in (debug configuration)	Debug configuration	Release configuration

#

You cannot build or release individual plug-ins. A development plug-in becomes a release plug-in when you release a service template that contains that plug-in. A development plug-in remains as such when you build a service template that contains the plug-in.

## Related topics

- [A.1\(4\) Ability to display window items by service template configuration type and user role](#)

## (3) Behavior when multiple build or release operations apply to the same service template or plug-in

If you repeatedly build a service template during its development, or you build then later release a service template, the same service template or plug-ins might have already been imported. The behavior of the build or release operation in this scenario depends on the configuration type of the service template or plug-in.

Here, the same service template (or same plug-in) means a service template or plug-in with the same vendor ID, service template ID (plug-in ID), and service template version (plug-in version). The plug-ins referred to below are those used by the service template being built or released.

### When building a service template

- The service template is overwritten.
- Development plug-ins in the service template you are building are overwritten.
- Release plug-ins in the service template you are building are not overwritten.

## When releasing a service template

- The service template is overwritten and becomes a release service template.
- Development plug-ins in the released service template become release plug-ins.
- Release plug-ins in the released service template are not overwritten.

## (4) Ability to display window items by service template configuration type and user role

The configuration type of the service template and the role of the user determine whether service templates and elements created from service templates (services, tasks, and task histories) appear in the **Services** window and the **Tasks** window. The following table describes whether these items appear in the user interface for each combination of configuration type and user role.

Table A–3: Ability to display elements by service template configuration type and user role

Service template configuration type	Managed element	Displayed in the <b>Services</b> window and <b>Tasks</b> window		
		Admin Develop	Modify	Submit
Debug configuration	Service template	Y	N	N
	Service	Y	N	N
	Task	Y	N	N
	History	Y	N	N
Release configuration	Service template	Y	Y	N
	Service	Y	Y	Y
	Task	Y	Y	Y
	History	Y	Y	VO

Legend:

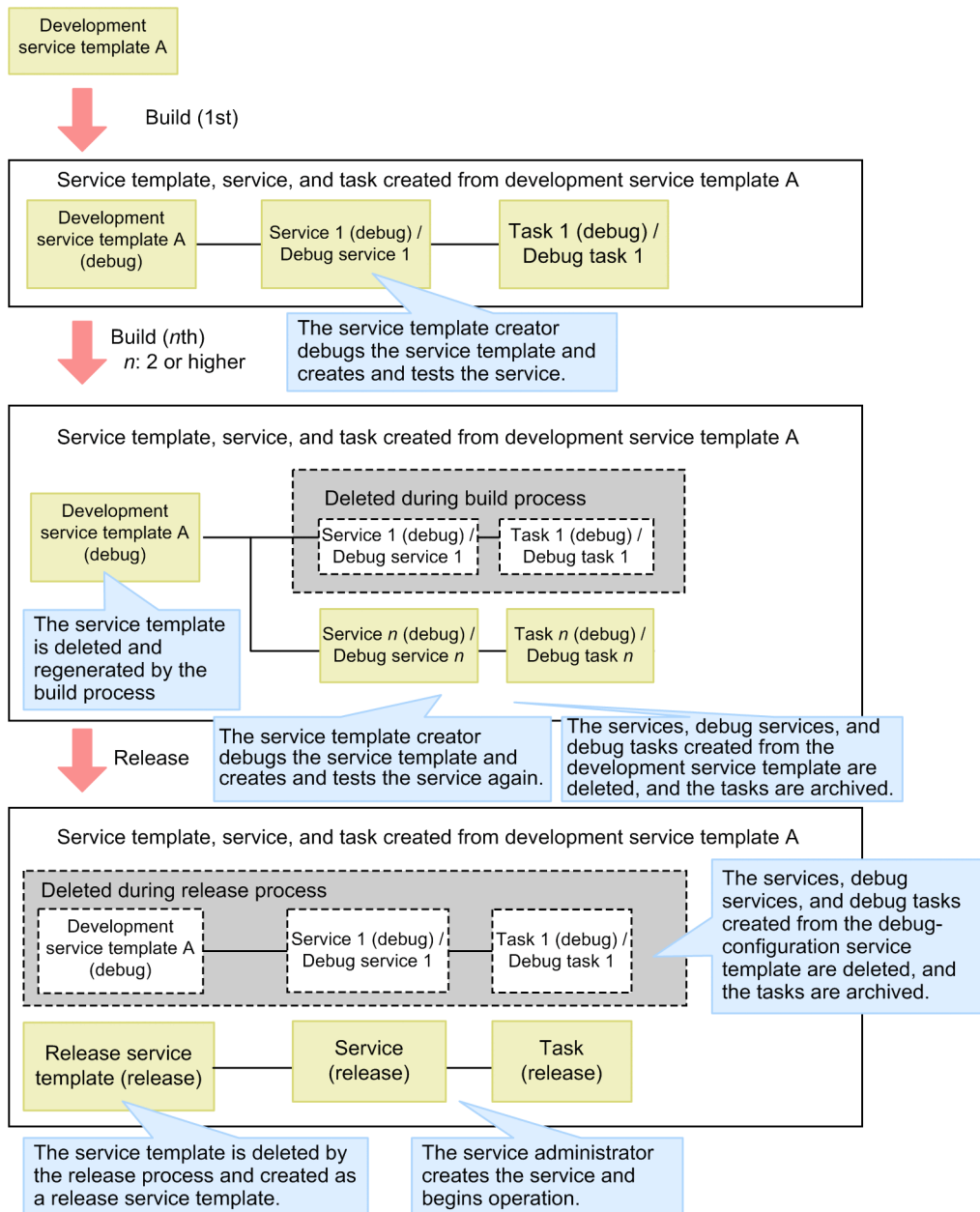
Y: Can be viewed and manipulated. VO: Can only be viewed. N: Cannot be viewed or manipulated.

## (5) Deletion and archiving of service templates, services, and tasks during build and release operations

When you build or release a development service template, JPI/AO automatically deletes service templates and services and archives tasks. Debug services and debug tasks are automatically deleted.

Note that service templates and services are deleted and tasks archived even if the build or release process fails. Debug services and debug tasks are also deleted.

Figure A–2: Deletion and archiving of elements during build and release operations



# Index

## A

- ability to display elements by service template configuration type and user role 170
- active environments
  - reason for maintaining separate from development environment 164

## B

- basic plug-in 93
- basic plug-ins 93
- building service templates 128
- build or release operations performed on multiple instances of same service template or plug-in 169

## C

- cautionary notes
  - Service Share Properties 59
- character set
  - setting specific character set at plug-in execution 99
  - used during plug-in execution 98
- commands
  - required for plug-in execution 99
  - setting procedure 115
  - specifying in Command-line text box 109
- compatibility with steps
  - created in earlier versions 48
- configuration types assigned to service templates and plug-ins by build and release operations 169
- connecting steps
  - relational lines 86
- content plug-ins 93
- copying
  - service templates 66
- creating
  - flow hierarchy 77
- credential types
  - plug-in 103
- custom file
  - format 56
  - setting 54
- custom files
  - overview 55

## D

- debugging service templates 129
- debugging service templates again without rebuilding 134
- debugging without pausing between steps 135
- debugging with pausing between steps 136
- definitions
  - service resource files 70
- deleting
  - service template 68
- deletion and archiving of service templates, tasks, and services during build and release operations 170
- development environments
  - reason for maintaining separate from active environment 164
- development plug-ins 93
- development service template 20, 46
- development service templates
  - overview 46
- displaying
  - Editor window 45
- displaying a plug-in on a Web browser set to a locale for which no plug-in resource file is available 121
- displaying debug task flow 149
- displaying flow tree of debug task 150
- displaying list
  - plug-ins 93
  - service templates 65
- displaying repeated execution flow during debugging 151
- displaying values of plug-in properties during debugging 146
- display procedure
  - Editor window 45
  - Flow view 76

## E

- editing existing service template 30
- editing plug-ins to apply to service templates
  - general procedure 32
- Editor window
  - behavior when intervening actions occur 47
  - displaying 45
  - display procedure 45

version compatibility for service templates and plug-ins 48

effect of changing values of plug-in properties during debugging 147

example of service template debugging 131

executing users

plug-in 94

Execution Directory

specifying 114

## F

file transfer

files transferred to UNIX systems 97

files transferred to Windows systems 96

flow 20

creating 75

editing 75

relationship to steps 76

flow hierarchy 77

flow of service template debugging 129

flow of service template development 17

flow of service template validation 123

flow tree view for repeated flows during debugging 151

Flow view

displaying 76

display procedure 76

format

service resource file 69

functions used during debug operations 129

## G

general procedure

creating new service templates 28

## H

handling debug tasks that are waiting for response (response entry) 142

## I

importing into active environment

service templates 164

information displayed for repeated execution plug-ins and repeated flows during debugging 152

input property mapping

setting procedure 81

## L

linefeed codes in values of plug-in properties (when step execution is paused) 148

## M

managing debug tasks 154

mapping parameter

definition example 52

flow of data 52

## O

output property mapping

setting procedure 82

overview

service template development 17

subsequent step conditions 80

overview of building 124

overview of debugging 125

overview of operation tests 126

overview of service template validation 123

## P

parameters

setting dynamically or statically 38

parameters to set

plug-in definition information 102

plug-in

available operations by plug-in type 96

credential types 103

executing users 94

image files usable as icons 103

setting display information 118

using plug-in resource files to set display information 40

plug-in definition information

parameters to set 102

plug-in execution

locale settings for operation target devices 97

plug-in properties 104

plug-in resource file

correspondence between properties and displayed information 119

plug-in resource files

format 118

procedure for setting 118

- plug-in resource files automatically generated when a plug-in is created [120](#)
- plug-in resource files updated when a plug-in is edited [121](#)
- plug-ins
  - copying [117](#)
  - copying procedure [117](#)
  - creating [92, 100](#)
  - creating and adding to service templates [33](#)
  - creation procedure [100](#)
  - deleting [116](#)
  - deletion procedure [116](#)
  - displaying list [93](#)
  - editing [92, 102](#)
  - parameters to set when creating or copying [100](#)
  - procedure for editing definition information [102](#)
  - procedure for listing [93](#)
  - uniqueness [66](#)
- plug-ins definition information
  - procedure for editing [102](#)
- Plug-ins that cannot be paused during debugging [139](#)
- procedure
  - copying plug-ins [117](#)
  - deletion plug-ins [116](#)
  - setting commands [115](#)
  - setting plug-in resource files [118](#)
- procedure for building service template [128](#)
- procedure for changing plug-in return values during debugging [146](#)
- procedure for changing value of plug-in input property during debugging [144](#)
- procedure for changing value of plug-in output property during debugging [145](#)
- procedure for checking detailed progress of debug tasks in list format [155](#)
- procedure for checking progress of debug tasks from Tasks window [154](#)
- procedure for checking property mapping settings during debugging [142](#)
- procedure for checking status of all debug tasks [154](#)
- procedure for checking task log entries for debug tasks [155](#)
- procedure for deleting debug tasks [158](#)
- procedure for forcibly stopping debug tasks [157](#)
- procedure for retrying from failed step during debugging [141](#)
- procedure for retrying task from step after failed step during debugging [141](#)

- procedure for skipping plug-in processing during debugging [140](#)
- procedure for starting debug [132](#)
- procedure for stopping debug tasks [156](#)
- procedure for testing service templates [160](#)
- processing of debug process when pausing between steps [137](#)
- properties
  - defined in plug-ins [104](#)
  - defined in services [57](#)
- property
  - visibility [62](#)

## R

- reference information [167](#)
- reference information for build and release [167](#)
- relational line
  - procedure for deleting [86](#)
- relational lines
  - behavior when connected to multiple steps [89](#)
  - connecting steps [86](#)
  - information inherited when pasting [88](#)
  - plug-ins when processing branches [91](#)
  - procedure for connecting steps [86](#)
  - procedure for copying [87](#)
  - procedure for cutting [87](#)
  - procedure for pasting [88](#)
  - restrictions [90](#)
- relationship of command and script return values to return codes of plug-ins and steps [112](#)
- release
  - overview [163](#)
- release plug-ins [93](#)
- release service template [20, 46](#)
- release service templates
  - deleting procedure [68](#)
  - overview [46](#)
- releasing
  - service templates [162](#)
- reserved plug-in properties
  - credential information [104](#)
  - specifying execution-target hosts [104](#)
- reserved property
  - list [82](#)
- resource files
  - setting service template display information [69](#)
- return values of content plug-ins [110](#)

## S

- script
  - differences between setting methods 113
- scripts
  - setting procedure 107
- selecting multiple steps
  - procedure 88
- service definition information
  - parameters to set 51
  - setting procedure 51
- service properties 57
- service resource file
  - format 69
  - setting displayed information 39
- service resource files
  - automatically generated at service template creation 72
  - correspondence between properties and displayed information 71
  - definitions 70
  - displaying service template in Web browser whose locale has no service resource file 74
  - setting procedure 69
  - updated when service template is edited 73
- services
  - changing description 35
- Service Share Properties
  - cautionary notes 59
  - overview 58
- Service Share Property
  - adding 58
- service template 20
  - active environment 20
  - adding processing 35
  - available operations by configuration type 46
  - compatibility with steps created in earlier versions 48
  - creating blank template 50
  - deleting 68
  - deleting processing 37
  - development environment 19
  - elements involved in development 19
  - flow of development 16
  - list of development features 42
  - plug-in 20
  - procedure for releasing 162
  - setting definition information 44
  - viewing another service template during editing 64
- service template definition information
  - editing 31
- service template development
  - overview 17
- service templates
  - copying 66
  - creating 50
  - creating new 28
  - deleting procedure 68
  - displaying list 65
  - editing 50
  - importing into active environment 164
  - parameters to set when creating or copying 50
  - procedure for copying 66
  - procedure for displaying list 65
  - procedure for importing into active environment 164
  - releasing 162
  - tasks 25
  - tasks when creating new service templates 25
  - tasks when editing 26
  - tasks when using existing service templates as-is 26
  - uniqueness 66
  - using existing templates 41
  - windows used for development 22
- setting definition information
  - service template 44
- setting procedure
  - input property mapping 81
  - output property mapping 82
  - service definition information 51
- settings
  - step definition information 79
- settings used when beginning debug process 134
- shared built-in service properties
  - overview 60
- specifying
  - Execution Directory 114
- standard error output
  - mapping to output properties 106
- standard output
  - mapping to output properties 106
- standard output of plug-ins 112
- step 20
  - definition information 79
  - procedure for changing definition information 78
- step information changeable while step execution is paused 140

## steps

adding 78

adding procedure 78

information inherited when pasting 88

procedure for copying 87

procedure for cutting 87

procedure for deleting 86

procedure for pasting 88

relationship to flow 76

selecting multiple 88

warning icon 85

structure and differences of build and release processes 167

subsequent step conditions

overview 80

surrogate pair characters and control characters in values of plug-in properties 148

## T

testing service templates 160

## U

using command or script return values as flow branching conditions (for values outside 0 to 63 range) 112

## V

validating service templates 122

visibility

property 62

## W

warning icon

steps 85