

**JP1 Version 13**

**JP1/Automatic Job Management System 3 System  
Design (Work Tasks) Guide**

**3021-3-L44-10(E)**

## Notices

### ■ Relevant program products

For details about the applicable OS versions, and the service packs and patches required for JP1/Automatic Job Management System 3, see the *Release Notes*.

*JP1/Automatic Job Management System 3 - Manager (For Windows):*

P-2A12-3KDL JP1/Automatic Job Management System 3 - Manager version 13-10

The above product includes the following:

P-CC2A12-4KDL JP1/Automatic Job Management System 3 - Manager version 13-10 (For Windows Server 2022, Windows Server 2019, Windows Server 2016)

P-CC2912-39DL JP1/Automatic Job Management System 3 - Web Console version 13-10 (For Windows Server 2022, Windows Server 2019, Windows Server 2016)

P-CC8412-39DL JP1/Automatic Job Management System 3 - Web Console version 13-10 (For Linux 7, Linux 8, Linux 9, Oracle Linux 7, Oracle Linux 8, Oracle Linux 9, SUSE Linux 12, SUSE Linux 15)

P-CC2A12-3NDL JP1/Automatic Job Management System 3 - Print Option Manager version 13-10 (For Windows Server 2022, Windows Server 2019, Windows Server 2016)

P-CC2A2C-6LDL JP1/Base version 13-10 (For Windows Server 2022, Windows Server 2019, Windows Server 2016)

*JP1/Automatic Job Management System 3 - Manager (For Linux):*

P-8412-3KDL JP1/Automatic Job Management System 3 - Manager version 13-10

The above product includes the following:

P-CC8412-4KDL JP1/Automatic Job Management System 3 - Manager version 13-10 (For Linux 7, Linux 8, Linux 9, Oracle Linux 7, Oracle Linux 8, Oracle Linux 9, SUSE Linux 12, SUSE Linux 15)

P-CC2912-39DL JP1/Automatic Job Management System 3 - Web Console version 13-10 (For Windows Server 2022, Windows Server 2019, Windows Server 2016)

P-CC8412-39DL JP1/Automatic Job Management System 3 - Web Console version 13-10 (For Linux 7, Linux 8, Linux 9, Oracle Linux 7, Oracle Linux 8, Oracle Linux 9, SUSE Linux 12, SUSE Linux 15)

P-CC8412-3NDL JP1/Automatic Job Management System 3 - Print Option Manager version 13-10 (For Linux 7, Linux 8, Linux 9, Oracle Linux 7, Oracle Linux 8, Oracle Linux 9, SUSE Linux 12, SUSE Linux 15)

P-CC842C-6LDL JP1/Base version 13-10 (For Linux 7, Linux 8, Linux 9, Oracle Linux 7, Oracle Linux 8, Oracle Linux 9, SUSE Linux 12, SUSE Linux 15)

*JP1/Automatic Job Management System 3 - Agent (For Windows):*

P-2A12-33DL JP1/Automatic Job Management System 3 - Agent version 13-10

The above product includes the following:

P-CC2A12-43DL JP1/Automatic Job Management System 3 - Agent version 13-10 (For Windows Server 2022, Windows Server 2019, Windows Server 2016)

P-CC2A2C-6LDL JP1/Base version 13-10 (For Windows Server 2022, Windows Server 2019, Windows Server 2016)

*JP1/Automatic Job Management System 3 - Agent (For AIX):*

P-1M12-33DL JP1/Automatic Job Management System 3 - Agent version 13-10

The above product includes the following:

P-CC1M12-43DL JP1/Automatic Job Management System 3 - Agent version 13-10 (For AIX)

P-CC1M2C-6LDL JP1/Base version 13-10 (For AIX)

*JP1/Automatic Job Management System 3 - Agent (For Linux):*

P-8412-33DL JP1/Automatic Job Management System 3 - Agent version 13-10

The above product includes the following:

P-CC8412-43DL JP1/Automatic Job Management System 3 - Agent version 13-10 (For Linux 7, Linux 8, Linux 9, Oracle Linux 7, Oracle Linux 8, Oracle Linux 9, SUSE Linux 12, SUSE Linux 15)

P-CC842C-6LDL JP1/Base version 13-10 (For Linux 7, Linux 8, Linux 9, Oracle Linux 7, Oracle Linux 8, Oracle Linux 9, SUSE Linux 12, SUSE Linux 15)

*JP1/Automatic Job Management System 3 - Agent Minimal Edition (For Windows):*

P-2A12-38DL JP1/Automatic Job Management System 3 - Agent Minimal Edition version 13-10

The above product includes the following:

P-CC2A12-48DL JP1/Automatic Job Management System 3 - Agent Minimal Edition version 13-10 (For Windows Server 2022, Windows Server 2019, Windows Server 2016)

P-CC2A2C-6LDL JP1/Base version 13-10 (For Windows Server 2022, Windows Server 2019, Windows Server 2016)

*JP1/Automatic Job Management System 3 - Agent Minimal Edition (For Linux):*

P-8412-38DL JP1/Automatic Job Management System 3 - Agent Minimal Edition version 13-10

The above product includes the following:

P-CC8412-48DL JP1/Automatic Job Management System 3 - Agent Minimal Edition version 13-10 (For Linux 7, Linux 8, Linux 9, Oracle Linux 7, Oracle Linux 8, Oracle Linux 9, SUSE Linux 12, SUSE Linux 15)

P-CC842C-6LDL JP1/Base version 13-10 (For Linux 7, Linux 8, Linux 9, Oracle Linux 7, Oracle Linux 8, Oracle Linux 9, SUSE Linux 12, SUSE Linux 15)

*JP1/Automatic Job Management System 3 - View (For Windows):*

P-2A12-34DL JP1/Automatic Job Management System 3 - View version 13-10

The above product includes the following:

P-CC2A12-44DL JP1/Automatic Job Management System 3 - View version 13-10 (For Windows Server 2022, Windows Server 2019, Windows Server 2016, Windows 11, Windows 10)

P-CC2A12-3MDL JP1/Automatic Job Management System 3 - Print Option version 13-10 (For Windows Server 2022, Windows Server 2019, Windows Server 2016, Windows 11, Windows 10)

## ■ Trademarks

HITACHI, JP1, Job Management Partner 1, uCosminexus, HiRDB are either trademarks or registered trademarks of Hitachi, Ltd. in Japan and 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 the Apache Software Foundation (<http://www.apache.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/>

1. This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.openssl.org/>)

2. This product includes cryptographic software written by Eric Young ([ey@cryptsoft.com](mailto:ey@cryptsoft.com))

3. This product includes software written by Tim Hudson ([tjh@cryptsoft.com](mailto:tjh@cryptsoft.com))

4. This product includes the OpenSSL Toolkit software used under OpenSSL License and Original SSLeay License. OpenSSL License and Original SSLeay License are as follow:

#### LICENSE ISSUES

=====

The OpenSSL toolkit stays under a double license, i.e. both the conditions of the OpenSSL License and the original SSLeay license apply to the toolkit.

See below for the actual license texts.

#### OpenSSL License

-----

/\* =====

\* Copyright (c) 1998-2019 The OpenSSL Project. All rights reserved.

\*

\* Redistribution and use in source and binary forms, with or without  
\* modification, are permitted provided that the following conditions  
\* are met:

\*

\* 1. Redistributions of source code must retain the above copyright  
\* notice, this list of conditions and the following disclaimer.

\*

\* 2. Redistributions in binary form must reproduce the above copyright  
\* notice, this list of conditions and the following disclaimer in  
\* the documentation and/or other materials provided with the  
\* distribution.

\*

\* 3. All advertising materials mentioning features or use of this  
\* software must display the following acknowledgment:  
\* "This product includes software developed by the OpenSSL Project  
\* for use in the OpenSSL Toolkit. (<http://www.openssl.org/>)"

\*

\* 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to  
\* endorse or promote products derived from this software without  
\* prior written permission. For written permission, please contact  
\* [openssl-core@openssl.org](mailto:openssl-core@openssl.org).

```

*
* 5. Products derived from this software may not be called "OpenSSL"
* nor may "OpenSSL" appear in their names without prior written
* permission of the OpenSSL Project.
*
* 6. Redistributions of any form whatsoever must retain the following
* acknowledgment:
* "This product includes software developed by the OpenSSL Project
* for use in the OpenSSL Toolkit (http://www.openssl.org/)"
*
* THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS" AND ANY
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
* PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
* OF THE POSSIBILITY OF SUCH DAMAGE.
* =====
*
* This product includes cryptographic software written by Eric Young
* (eay@cryptsoft.com). This product includes software written by Tim
* Hudson (tjh@cryptsoft.com).
*
*/
Original SSLeay License
-----
/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
* All rights reserved.
*
* This package is an SSL implementation written
* by Eric Young (eay@cryptsoft.com).
* The implementation was written so as to conform with Netscapes SSL.
*
* This library is free for commercial and non-commercial use as long as
* the following conditions are aheared to. The following conditions
* apply to all code found in this distribution, be it the RC4, RSA,
* lhash, DES, etc., code; not just the SSL code. The SSL documentation

```

- \* included with this distribution is covered by the same copyright terms
- \* except that the holder is Tim Hudson (tjh@cryptsoft.com).
- \*
- \* Copyright remains Eric Young's, and as such any Copyright notices in
- \* the code are not to be removed.
- \* If this package is used in a product, Eric Young should be given attribution
- \* as the author of the parts of the library used.
- \* This can be in the form of a textual message at program startup or
- \* in documentation (online or textual) provided with the package.
- \*
- \* Redistribution and use in source and binary forms, with or without
- \* modification, are permitted provided that the following conditions
- \* are met:
- \* 1. Redistributions of source code must retain the copyright
- \* notice, this list of conditions and the following disclaimer.
- \* 2. Redistributions in binary form must reproduce the above copyright
- \* notice, this list of conditions and the following disclaimer in the
- \* documentation and/or other materials provided with the distribution.
- \* 3. All advertising materials mentioning features or use of this software
- \* must display the following acknowledgement:
- \* "This product includes cryptographic software written by
- \* Eric Young (eay@cryptsoft.com)"
- \* The word 'cryptographic' can be left out if the routines from the library
- \* being used are not cryptographic related :-).
- \* 4. If you include any Windows specific code (or a derivative thereof) from
- \* the apps directory (application code) you must include an acknowledgement:
- \* "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
- \*
- \* THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS" AND
- \* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
- \* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
- \* ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
- \* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
- \* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
- \* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
- \* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
- \* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
- \* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
- \* SUCH DAMAGE.
- \*
- \* The licence and distribution terms for any publically available version or
- \* derivative of this code cannot be changed. i.e. this code cannot simply be

\* copied and put under another distribution licence

\* [including the GNU Public Licence.]

\*/

This product includes the OpenSSL library.

The OpenSSL library is licensed under Apache License, Version 2.0.

<https://www.apache.org/licenses/LICENSE-2.0>

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 IAIK of Graz University of Technology.

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

This product includes software developed by Andy Clark.

Java is a registered trademark of Oracle and/or its affiliates.



## ■ Microsoft product name abbreviations

This manual uses the following abbreviations for Microsoft product names.

| Abbreviation         | Full name or meaning                         |
|----------------------|--|
| Excel                | Microsoft(R) Excel                           |
|                      | Microsoft(R) Office Excel                    |
| Exchange Server      | Microsoft(R) Exchange 2000 Enterprise Server |
|                      | Microsoft(R) Exchange 2000 Server            |
|                      | Microsoft(R) Exchange Server                 |
| Internet Explorer    | Windows(R) Internet Explorer(R)              |
| Microsoft Edge       | Microsoft(R) Edge                            |
| Microsoft SQL Server | Microsoft(R) SQL Server                      |
|                      | Microsoft(R) SQL Server Enterprise Edition   |
| MSMQ                 | Microsoft(R) Message Queue Server            |

| Abbreviation        |              | Full name or meaning                           |
|---------------------|--------------|--|
| Outlook             | Outlook 2016 | Microsoft(R) Office Outlook(R) 2016            |
|                     | Outlook 2019 | Microsoft(R) Office Outlook(R) 2019            |
|                     | Outlook 2021 | Microsoft(R) Office Outlook(R) 2021            |
| Outlook Express     |              | Microsoft(R) Outlook(R) Express                |
| Windows 10          |              | Windows(R) 10 Enterprise                       |
|                     |              | Windows(R) 10 Pro                              |
|                     |              | Windows(R) 10 Home                             |
| Windows 11          |              | Windows(R) 11 Enterprise                       |
|                     |              | Windows(R) 11 Pro                              |
|                     |              | Windows(R) 11 Home                             |
| Windows Server 2016 |              | Microsoft(R) Windows Server(R) 2016 Datacenter |
|                     |              | Microsoft(R) Windows Server(R) 2016 Standard   |
| Windows Server 2019 |              | Microsoft(R) Windows Server(R) 2019 Datacenter |
|                     |              | Microsoft(R) Windows Server(R) 2019 Standard   |
| Windows Server 2022 |              | Microsoft(R) Windows Server(R) 2022 Datacenter |
|                     |              | Microsoft(R) Windows Server(R) 2022 Standard   |

*Windows* is sometimes used generically, referring to Windows Server 2022, Windows Server 2019, Windows Server 2016, Windows 11, and Windows 10.

## ■ Restrictions

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

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

No part of this material may be reproduced in any form or by any means without permission in writing from the publisher.

## ■ Issued

Sep. 2024: 3021-3-L44-10(E)

## ■ Copyright

Copyright (C) 2023, 2024, Hitachi, Ltd.

Copyright (C) 2023, 2024, Hitachi Solutions, Ltd.



## Summary of amendments

For details about the amendments, see the manual *JP1/Automatic Job Management System 3 Overview*.

# Preface

This manual describes how to design work tasks for JP1/Automatic Job Management System 3 (abbreviated hereafter to *JP1/AJS3*). Read this manual in conjunction with the manual *JP1/Automatic Job Management System 3 Overview*, which describes JP1/AJS3 functionality.

For details on the prerequisites before reading this manual, see the manual *JP1/Automatic Job Management System 3 Overview*.

## ■ Organization of this manual

This manual organized into the following chapters. The manual is a common reference for all supported operating systems. Any platform-dependent differences in functionality are noted in the manual.

### *1. Overview of Work Task Design*

Chapter 1 provides an overview of designing work tasks that will be automated with JP1/AJS3. The chapter also includes the sequence of design steps and considerations during design.

### *2. Job Definition and Job Execution Order Considerations*

Chapter 2 describes the considerations necessary for constructing the jobs and jobnets that are required for automating work tasks with JP1/AJS3.

### *3. Operation Calendar and Execution Schedule Considerations*

Chapter 3 describes the considerations necessary for setting a calendar and execution schedule for JP1/AJS3 operation.

### *4. Execution Registration Method Considerations*

Chapter 4 describes how to register jobnets for execution in JP1/AJS3.

### *5. Monitoring Method Considerations*

Chapter 5 describes the considerations necessary for using JP1/AJS - View to monitor jobnets.

### *6. Access Permission Considerations*

Chapter 6 describes the considerations necessary for setting the access permissions for jobnets and for associating JP1 users and OS users.

### *7. Cautionary Notes on Designing Work Tasks*

Chapter 7 provides cautionary notes on designing work tasks.

### *8. Definition Pre-Check*

Chapter 8 describes the procedure for the definition pre-check performed before actual JP1/AJS3 operation starts. The chapter also describes check items and includes cautionary notes.

# Contents

Notices 2

Summary of amendments 9

Preface 10

## **1 Overview of Work Task Design 15**

1.1 Flow of work task design 16

1.2 Automating work tasks 17

1.3 Flow of work task automation 18

1.4 Key decisions when planning work task automation 19

## **2 Job Definition and Job Execution Order Considerations 20**

2.1 Job definition considerations 21

2.1.1 Investigating tasks to automate 21

2.1.2 Designing executable files 21

2.1.3 Job type considerations 23

2.1.4 Job definition considerations 26

2.2 Jobnet definition considerations 30

2.2.1 Job execution order considerations 30

2.2.2 Corrective action when job execution fails 38

2.2.3 Jobnet definition considerations 39

2.2.4 Using jobnet connectors to control the order of root jobnet execution 40

2.2.5 Using wait conditions to control the order of unit execution 59

2.2.6 Considerations regarding the use of macro variables 109

2.3 Tips on work task automation 119

2.3.1 Processing with a distributed load 119

2.3.2 JP1/AJS3 functions useful for work task automation 125

2.4 Jobnet definition examples 128

2.4.1 Executing a process in a specified file (example of defining a jobnet consisting of standard jobs) 128

2.4.2 Executing a process when one of multiple conditions is satisfied (example of defining a jobnet that uses an OR job) 128

2.4.3 Dynamically changing a process depending on the result of a preceding job (example of defining a jobnet that uses a judgment job) 130

2.4.4 Executing an event-driven process (example of defining a jobnet that uses an event job and a custom event job) 136

2.4.5 Sending a JP1 event at completion of the preceding job or when an event occurs (example of defining a jobnet that uses a Send JP1 event job) 152

2.4.6 Executing a specific process when a job ends abnormally (example of defining a jobnet that uses a recovery unit) 154

- 2.4.7 Controlling the execution order of a root jobnet (example of defining a jobnet that uses a jobnet connector) 156
- 2.4.8 Controlling the order of unit execution in different jobnets (example of defining a jobnet that uses a wait condition) 157
- 2.4.9 Passing information that changes dynamically to a succeeding unit (example of defining a jobnet that uses a passing information setting job) 160
- 2.4.10 Executing jobs in a cloud environment (example of defining a jobnet that uses flexible jobs) 170
- 2.4.11 Executing a job on multiple execution agents simultaneously (example of defining a jobnet that uses flexible jobs) 171
- 2.4.12 Linking with a business system on the web (example of defining a jobnet that uses HTTP connection jobs) 173
- 2.4.13 Automatic retry for abnormally ending jobs 178

### **3 Operation Calendar and Execution Schedule Considerations 193**

- 3.1 Flow of calendar and schedule planning 194
- 3.2 Considerations when defining a calendar for JP1/AJS3 operation 195
- 3.3 Considerations when defining a jobnet execution schedule 196
- 3.4 Start condition considerations 197
  - 3.4.1 Detailed considerations 197
- 3.5 Setting schedules 199
  - 3.5.1 Establishing schedules for applications that extend over two days 199
  - 3.5.2 Setting multiple execution start times 202
  - 3.5.3 Defining a different schedule for some jobs in a jobnet 203
  - 3.5.4 Executing the same jobnet several times a day (defining cycle jobs) 207
  - 3.5.5 Shifting the scheduled execution date forward or back based on a calculated schedule (Schedule by days from start) 208
  - 3.5.6 Defining a schedule that has different behavior at different times of the month 211
  - 3.5.7 Defining a different calendar for each application 212

### **4 Execution Registration Method Considerations 218**

- 4.1 Jobnet execution registration methods 219
  - 4.1.1 Planned execution registration 219
  - 4.1.2 Fixed execution registration 220
  - 4.1.3 Immediate execution registration 220

### **5 Monitoring Method Considerations 222**

- 5.1 Monitoring methods in JP1/AJS3 - View 223
  - 5.1.1 Planning which work tasks to monitor in the Summary Monitor window 223
  - 5.1.2 Jobnet delay monitoring considerations 223
  - 5.1.3 End delay monitoring based on time-required-for-execution 224
- 5.2 Considerations for monitoring by using a Web GUI (Job Portal) 225
  - 5.2.1 Considering what to monitor in the Dashboard screen 225
- 5.3 Considerations for monitoring by using an API-based user application 226

|          |  |            |
|----------|--|------------|
| <b>6</b> | <b>Access Permission Considerations</b>                                      | <b>227</b> |
| 6.1      | Steps for considering work task access permissions                           | 228        |
| 6.2      | Ranges for setting access permissions  | 229        |
| 6.3      | Setting registered users   | 230        |
| 6.4      | Setting access permissions   | 232        |
| 6.4.1    | Access permission considerations   | 232        |
| 6.5      | Mapping users  | 254        |
| 6.5.1    | Considerations when mapping users  | 254        |
| <br>     |  |            |
| <b>7</b> | <b>Cautionary Notes on Designing Work Tasks</b>                              | <b>258</b> |
| 7.1      | Notes on the number of root jobnets registered for execution                 | 259        |
| 7.2      | Relationship between number of logs to keep and performance                  | 260        |
| 7.3      | Notes on using PC jobs   | 262        |
| 7.4      | Notes on using Unix jobs   | 264        |
| 7.5      | Notes on using a recovery unit   | 267        |
| 7.6      | Notes on using event jobs  | 269        |
| 7.6.1    | Notes on the Receive JP1 Event job   | 271        |
| 7.6.2    | Notes on the Monitoring Files job  | 275        |
| 7.6.3    | Notes on the Receive Mail job  | 293        |
| 7.6.4    | Notes on the Monitoring Log Files job  | 293        |
| 7.6.5    | Notes on the Monitoring Event Log job  | 295        |
| 7.6.6    | Notes on the Interval Control job  | 297        |
| 7.6.7    | Notes on defining passing information  | 298        |
| 7.6.8    | Notes on restarting the JP1/AJS3 service while event jobs are running        | 299        |
| 7.6.9    | Monitoring events or messages issued by JP1/AJS3                             | 301        |
| 7.7      | Notes on using action jobs   | 306        |
| 7.7.1    | Notes on the Send JP1 Event job  | 306        |
| 7.7.2    | Notes on the Send Mail job   | 307        |
| 7.7.3    | Notes on the OpenView Status Report job                                      | 307        |
| 7.7.4    | Notes on the Local Power Control job and Remote Power Control job            | 308        |
| 7.8      | Notes on using flexible jobs   | 310        |
| 7.9      | Notes on using HTTP connection jobs  | 315        |
| 7.10     | Notes on using custom event jobs   | 317        |
| 7.10.1   | Notes on defining passing information  | 318        |
| 7.10.2   | Notes on restarting the JP1/AJS3 service while custom event jobs are running | 319        |
| 7.11     | Notes on defining jobs   | 322        |
| 7.11.1   | Notes on the standard output file and standard error output file             | 322        |
| 7.11.2   | Notes on the execution priority of jobs                                      | 326        |
| 7.11.3   | Checking the return code of a job  | 329        |
| 7.12     | Startup performance of scheduler services                                    | 334        |

|          |  |            |
|----------|--|------------|
| <b>8</b> | <b>Definition Pre-Check</b>  | <b>335</b> |
| 8.1      | Checking definitions before live JP1/AJS3 operation                | 336        |
| 8.1.1    | Overview of the definition pre-check function and cautionary notes | 337        |

## **Appendixes 343**

|     |  |     |
|-----|--|-----|
| A   | Return Values from Event Jobs, Custom Event Jobs and Action Jobs | 344 |
| B   | Information Passed by Event Jobs and Custom Event Jobs           | 350 |
| B.1 | Information Passed by Event Jobs                                 | 350 |
| B.2 | Information Passed by Custom Event Jobs                          | 353 |
| C   | Files Used for HTTP Connection Jobs                              | 355 |
| C.1 | Connection configuration file                                    | 355 |
| C.2 | Transmission information file                                    | 360 |
| C.3 | Transmission information file (URL parameter)                    | 361 |
| C.4 | Transmission information file (Message body)                     | 362 |
| C.5 | Status file  | 362 |
| C.6 | Received header file   | 363 |
| C.7 | Received body file   | 364 |
| D   | Version Revisions  | 365 |
| E   | Reference Material for This Manual                               | 366 |
| F   | Glossary   | 367 |

## **Index 368**

# 1

## Overview of Work Task Design

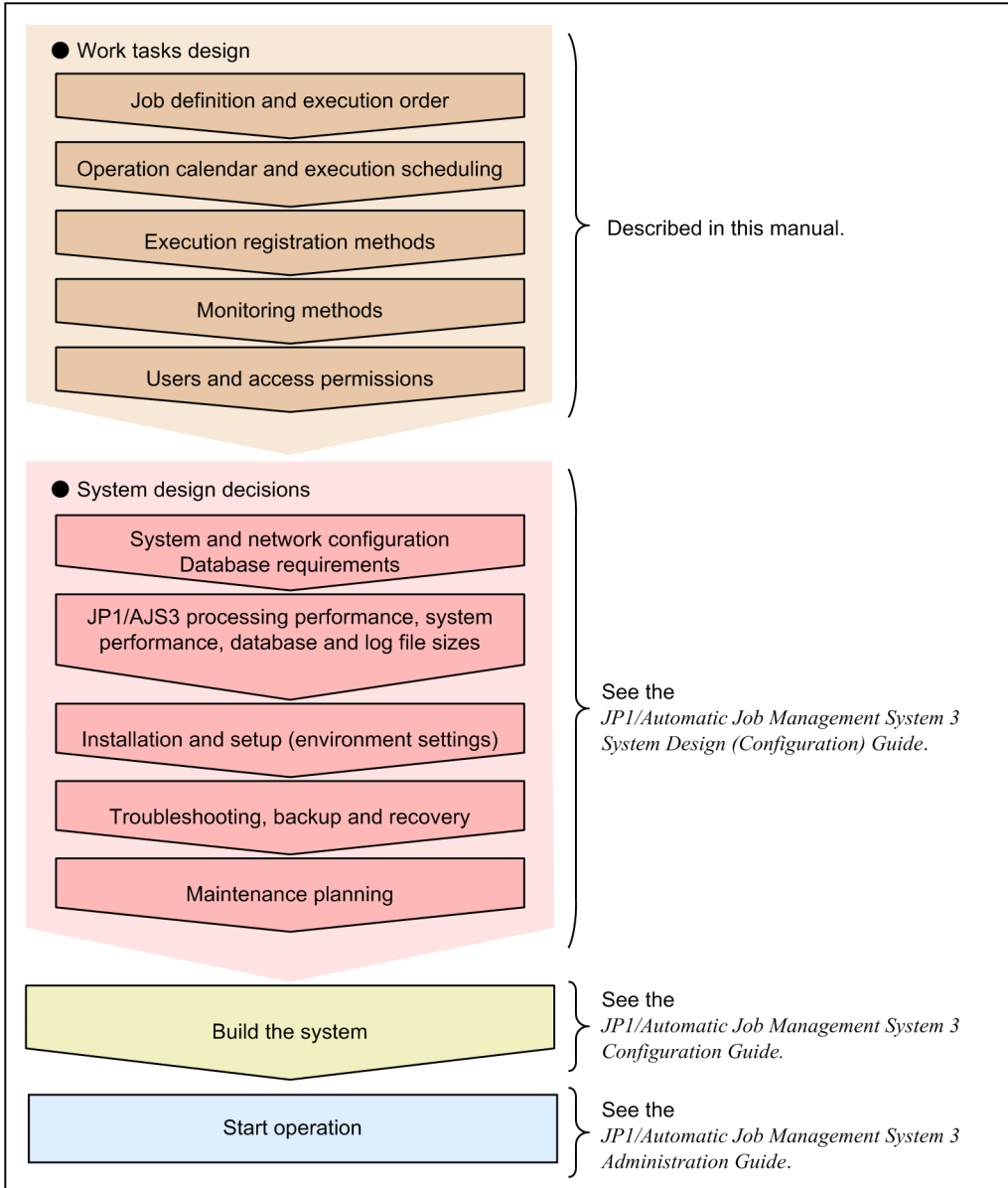
This chapter gives an overview of designing work tasks that will be automated with JP1/AJS3.

# 1.1 Flow of work task design

The design flow for deploying JP1/AJS3 can be broadly categorized as work task design for automating job execution under JP1/AJS3, and system design for installing and running JP1/AJS3 efficiently.

The following figure shows the basic design steps when you deploy JP1/AJS3:

Figure 1–1: JP1/AJS3 design steps



Note: The sequence might differ depending on the customer's environment.

This manual describes the matters to be considered when planning work task automation.

For information about system design, see the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*. For information about JP1/AJS3 operation, see the *JP1/Automatic Job Management System 3 Administration Guide*.



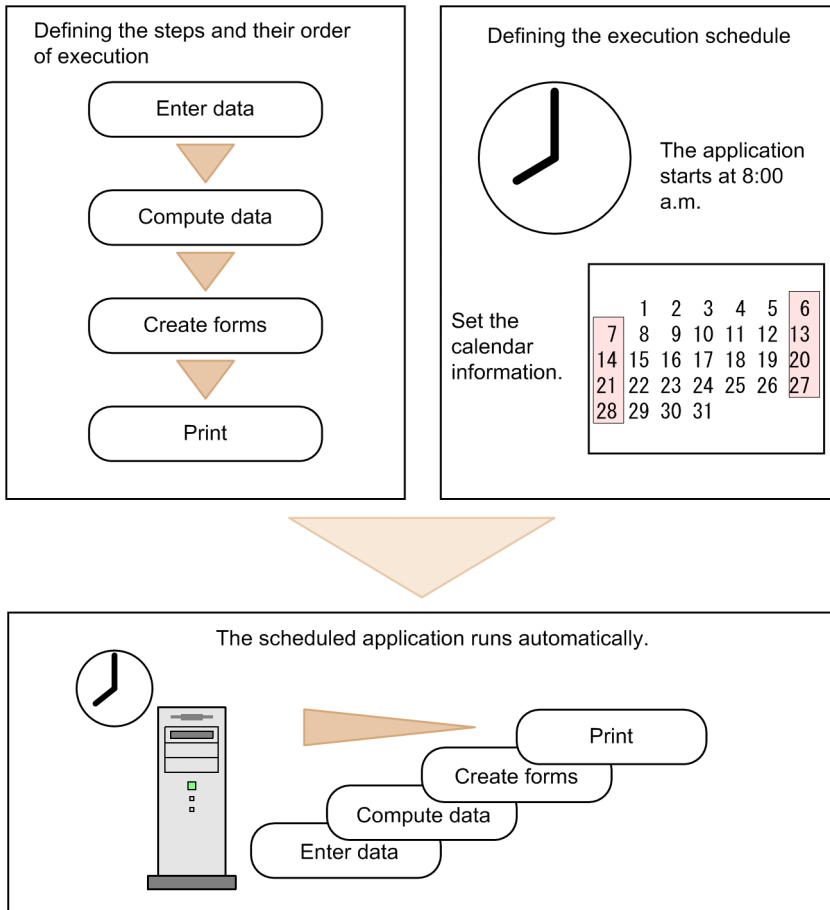
## 1.2 Automating work tasks

To use JP1/AJS3 to automate work tasks, you must define the following information:

- The content of each work task and its execution order
- The calendar or schedule determining when and under what conditions each work task is executed

The following figure gives an overview of work task automation:

Figure 1–2: Overview of work task automation



When you define work tasks for automation by JP1/AJS3, due consideration must be given to these aspects.

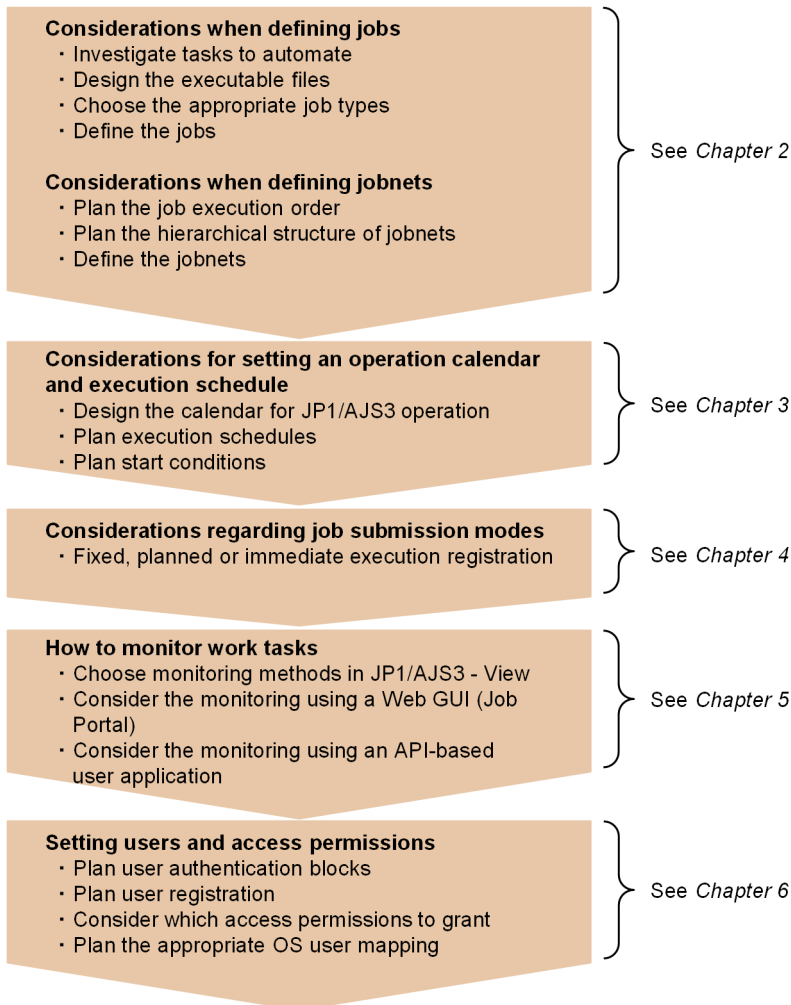
## 1.3 Flow of work task automation

---

JP1/AJS3 automates work tasks based on the individual components that make up each task, such as commands, application programs, and shell scripts. Each component is defined as a *job*. When assigned an execution schedule, these jobs collectively form a *jobnet*, which can be executed automatically at a date and time determined by one or more *schedule rules*.

The following figure shows the flow of work task automation in JP1/AJS3.

Figure 1–3: Flow of work task automation



## 1.4 Key decisions when planning work task automation

---

The following table lists the key questions to be decided when you plan work task automation, and the relevant chapters in this manual.

Table 1–1: Work task automation decisions and relevant chapters

| No. | Key decisions   | Relevant chapter   |
|-----|---|--|
| 1   | Defining jobs   | <i>2.1 Job definition considerations</i>                           |
| 2   | Defining jobnets  | <i>2.2 Jobnet definition considerations</i>                        |
| 3   | Considerations for setting an operation calendar and execution schedule | <i>3. Operation Calendar and Execution Schedule Considerations</i> |
| 4   | How to register jobnets for execution                                   | <i>4. Execution Registration Method Considerations</i>             |
| 5   | How to monitor work tasks   | <i>5. Monitoring Method Considerations</i>                         |
| 6   | Setting users and access permissions                                    | <i>6. Access Permission Considerations</i>                         |
| 7   | Consideration of cautionary notes associated with work task design      | <i>7. Cautionary Notes on Designing Work Tasks</i>                 |

For information about JP1/AJS3 functions, see the manual *JP1/Automatic Job Management System 3 Overview*.

# 2

## Job Definition and Job Execution Order Considerations

In JP1/AJS3, users can automate work tasks by defining them as jobs and jobnets, and setting schedules and conditions that govern when the task should start.

This chapter describes the considerations necessary for constructing the jobs and jobnets that are required for automating work tasks with JP1/AJS3.

## 2.1 Job definition considerations

The individual operations in an automated work task are carried out in the form of jobs. A job is a group of commands, shell scripts, or application programs, and is the smallest building block of any automated task. Considerations include how best to pare down the steps involved in a work task, and the design of executable programs to achieve the required processing.

### 2.1.1 Investigating tasks to automate

You must investigate the work tasks that you want to automate with JP1/AJS3. In addition to repetitive work tasks executed on a fixed daily or monthly schedule, for example, JP1/AJS3 supports the automation of work tasks whose processing changes dynamically depending on the result of the preceding process. Non-regular work tasks which are not executed at a specific date or time can be set to execute automatically in response to a particular event, such as when a file is updated or a specific event is received.

To automate work tasks with JP1/AJS3, users should be familiar with the functionality that JP1/AJS3 has to offer. For information about JP1/AJS3 functions, see the manual *JP1/Automatic Job Management System 3 Overview*.

To help users get started with the work task automation process, this manual gives an introduction to the specific JP1/AJS3 functions they are most likely to use. For an overview of the JP1/AJS3 functions for automating work tasks, see [2.3.2 JP1/AJS3 functions useful for work task automation](#).

### 2.1.2 Designing executable files

JP1/AJS3 processes work tasks based on *executable files*, such as batch files and shell scripts. This section identifies the considerations involved in designing executable files to carry out work tasks.

When designing executable files, consider the following points:

#### (1) Use a format supported by JP1/AJS3

Processing cannot be automated in JP1/AJS3 unless it complies with a supported executable file format. The following table lists the types of executable files that can be used with JP1/AJS3.

Table 2–1: Executable file types supported by JP1/AJS3

| Host type    | Supported executable files  |
|--------------|---|
| Windows host | <ul style="list-style-type: none"><li>• .exe files</li><li>• .com files</li><li>• .cmd files</li><li>• Batch files (.bat)</li><li>• Script files (.spt) created in JP1/Script</li><li>• Data files associated with an application program (as indicated by the extension)</li></ul> |
| UNIX host    | <ul style="list-style-type: none"><li>• Shell scripts</li><li>• Executable files</li></ul>  |

#### (2) Define one command per executable file

We recommend that you define only one command in each executable file.

Dividing work task processing in this way allows you to see which processing ends normally and which does not. Because you can identify the specific process that caused a work task to terminate abnormally, you can then either rerun the work task from that process, or rerun just the process that ended abnormally.

### (3) Choose processing that does not generate on-screen messages or require a response

JP1/AJS3 runs executable files in the background. Therefore, do not use an executable file that displays a window or message and then waits for a user response. However, JP1/AJS3 can use executable files if, for example, the **Yes** button is automatically selected on a displayed message dialog box and the process proceeds.

### (4) Choose processing that outputs a return code

JP1/AJS3 judges whether processing has ended normally or abnormally based on the return code of the executable file. Therefore, create executable files whose return codes reflect the processing result.

For processing by batch file:

Use the `jplexec` or `jplexit` command to ensure that the executable programs in the batch file output return codes that reflect the processing result. To enable analysis of the cause when a command ends abnormally, ensure that the return code that caused the abnormal end is output unaltered.

For details on the `jplexit` and `jplexec` commands, see 3. *Commands Used for Normal Operations* in the manual *JP1/Automatic Job Management System 3 Command Reference*.

For processing by shell script:

With shell scripts, ensure that a return code that reflects the processing result is output, as in the following example:

```
:
RC=$?
exit $RC
```

Set the threshold values that determine the ranges for *end with warning* or *abnormal end* results.

### (5) Choose processing with a short execution time

If the execution time for a single process is too long, you might not be able to determine whether the work task is proceeding normally, or processing has stopped due to an error of some kind. As a guide, make sure that no individual process takes longer than two hours.

### (6) Use meaningful file and folder names

We recommend that you create a naming convention for executable files and the folders and directories where executable files are saved. Using consistent names with work tasks, processes, and other elements used with JP1/AJS3 makes it easier to manage work tasks as a whole. One effective method is to define a set of identifiers that indicate work task names, process names, processing cycles, execution locations, and so on.

One-byte alphanumeric characters, multi-byte characters and the following symbols can be used in the names of work tasks and processes used with JP1/AJS3.

```
! # $ % + @ - (hyphen) . (period) _ (underscore)
```

Note that some symbols have special meanings in command interpreters such as UNIX shells. Because symbols might cause unpredictable results, avoid using them for work task and process names.

In addition, do not use the following characters or symbols:

- Periods ( . ) or at marks ( @ ) at the beginning of work task or process names
- Machine-dependent characters

## (7) Save executable files on the host where the processing is to be executed

Save the executable files (such as commands and batch files) on the host where you intend the processing to be executed.

## (8) Other points

- When specifying commands and input or output files in executable files, specify an absolute path including the folder names.
- To enable analysis of the cause when a command ends abnormally, output an arbitrary message to the standard error output by, for example, using an `echo` command. It is useful for this message to include information such as the cause of the error and how to recover from it, the work task name, and notes on rerunning the work task. You can view these messages in the Execution Result Details dialog box in JP1/AJS3 - View. For details about the Execution Result Details dialog box, see *12.3.39 Execution Result Details dialog box* in the *JP1/Automatic Job Management System 3 Operator's Guide*.
- As a general rule, if you use environment variables, you should define them in the executable file.
- When using a shell script, declare the shell used at the head of the file. For details about the supported shells, also see *2.5.3 OS user environment when a job is executed* in the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*.

## 2.1.3 Job type considerations

Because there are many different types of jobs, you must order the jobs that you are defining according to the type of process required. JP1/AJS3 provides the following job types:

- Standard job
- OR job
- Judgment job
- Event job
- Action job
- Custom job
- Passing information setting job
- HTTP connection job
- Custom event job

The characteristics of each type of job are described below.

### (1) Standard job

This type of job executes processing by specifying an executable file. The following table lists the types of executable files that can be specified in each type of *standard job*.

Table 2–2: Executable files specifiable in a standard job

| No. | Job type     | Description   | Specifiable executable files  |
|-----|--------------|---|---|
| 1   | Unix job     | Process executed on a UNIX host.  | <ul style="list-style-type: none"> <li>• Executable file</li> <li>• Shell script</li> </ul>   |
| 2   | PC job       | Process executed on a Windows host.   | <ul style="list-style-type: none"> <li>• .exe file</li> <li>• .com file</li> <li>• .cmd file</li> <li>• .pif file</li> <li>• .bat file</li> <li>• .spt file<sup>#</sup> (script file created with JP1/Script)</li> <li>• Data file that has a file type (extension) associated with the application</li> </ul>  |
| 3   | QUEUE job    | <p>Process executed by submitting a job to a specified queue.</p> <p>This job type is used for linking with other systems (such as JP1/OJE).</p>  | <ul style="list-style-type: none"> <li>• Executable file</li> <li>• Shell script</li> <li>• .exe file</li> <li>• .com file</li> <li>• .cmd file</li> <li>• .pif file</li> <li>• .bat file</li> <li>• .spt file<sup>#</sup> (script file created with JP1/Script)</li> <li>• Data file that has a file type (extension) associated with the application</li> </ul> |
| 4   | Flexible job | <ul style="list-style-type: none"> <li>• Process executed on an agent host that is not directly managed by the manager host.</li> <li>• Job broadcast to multiple execution agents.</li> <li>• Flexible jobs can be used to execute processing in a cloud environment. Load balancers are supported.</li> </ul> | <ul style="list-style-type: none"> <li>• Executable file</li> <li>• Shell script</li> <li>• .exe file</li> <li>• .com file</li> <li>• .cmd file</li> <li>• .bat file</li> </ul>   |

#  
To execute an .spt file, JP1/Script must also be installed on the host running the job.

Supplementary note

In an environment in which the agent host can be managed by the manager host because the host name and IP address are fixed, we recommend that you use PC jobs or Unix jobs rather than flexible jobs.

## (2) OR job

For an *OR job*, you define a number of jobs (*event jobs* and *custom event jobs*) which will monitor the system for a specific event. You define these event jobs and custom event jobs as the preceding jobs of the OR job. If any one of the monitored events occurs, the OR job executes its succeeding job.

For details, see 3.1.1(1)(b) *OR job* in the manual *JP1/Automatic Job Management System 3 Overview*.

## (3) Judgment job

A *judgment job* checks whether a given condition is satisfied.

The judgment can be based on the following conditions:



- The return code of the preceding job
- Whether a specific file exists
- The result of a comparison between variables

For details, see 3.1.1(1)(c) *Judgment job* in the manual *JP1/Automatic Job Management System 3 Overview*.

## (4) Event job

An *event job* monitors an event occurring in the system. You can define an event job as the start condition for a job flow or jobnet, so that the particular job or jobnet will be executed only when the monitored event occurs.

The following table describes each event job.

Table 2–3: Types of event jobs

| No. | Event job name                      | Description   |
|-----|-------------------------------------|---|
| 1   | Receive JP1 event job               | This job terminates when it receives a specific event from JP1/Base.  |
| 2   | Monitoring files job                | This job terminates when a specific file is created, deleted, or updated.   |
| 3   | Receive mail job                    | This job terminates when it receives a specific email.  |
| 4   | Monitoring log files job            | This job is linked with the log file trapping of JP1/Base, and terminates when specific information is written to the specified log file.     |
| 5   | Monitoring event log job            | This job is linked with the log file trapping of JP1/Base, and terminates when specific information is written to the Windows event log file. |
| 6   | Interval control job                | This job terminates when the specified time period elapses.   |
| 7   | Receive MQ message job <sup>#</sup> | This job terminates when it receives a specific message from MQSeries.  |
| 8   | Receive MSMQ message job            | This job terminates when it receives a specific message from MSMQ.  |

### Note

JP1/AJS3 must be linked with the appropriate program to use a Receive mail job, Receive MQ message job, or Receive MSMQ message job. For details, see the *JP1/Automatic Job Management System 3 Linkage Guide*.

#

Does not apply to Linux.

Operation of event jobs is independent of JP1 user permissions and the authority level defined for the job (by owner, JP1 resource group, or by the user executing the job). In Windows, because event job operation is governed by the account rights to the JP1/AJS3 service, JP1/AJS3 service rights must be set in advance.

## (5) Action job

An *action job* executes a specific process. You can combine an action job with an event job or a custom event job to execute a process (action) when an event occurs. Typical actions might be to send a JP1 event, an email message, or a status notification.

The following table describes each action job.

Table 2–4: Types of action jobs

| No. | Action job name    | Description   |
|-----|--------------------|---|
| 1   | Send JP1 event job | Sends a JP1 event to the event service of JP1/Base. |

| No. | Action job name                  | Description                  |
|-----|----------------------------------|------------------------------|
| 2   | Send mail job                    | Sends an email message.      |
| 3   | OpenView Status Report job       | Sends a status to HP NNM.    |
| 4   | Send MQ message job <sup>#</sup> | Sends a message to MQSeries. |
| 5   | Send MSMQ message job            | Sends a message to MSMQ.     |

#### Note

JP1/AJS3 must be linked with the appropriate program to use a Send mail job, Send MQ message job, Send MSMQ message job, or OpenView Status Report job. For details on program linkage, see the *JP1/Automatic Job Management System 3 Linkage Guide*.

#

Does not apply to Linux.

## (6) Custom job

A *custom job* is used to link an external program with JP1/AJS3 to execute processing.

For details, see 3. *Adding Custom Jobs* in the *JP1/Automatic Job Management System 3 Linkage Guide*.

## (7) Passing information setting job

A *passing information setting job* extracts the necessary information from a standard output file that is output by a preceding job, and then passes that information to a succeeding job. When a job needs to reference information that changes dynamically, you can have a job output the information to a standard output file and use a passing information setting job to extract this information for use in a succeeding job.

For details, see 3.1.1(1)(g) *Passing information setting job* in the manual *JP1/Automatic Job Management System 3 Overview*.

## (8) HTTP connection job

An *HTTP connection job* can send a request or receive a response via HTTP. You can use an HTTP connection job to call a web API (such as the REST API) that is provided in a cloud environment or on a web server. HTTP connection jobs can also link a JP1/AJS3-based business system with a business system on the web.

For details, see 3.1.1(1)(h) *HTTP connection job* in the manual *JP1/Automatic Job Management System 3 Overview*.

## (9) Custom event job

A *custom event job* has the functionality of both an event job and custom job. A custom event job links with a non-JP1/AJS3 program to monitor occurrence of events. In the same way as an ordinary event job, by defining a custom event job in a job flow or as a jobnet start condition, the job or jobnet can be started when a specific event occurs.

### 2.1.4 Job definition considerations

This section describes the considerations that apply when you define the content of a job.

For information about the items that can be defined, see the explanations of the Define Details dialog boxes for each job in 12. *Windows and Dialog Boxes* in the *JP1/Automatic Job Management System 3 Operator's Guide*.

## (1) Execution agent

You can specify the name of the agent that is to execute a job. In systems that use execution agent restriction, you can select execution agents from a list of execution agent profiles. An *execution agent* is a logical name representing the agent host to which the job is to be distributed. Based on the execution agent information defined in the manager host, the manager maps the execution agent specified in the job to the physical host name of the agent host, and distributes the job accordingly.

By specifying an execution agent group, you can distribute the processing load among a group of execution agents. The manager distributes jobs among the execution agents according to their assigned priorities.

You can specify an execution agent for the following job types:

- PC jobs
- Unix jobs
- Event jobs<sup>#</sup>
- Action jobs
- Custom jobs
- HTTP connection jobs
- Custom event job<sup>#</sup>

#

Event jobs and custom event jobs do not support operations that use execution agent groups. For details, see [7.6 Notes on using event jobs](#) and [7.10 Notes on using custom event jobs](#).

A queueless job requires that you specify a target host. However, you can use execution agent restriction for queueless jobs by setting the desired target host in an execution agent profile.

## (2) Execution priority

You can set the priority of the processes that start when JP1/AJS3 runs an executable file. Determine whether you need to set execution priorities for jobs. For example, you might want to prioritize the processing of a certain executable file so that it finishes earlier.

JP1/AJS3 assigns a low default execution priority when none is specified for a job. This is to prevent some jobs from monopolizing system resources and taking down the entire JP1/AJS3 system, for example when a job executed from JP1/AJS3 goes into a loop. However, jobs with a low execution priority also have a lower CPU priority. This means that while the CPU is being monopolized by processes with a high execution priority, jobs executed from JP1/AJS3 can be forced to wait for long periods until the CPU becomes available. As a result, jobs might take a long time to finish, or job processes might remain in a stopped state. If one of these job processes places a lock on a resource while waiting for the CPU to become available, this will affect the execution of other processes that are waiting for the same resource to become available.

Such problems tend to occur more markedly in system configurations that have relatively high CPU usage, such as single-processor systems and configurations where multiple processes with high execution priorities are processed at the same time. Determine whether your system environment or manner of operation requires you to raise job execution priorities.

### (3) End judgment

Consider which method to use to judge whether a process has ended successfully. You can judge the execution results of standard jobs and HTTP connection jobs. JP1/AJS3 provides five methods of end judgment:

- Always end normally
- Always end abnormally
- End normally if a specified file exists on the execution target host
- End normally if a specified file is updated while the job is running  
(Note that for a flexible job, if a specified file is updated in the time between the start and the end of the job process on the destination agent, the job is assumed to have terminated normally.)
- Set thresholds (warning threshold and abnormal threshold) and compare them with a job's return code

For judgment based on a threshold, you can evaluate whether a job ended normally, ended with a warning, or ended abnormally, by locating the job's return code relative to the threshold. The job ends normally if the return code at termination falls below the warning threshold, and ends with a warning if the return code exceeds the warning threshold but falls below the abnormal threshold. If the return code exceeds the abnormal threshold, the job ends abnormally.

Return values are interpreted as unsigned integers. For example, `-1` is handled as `4,294,967,295` in Windows, and as `255` in UNIX.

### (4) Retry on abnormal end

Determine whether to perform an automatic retry without setting a failed job in the *Ended abnormally* status if an executable file defined for the job ends abnormally. If the executable file failure was caused by a temporary error, an automatic retry corrects the job error so that operation can continue. For details, see [2.4.13 Automatic retry for abnormally ending jobs](#).

### (5) Transfer files

With a standard job, if the executable files required to execute the job are not already present in the agent host, you can transfer the files from the manager host to the agent host. If you use the `jpqjobsub` command to execute submit jobs, you can also transfer the files from the host where the `jpqjobsub` command is executed for the target agent host and execute the files there. Note that you can only transfer text files for both standard jobs and submit jobs.

Also note that flexible jobs cannot be used to transfer files.

### (6) Target service

To use a queueless job, specify **Queueless Agent** as the job's execution target service. The default is **Standard**.

Queueless jobs offer improved performance compared with normal queued jobs, and more jobs can be executed within a given period of time. For information on queueless jobs, see [11.5 Queueless jobs](#) in the manual *JP1/Automatic Job Management System 3 Overview*.

However, because queueless jobs cannot be associated with an execution agent or agent group, we recommend the use of ordinary queued jobs in most circumstances.

### (7) Timeout period

If you set a timeout period for a job, the job will be canceled when a specific length of time has passed since the job started. Decide the timeout period to impose in situations where the processing does not complete for some reason. By timing-out

job execution, you can investigate the cause of the problem, perform processing to notify the system administrator, and execute specific processing to be performed only in the event of abnormal termination.

## **(8) End delay monitoring**

By setting the time required to execute a job, you can monitor for end delays based on how much time has passed since the job started. For details, see [5. \*Monitoring Method Considerations\*](#).

## **(9) JP1 resource groups**

Consider the user access permissions you need to assign to each job. For details, see [6. \*Access Permission Considerations\*](#).

## **(10) Cautionary notes**

- Do not use a space character at the end of a comment in a job definition. All space characters after a comment are treated as invalid.
- Do not use machine-dependent characters in any part of a job definition, because these characters might not be displayed correctly.

## 2.2 Jobnet definition considerations

You can define a jobnet by grouping a set of jobs that achieve a work task, and assigning an execution order to the jobs. When constructing a jobnet, you need to consider the order of job execution, and the hierarchical structure of the jobnet to facilitate job management in JP1/AJS3.

### 2.2.1 Job execution order considerations

After you have given thought to the construction of each job, you then need to consider the execution order (job flow) of the jobs that make up each work task. A chart indicating the desired job flow and hierarchical structure can be a useful tool for defining jobs and jobnets.

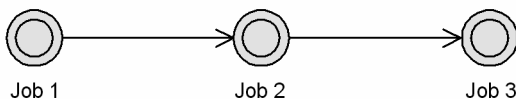
The following is an example of job flow creation.

#### (1) Work task with only one processing path

The following figure shows an example of creating a job flow for three jobs, Job 1, Job 2, and Job 3, designed to execute in a specific order. This job flow has only one processing path.

Figure 2–1: Job flow with a single processing path

■ One path only

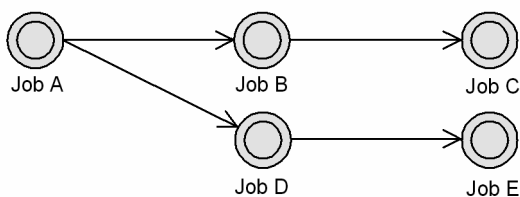


#### (2) Work task with more than one processing path

The following figure shows an example of creating a job flow with more than one processing path.

Figure 2–2: Job flow with multiple processing paths

■ Multiple paths



Here, when Job A is executed, processing diverges into two paths: Job A-Job B-Job C, and Job A-Job D-Job E.

#### (3) Work task with nested jobnets

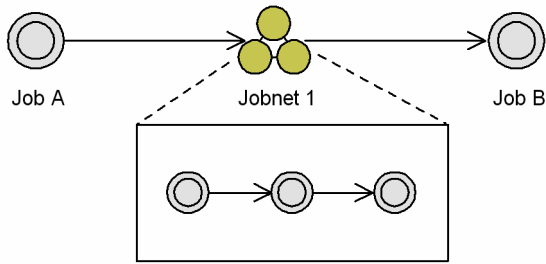
You can incorporate a jobnet into a job flow. Examples of using a nested jobnet are shown below.

##### (a) Nesting a jobnet

The following figure shows an example of incorporating a jobnet into a job flow.

Figure 2–3: Example of nested jobnet

■ Incorporate a nested jobnet into the jobnet



Job A is executed first, followed by the jobs defined in Jobnet 1 when Job A finishes. After Jobnet 1 finishes processing, Job B is executed.

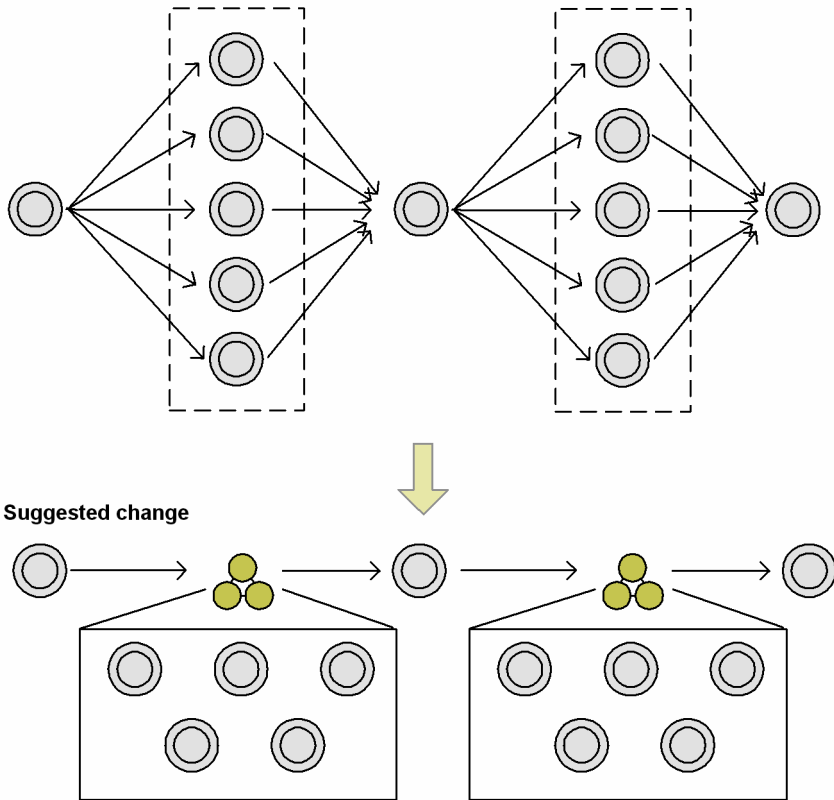
### (b) Grouping multiple jobs

It is not advisable to create jobs flows with multiple succeeding jobs. Multiple jobs brought into one jobnet are easier to administer.

The following figure shows an example of grouping multiple jobs in a nested jobnet.

Figure 2–4: Grouping multiple jobs in nested jobnet

Flow with multiple subsequent processes

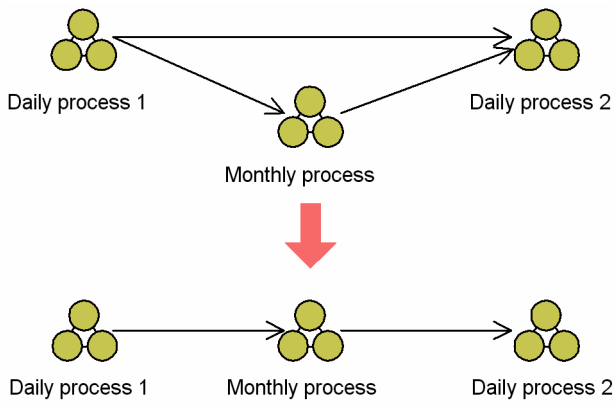


### (c) Merging two paths into one

You can use nested jobnets to merge two processing paths into one.

The following figure shows an example of merging the two paths Daily Process 1-Daily Process 2, and Daily Process 1-Monthly Process-Daily Process 2, into a single flow.

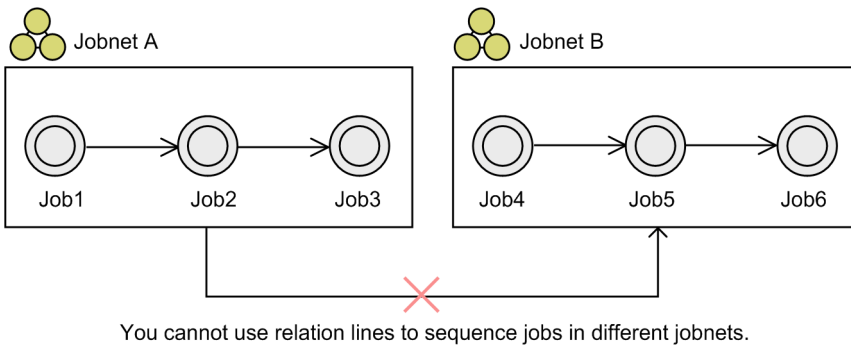
Figure 2–5: Incorporating a jobnet into a job flow



Daily Process 1 and Daily Process 2 are executed daily, and Monthly Process is executed only once a month. Because JP1/AJS3 skips jobnets that are not scheduled for execution on that day, you can incorporate all these jobnets into a single path.

#### (4) Sequencing jobs in different jobnets

In JP1/AJS3, you cannot draw relation lines to sequence jobs that reside in different jobnets.



To sequence jobs that reside in different jobnets, you can:

- Split the jobnet
- Integrate the jobnets
- Use a jobnet connector
- Use a wait condition

Each method is described below.

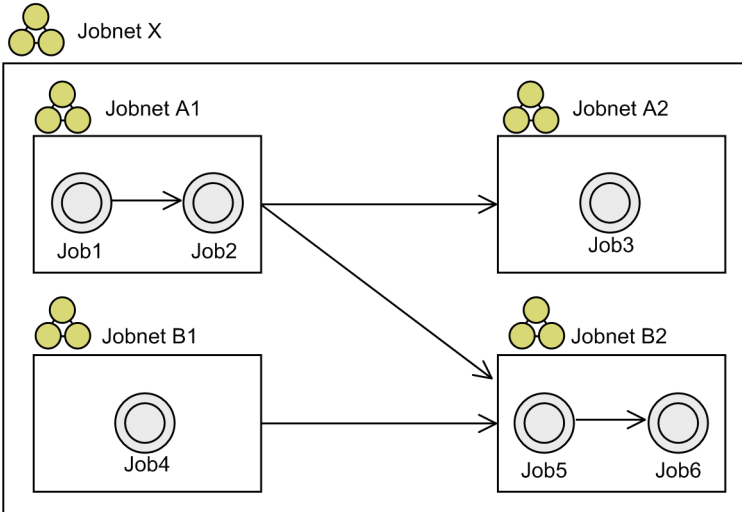
##### (a) Splitting the jobnet

The following figure shows an example of splitting the jobnet.



Figure 2–6: Splitting the jobnet

■ Split the jobnets

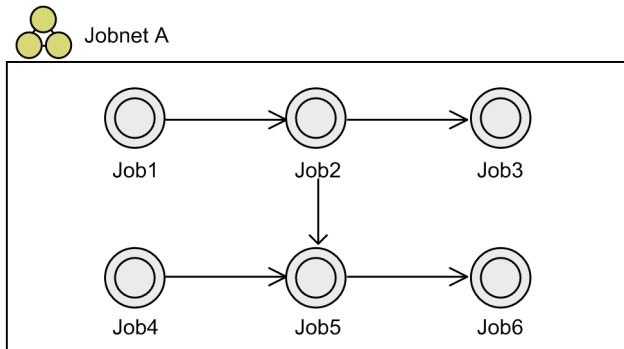


### (b) Integrating the jobnets

The following figure shows an example of integrating the jobnets.

Figure 2–7: Integrating the jobnets

■ Integrate the jobnets

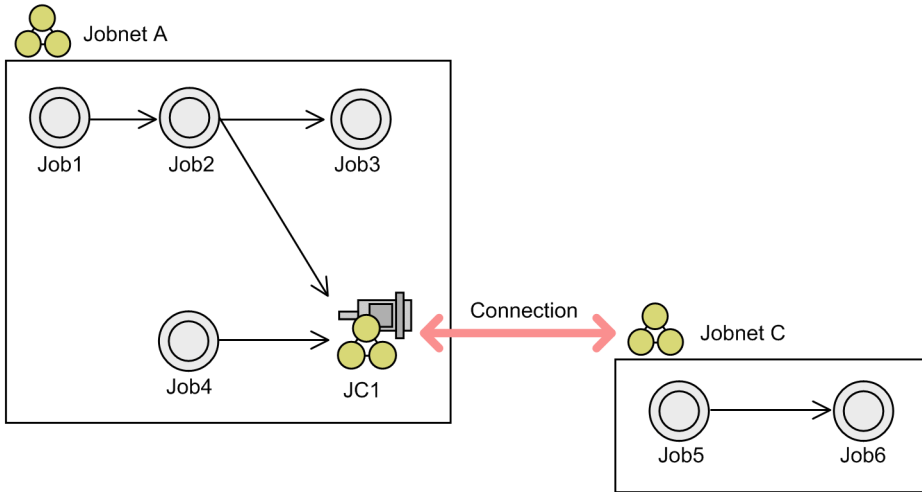


### (c) Using jobnet connectors

You can use a jobnet connector to control the execution sequence of different root jobnets. The following figure shows an example of using a jobnet connector.

Figure 2–8: Using a jobnet connector

■ Using a jobnet connector



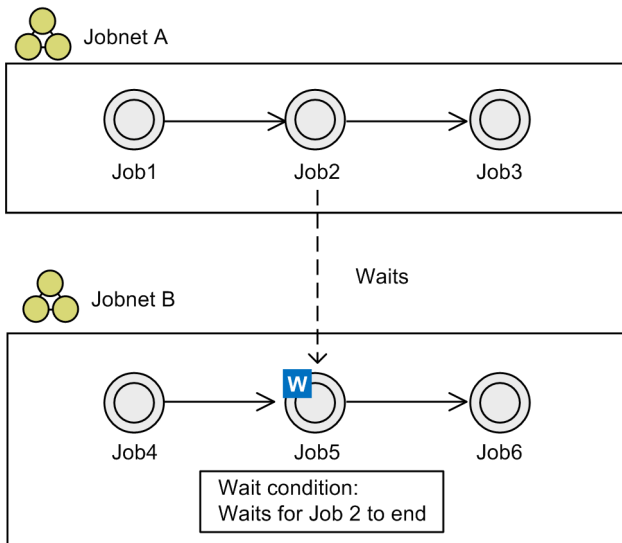
For details on jobnet connectors, see [2.2.4 Using jobnet connectors to control the order of root jobnet execution](#).

### (d) Using wait conditions

You can use wait conditions to control the execution sequence of units under different jobnets. The following figure shows an example of using a wait condition.

Figure 2–9: Using a wait condition

■ Using a wait condition



For details on wait conditions, see [2.2.5 Using wait conditions to control the order of unit execution](#).

## (5) Concepts for building jobnet hierarchies

The objectives and advantages of creating jobnet hierarchies are:

- Jobs and jobnets are easier to monitor.
- The job or jobnet location is easier to specify when changes are made.

- By assigning access permissions (JP1 resource groups) at the work task level, you can restrict access by those responsible for other work task groups.
- An appropriate hierarchical structure will help prevent poor start performance of jobnets and jobs.

Supplementary notes

- For notes on the number of jobnets and on creating jobnet hierarchies, see *7.1 Notes on the number of root jobnets registered for execution*.
- Monitoring and other operations might be difficult if there are too many levels in the hierarchy. Therefore, we recommend a hierarchy consisting of no more than two or three levels.

We recommend the following method of creating a jobnet hierarchy:

1. Establish the highest-level jobnets.

As shown in the following figure, establish the jobnet that is to occupy the highest level in each chosen category.

**Figure 2–10: Example of establishing highest-level jobnets**

Example 1: Work task categories



Example 2: Department categories



Example 3: Processing categories



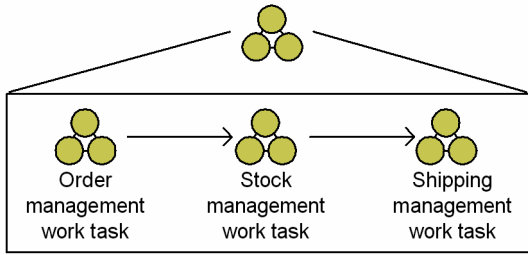
If there is no mutual order among jobnets, you can manage each of them as a highest-level jobnet.

2. If there is an order among the jobnets established in each of the chosen categories, group them into a single jobnet and assign it a hierarchy level.

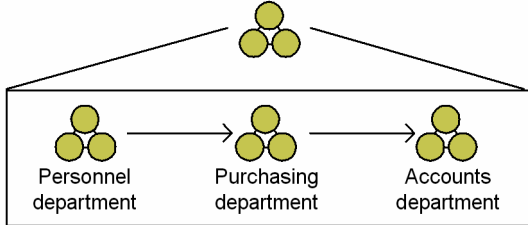
The following figure shows an example of hierarchization when there is an order among jobnets.

Figure 2–11: Hierarchization of ordered jobnets

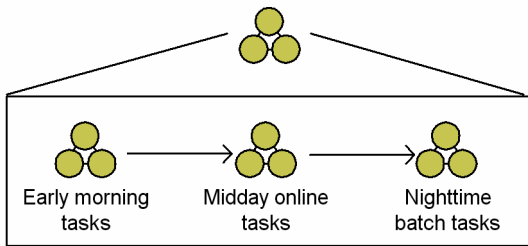
Example 1: Work task categories



Example 2: Department categories



Example 3: Processing categories



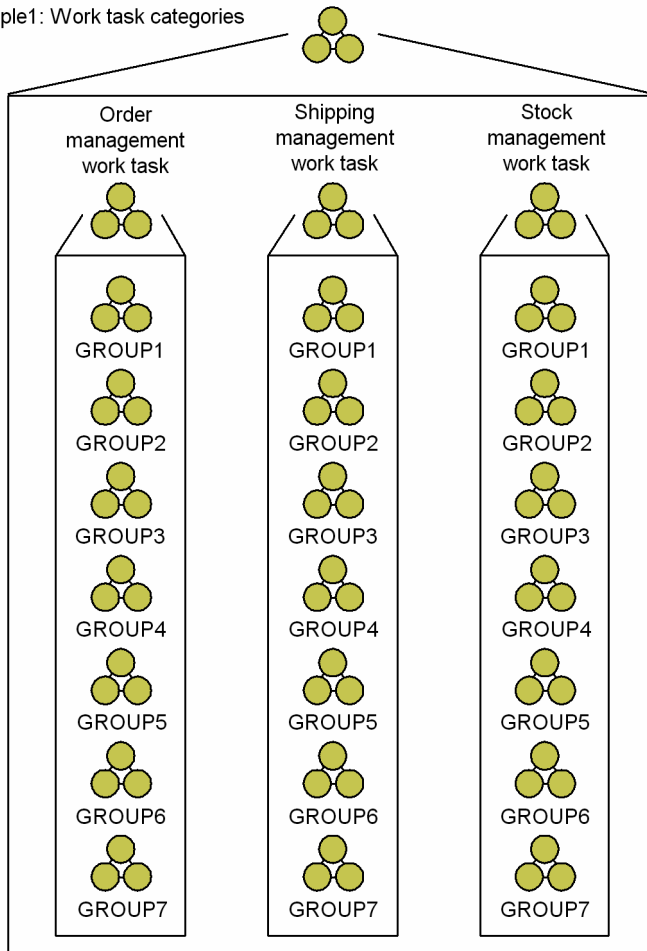
3. Establish a jobnet for each processing cycle.

If jobnets are executed in different execution cycles under the same work task, group the jobnets that have the same execution cycle.

The following figure shows an example of grouping jobnets according to their processing cycle.

Figure 2–12: Example of establishing jobnets by processing cycle

Example1: Work task categories



Decide rules like those shown in the following table for the jobnets of each processing cycle. We recommend that you include these rules as comments in the definition.

Table 2–5: Rules when jobnets are grouped by processing cycle

| Jobnets in each processing cycle | Rule  |
|----------------------------------|---|
| GROUP1                           | Group of <i>daily, weekly, monthly, six-monthly, or yearly</i> work tasks that are mutually related |
| GROUP2                           | Group of <i>daily work tasks</i> executed in isolation  |
| GROUP3                           | Group of <i>weekly work tasks</i> executed in isolation   |
| GROUP4                           | Group of <i>monthly work tasks</i> executed in isolation  |
| GROUP5                           | Group of <i>six-monthly work tasks</i> executed in isolation  |
| GROUP6                           | Group of <i>yearly work tasks</i> executed in isolation   |
| GROUP7                           | Group of <i>irregular work tasks</i> executed in isolation  |

4. Establish the lowest-level jobnets.

Next, at the lowest level of the jobnets that have been grouped according to their processing cycles, establish jobnets assigned the names listed below. We recommend that you use names consisting of one-byte alphanumeric characters. This will enable command-line execution of jobs and jobnets, and use of regular expressions with JP1/IM's automated action functionality.

xxxxxxDN  
 xxxxxxWN

xxxxxxMN  
xxxxxxHN  
xxxxxxYN  
xxxxxxRN

Legend:

- D: Indicates a jobnet executed in a "daily" cycle.
- W: Indicates a jobnet executed in a "weekly" cycle.
- M: Indicates a jobnet executed in a "monthly" cycle.
- H: Indicates a jobnet executed in a "six-monthly" cycle.
- Y: Indicates a jobnet executed in a "yearly" cycle.
- R: Indicates a jobnet executed at irregular intervals.
- N: Indicates a jobnet.

## 5. Create jobs.

Finally, create jobs assigned the following names:

xxxxxxDJ  
xxxxxxWJ  
xxxxxxMJ  
xxxxxxHJ  
xxxxxxYJ

Legend:

- D: Indicates a job executed in a "daily" cycle.
- W: Indicates a job executed in a "weekly" cycle.
- M: Indicates a job executed in a "monthly" cycle.
- H: Indicates a job executed in a "six-monthly" cycle.
- Y: Indicates a job executed in a "yearly" cycle.
- J: Indicates a job.

## 2.2.2 Corrective action when job execution fails

When you have decided the order of job execution, you must then think about the corrective action to take when a job fails to execute properly.

Consider the following:

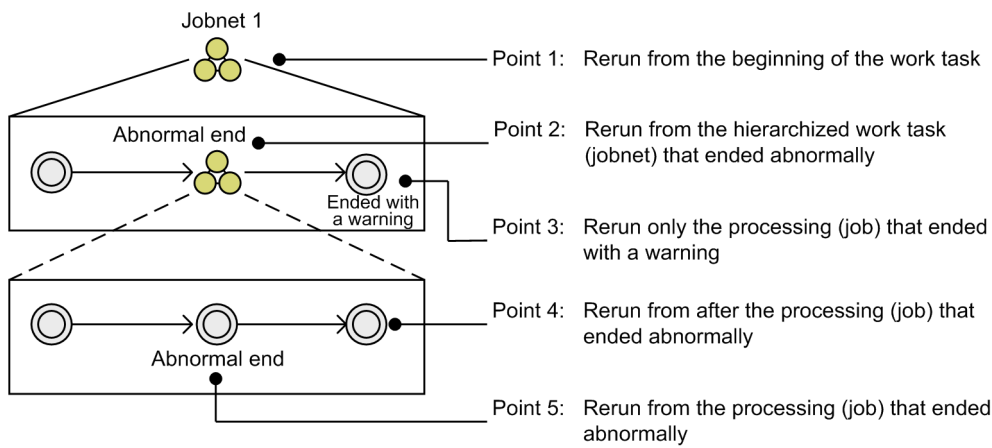
- How to execute processing automatically after a job fails
- How to run the job again

JP1/AJS3 judges whether processing has ended normally or abnormally based on the return code of the executable file. This allows processing, such as sending an email informing the system administrator of the problem, to be executed automatically when an abnormal return code is output.

If you choose to rerun a failed job, you must consider whether to re-execute all the jobs from the beginning of the job flow, or just some of them.

The following figure lists the considerations for re-executing a job.

Figure 2–13: Points from which jobs can be re-executed



Whichever method you decide on, you can put a job on hold temporarily before rerunning it. Because execution will not proceed until the "on hold" status is released, you can temporarily alter the execution schedule or suspend execution.

If you choose to re-execute only some jobs, consider whether you want to re-execute the job that ended abnormally, or re-start execution from the next job.

## 2.2.3 Jobnet definition considerations

This section describes the matters you need to consider when defining jobnets.

For details of the information you can define in a jobnet, see the description of the Define Details dialog box for a root jobnet in *12. Windows and Dialog Boxes in the JP1/Automatic Job Management System 3 Operator's Guide*.

### (1) Execution agent

You can specify the name of the agent that is to execute a jobnet. In systems that use execution agent restriction, you can select execution agents from a list of execution agent profiles. An *execution agent* is a logical name representing the agent host to which a job or jobnet is distributed. Based on the execution agent information defined in the manager host, the manager maps the execution agent specified in the job or jobnet to the physical host name of the agent host, and distributes the jobs accordingly.

By specifying an execution agent group, you can distribute the processing load among a group of execution agents. The manager distributes jobs among the execution agents according to their assigned priorities.

### (2) Concurrent execution and schedule option

For jobnets that are executed periodically, consider whether you want JP1/AJS3 to run multiple instances of the process concurrently if the process has not completed by the time the next scheduled start time arrives. For details, see *3.3.3 Concurrent execution and schedule option in the manual JP1/Automatic Job Management System 3 Overview*.

### (3) Number of logs to keep

Consider how many generations of execution logs you need to keep for the jobnet. For details, see *4.2.3 Jobnet generation management in the manual JP1/Automatic Job Management System 3 Overview*.

## (4) Timeout period

When a jobnet fails to execute by its scheduled start time, consider how long you want the jobnet to wait to execute before it times out. Jobnets that time out while waiting to execute are placed in *Skipped so not executed* status. For details, see the description of timeout periods in 3.1.1(2) *Jobnets* in the manual *JP1/Automatic Job Management System 3 Overview*.

A jobnet that is on hold is also placed in *Skipped so not exe.* status if the timeout period expires. If the timeout period expires while you are performing a temporary change operation, the jobnet is placed in *Skipped so not exe.* status. In this case, you might have to execute the jobnet again. If you do not want jobnets to be placed in *Skipped so not exe.* status, specify the timeout period with a margin, as shown below.

If you consider *one day* as the timeout period:

Specify *two days* or *unlimited*.

If you consider *two days* as the timeout period:

Specify *unlimited*.

To set **Use system settings** in the definition of the timeout period, specify any of the above values for the EXECDEFER environment setting parameter.

## (5) Jobnet monitoring

By setting the time required to execute a jobnet, you can monitor for end delays based on how much time has passed since the jobnet started. For details, see 5. *Monitoring Method Considerations*.

## (6) Execution order control

You can assign an execution order to the jobnets at the highest level of the hierarchy to facilitate job management. For details, see 2.2.4 *Using jobnet connectors to control the order of root jobnet execution*.

## (7) JP1 resource group

Consider the access permissions you need to assign to each jobnet. For details, see 6. *Access Permission Considerations*.

## (8) Cautionary notes

- Do not use a space character at the end of a comment in a jobnet definition. All space characters after a comment are treated as invalid.
- Do not use machine-dependent characters in any part of a jobnet definition, because these characters might not be displayed correctly.

## 2.2.4 Using jobnet connectors to control the order of root jobnet execution

Unlike other units such as jobs and nested jobnets, you cannot arrange root jobnet units in a order. However, you can control their execution order through a unit called a *jobnet connector*.

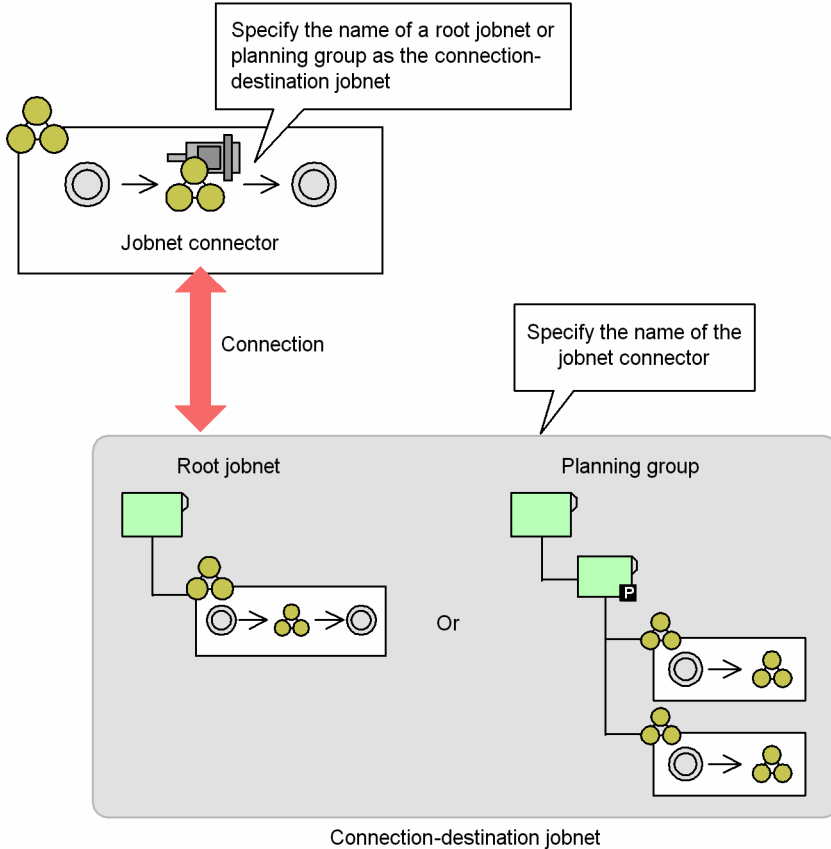


# (1) Overview of using jobnet connectors to control the root jobnet execution order

A jobnet connector links to a root jobnet and controls the order in which it executes. By forcing the root jobnet to wait for another root jobnet to end, or synchronizing its startup process with that of another root jobnet, the jobnet connector can control the execution timing of the root jobnet and its place in the execution order.

The following figure shows the relationship between the jobnet connector and the root jobnet whose execution order it controls.

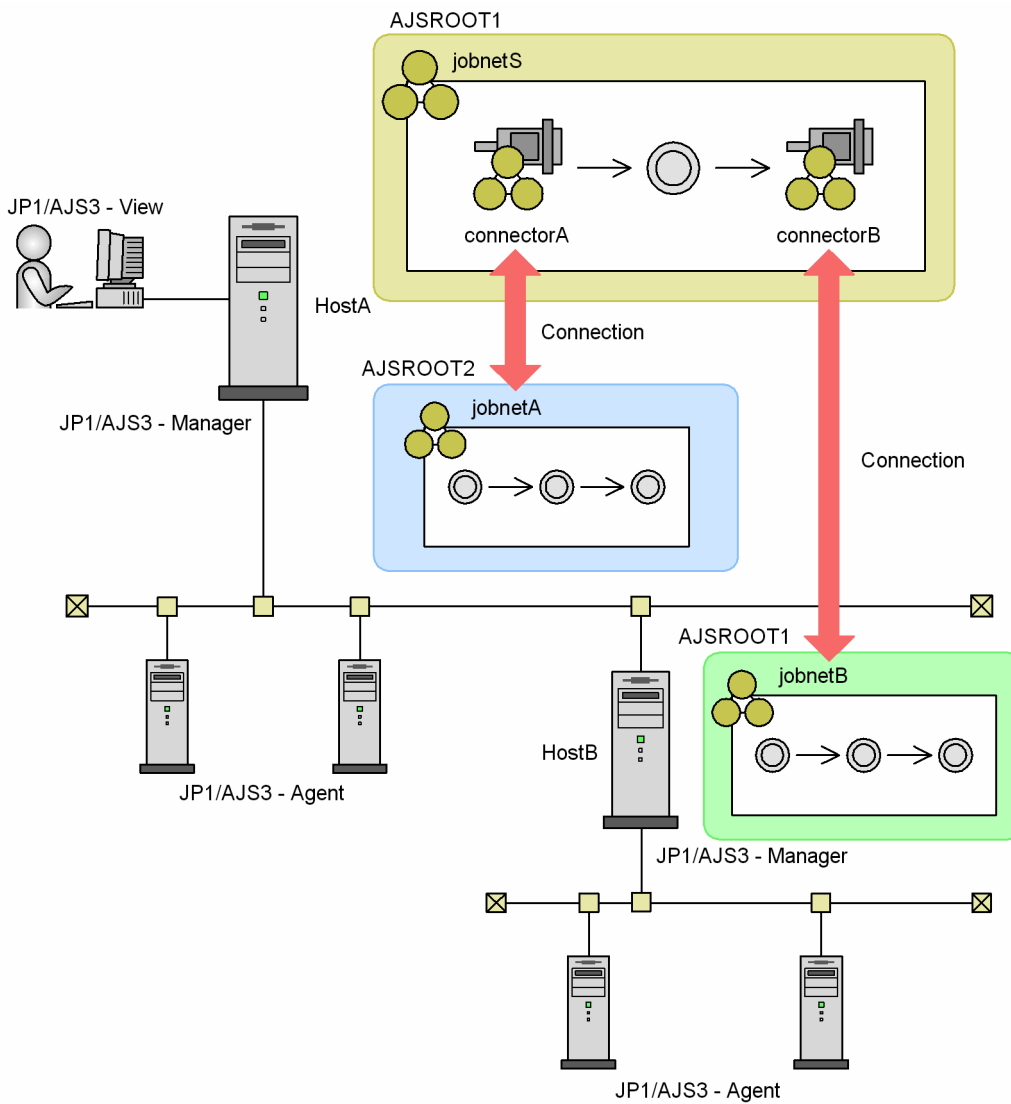
Figure 2–14: Relationship between jobnet connector and root jobnet



The jobnet connector is defined as a unit under a jobnet.

A jobnet connector can control the execution order of a root jobnet, or a root jobnet directly under a planning group. It can also control root jobnets that are managed by different scheduler services, as shown in the figure below.

Figure 2–15: Execution order control of root jobnets managed by different scheduler services



The root jobnet associated with the jobnet connector is called the *connection-destination jobnet*.

To use jobnet connectors to control the execution order of root jobnets:

1. Create the jobnets and define the jobnet connectors.
2. Define the connections between the jobnet connectors and the jobnets whose execution order they control.  
Specify the name of the connection-destination jobnet in the definition of the jobnet connector. If the jobnet whose execution order you are controlling is a root jobnet, specify a root jobnet name. If the target is a root jobnet in a planning group, specify the planning group name.  
In the definition of the connection-destination jobnet, specify the corresponding jobnet connector name.

**Supplementary note**

You can simplify the task of manually entering the definitions by using the **Save as Jobnet Connector** or **Auto-create Jobnet Connector** menu commands in JP1/AJS3 - View. However, you must manually enter the definitions if the jobnets in question are managed by different scheduler services.

3. Register the jobnets for execution.

For details on how to define jobnets and jobnet connectors, see 5. *Defining Jobnets* in the *JP1/Automatic Job Management System 3 Operator's Guide*. For details on how to register a jobnet for execution, see 7. *Executing Jobnets* in the *JP1/Automatic Job Management System 3 Operator's Guide*.

#### Cautionary notes

- You cannot use jobnet connectors with versions of JP1/AJS - View earlier than 08-10, or when connecting to a version of JP1/AJS - Manager earlier than 08-10.
- If you are using version 08-10 of JP1/AJS - View, or connecting to version 08-10 of JP1/AJS - Manager, you cannot use jobnet connectors to control the execution order of root jobnets that are under the control of different scheduler services. This capability is offered with version 08-50 or later.
- A jobnet connector must be defined under a root jobnet or nested jobnet.
- You must specify a root jobnet or planning group as the connection target jobnet for a jobnet connector.
- Do not define a jobnet connector under a connection-destination jobnet. If you do so, an error will occur when the jobnet is registered for execution.
- Do not assign a start condition to a jobnet for which a jobnet connector is defined, or to the connection-destination jobnet. If you do so, an error will occur when the jobnet is registered for execution.
- You cannot define a jobnet connector as a dependent unit or recovery unit.
- JP1/AJS3 allows you to define a jobnet connector under a dependent jobnet or recovery jobnet. However, we recommend that you avoid doing so due to the complicated relationships this type of arrangement creates.
- You cannot create a new jobnet connector or delete an existing one while execution is suspended.
- The execution order control function (jobnet connector) between scheduler services and within the same host cannot be used via NAT.
- Depending on the number of executing jobnet connectors and connection-destination jobnets that have reached their scheduled start times but are synchronized with a jobnet connector and are waiting to start, job execution performance and the performance of operations from commands, JP1/AJS3 - View, and JP1/AJS3 - Web console might be affected.

Therefore, minimize the number of executing jobnet connectors and the number of connection-destination jobnets that have reached the scheduled start time but are synchronized with a jobnet connector and are waiting to start. For example, specify times close to each other for the scheduled start time of the upper-level jobnet of the jobnet connector and the scheduled start time of the connection-destination jobnet.

Also, when operating in a situation where a large number of jobnet connectors are executing or a large number of connection-destination jobnets have reached the scheduled start time, assume the actual operation and perform tests by executing jobs or performing operations from commands, JP1/AJS3 - View, or JP1/AJS3 - Web Console, and check whether there is any problem with the performance.

## (2) Rules governing connections between jobnet connectors and connection-destination jobnets

When a jobnet with a jobnet connector and its connection-destination jobnet are registered for execution, a relationship is established between generations that share an execution date. You can control the execution order of related root jobnets, for example by having one wait for the other to finish, or having them start simultaneously.

### (a) Connection rules

Relationships are established between generations of a jobnet for which a jobnet connector is defined and those of its connection-destination jobnet, when both of the following conditions are satisfied:

- The generations share the same execution date.

- The generations are not part of a connected relationship with another generation.#

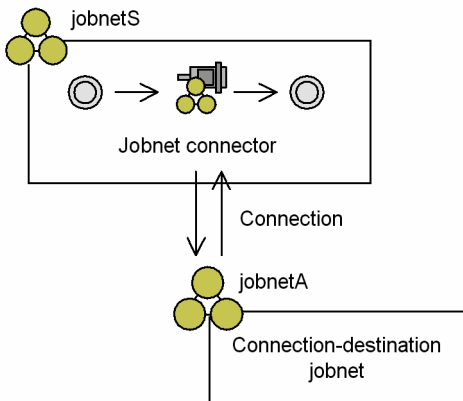
#

The relationship between the jobnets is not formalized until both have started, or one has skipped execution due to being placed in *skipped so not executed* status.

When a jobnet is scheduled for execution more than once on the same day, a relationship is established between generations with the earliest scheduled execution times among those that meet the above conditions. However, no relationship is formalized while either jobnet's schedule is still a dummy schedule.

The following figure shows an example of the connections established between a root jobnet for which a jobnet connector is defined (jobnet S) and its connection-destination jobnet (jobnet A) once both have been registered for execution.

Figure 2–16: Example of connections between jobnet connector and connection-destination jobnet



| Calendar date        | 4/17(Tue)     | 4/18(Wed)     | 4/19(Thu)     | 4/20(Fri)      |
|----------------------|---------------|---------------|---------------|----------------|
| Schedule for jobnetS | 6:00<br>@A103 | 6:00<br>@A104 | None          | Dummy schedule |
| Schedule for jobnetA | @A101<br>5:00 | None          | @A102<br>5:00 | Dummy schedule |

Red arrows indicate connections: a double-headed arrow between @A103 and @A101 on 4/17, and a single-headed arrow from @A104 to @A102 on 4/18. A red 'X' over a double-headed arrow between the dummy schedules on 4/20 indicates 'No connection'.

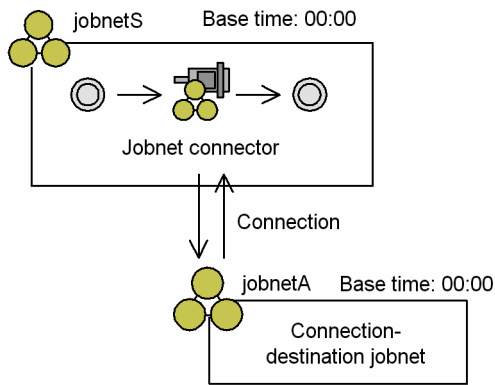
A connection is established between @A103 of jobnetS and generation @A101 of jobnetA because they share the same execution date. No connection is established between generation @A104 of jobnetS and generation @A102 of jobnetA because they are scheduled for execution on different dates. Although jobnetS and jobnetA are both tentatively scheduled for execution on 4/20 (Fri), no connection is established while the schedule is still a dummy schedule.

### (b) Using the 48-hour schedule

If you are using the 48-hour schedule to calculate schedules, connections are established between generations with the same execution date.

The following figure shows an example of the connections established under a 48-hour schedule.

Figure 2–17: Example of connections between jobnet connector and connection-destination jobnet (48-hour schedule)



|                      | 00:00                                      | 12:00      | 24:00<br>00:00                              | 36:00<br>12:00 | 48:00<br>00:00 |
|----------------------|--|------------|---|----------------|----------------|
| Calendar date        | 4/17                                       |            | 4/18  |                | 4/19           |
| Schedule for jobnetS | 4/17 under 48-hour schedule                |            |   |                |                |
|                      | @A103<br>Scheduled start time<br>4/17 7:00 |            |   |                |                |
| Schedule for jobnetA |  | Connection | @A102<br>Scheduled start time<br>4/17 25:00 |                |                |

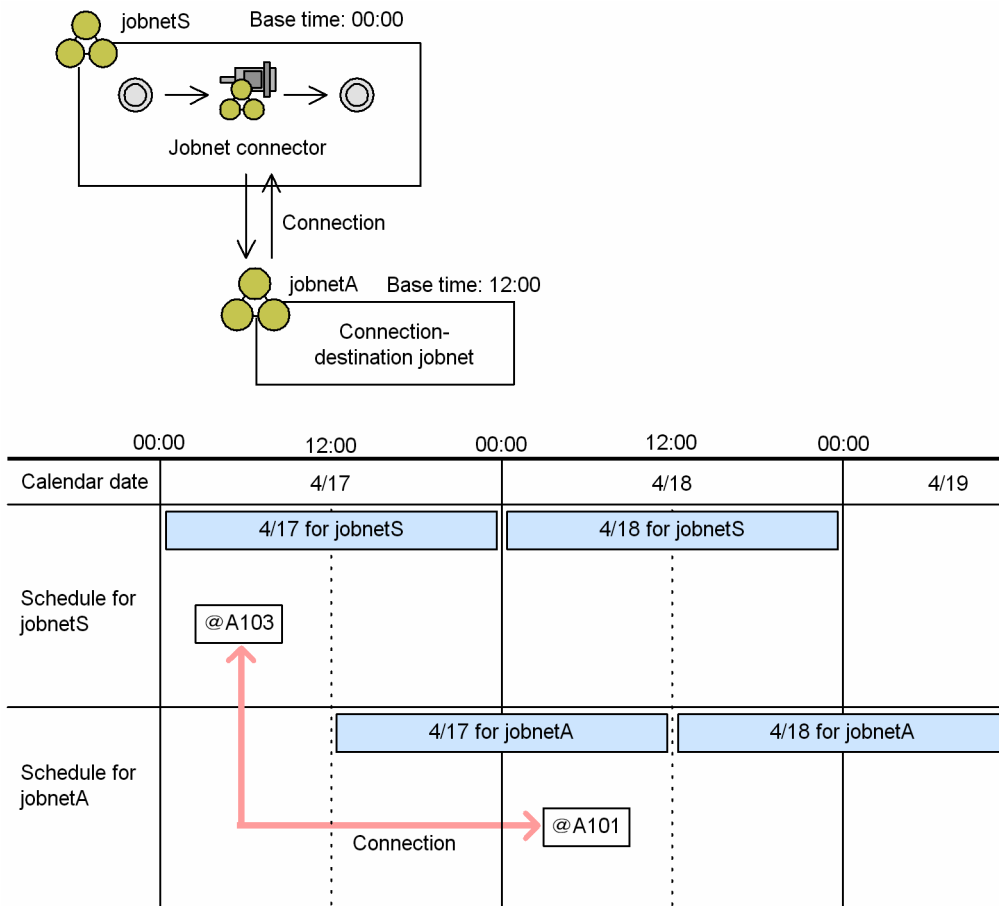
A connection is established between generation @A103 of jobnetS and generation @A102 of jobnetA, even though they are scheduled to execute on different calendar dates. This is because the two generations share an execution date under the 48-hour schedule.

**(c) Behavior when base times differ**

If a root jobnet with a jobnet connector has a different base time from its connection-destination jobnet, a connection is still established between generations that share an execution date.

The following figure shows an example of the connections established between jobnets with different base times.

Figure 2–18: Example of connections between jobnet connector and connection-destination jobnet (different base times)



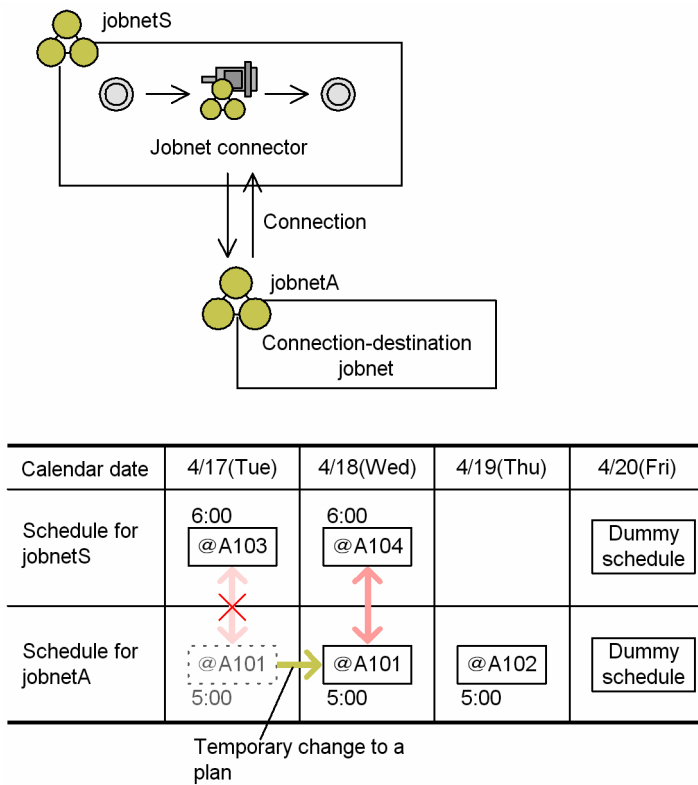
Although generation @A103 of jobnetS and generation @A102 of jobnetA are scheduled to execute on different calendar dates, a connection is established between the jobnets because both have an execution date of 4/17 with respect to their base time. In this manner, when you control the execution order of root jobnets that have different base times, connections might be established between jobnets that are scheduled to execute on different calendar dates. For this reason we recommend that you coordinate the base times of jobnet connectors and their connection-destination jobnets.

#### (d) Effects when the execution date of a generation changes

Connections between generations of a root jobnet with a jobnet connector and its connection-destination jobnet might be lost if either jobnet has its registration canceled or execution suspended before execution.

If a temporary plan change or a change to a schedule definition shifts a generation's execution date, this change will also affect the connections between generations. The following figure shows an example of how connections are affected when changes are made to a generation's scheduled execution date.

Figure 2–19: Connections when the scheduled execution date changes



Generation @A103 of jobnetS and generation @A101 of jobnetA were connected because they shared an execution date. However, when a temporary plan change shifts the scheduled execution date of @A101 back by a day to 4/18, the connection with @A103 is severed. Instead, @A101 establishes a connection with @A104, which is scheduled for execution on 4/18.

### (e) When a jobnet is scheduled for execution more than once in one day

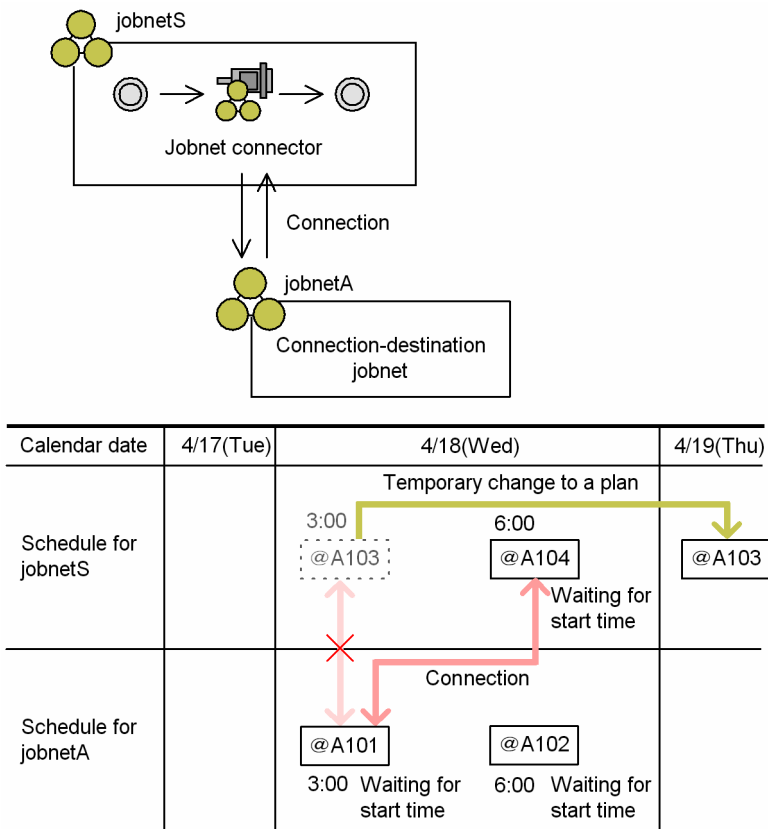
Connections are established between generations that have the same execution date, even if the jobnet is scheduled for execution more than once on a given day. However, because this can result in complicated relationships if the execution schedule of a jobnet changes, we recommend that you schedule jobnets related by a jobnet connector to execute only once per day.

#### Cautionary note

If execution is delayed for a jobnet with a jobnet connector that is scheduled to execute multiple times in one day, and *Synchronous* is specified as the method of execution order control, the next scheduled generation of the connection-destination jobnet might not be generated at all. Because the relationships of a delayed jobnet can be complicated, we recommend that you schedule jobnets with a jobnet connector to execute only once per day. For details, see (3)(b) *Synchronous execution*.

The following figure shows an example of temporarily rescheduling a jobnet that is scheduled to execute more than once per day.

Figure 2–20: Effect of temporarily rescheduling a jobnet scheduled to execute multiple times per day



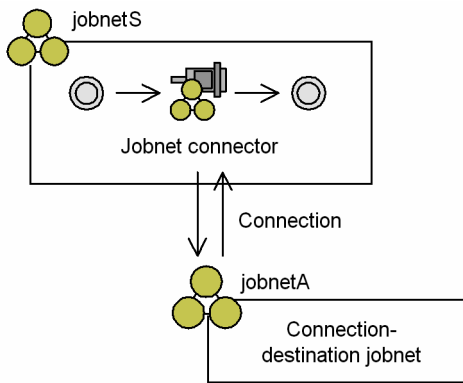
Of the generations scheduled to execute on the same day, connections are established in order, starting from those with the earliest scheduled execution times. Accordingly, generation @A103 of jobnetS establishes a connection with generation @A101 of jobnetA. At this stage, if a change to the schedule plan for jobnetS pushes the execution date of generation @A103 back to 4/19 (Thu), the connection between generation @A103 of jobnetS and generation @A101 of jobnetA is severed, and generation @A101 of jobnetA establishes a new connection with generation @A104 of jobnetS.

If you use JP1/AJS3 - View or the `ajsshow` command to simulate the connections between jobnets scheduled for execution more than once per day, the simulation will show multiple connections for unexecuted generations.

The following figure shows an example of such a situation. This simulation assumes that all generations are in *Wait for start time* status.



Figure 2–21: Example of connection simulation for jobnets scheduled to execute multiple times per day



|                      | 4/17 (Tue) | 4/18 (Wed)                              |   | 4/19 (Thu) |
|----------------------|------------|---|---|------------|
| Schedule for jobnetS |            | 3:00<br>Waiting for start time<br>@A103 | 6:00<br>Waiting for start time<br>@A104 |            |
| Schedule for jobnetA |            | 3:00<br>Waiting for start time<br>@A101 | 6:00<br>Waiting for start time<br>@A102 |            |

Legend:

- : Simulation of jobnetS
- : Simulation of jobnetA

In the simulation, because jobnetS and jobnetA share an execution date, generation @A103 of jobnetS connects to the generation of jobnetA with the earliest scheduled execution time, in this case @A101. However, because the connection between generation @A103 of jobnetS and generation @A101 of jobnetA remains unfixed, the simulation also connects generation @A101 to generation @A104 of jobnetS. The simulation similarly connects generation @A101 of jobnetA to generation @A103 of jobnetS, and generation @A102 of jobnetA to generation @A103 of jobnetS.

### (f) Behavior when time zones differ

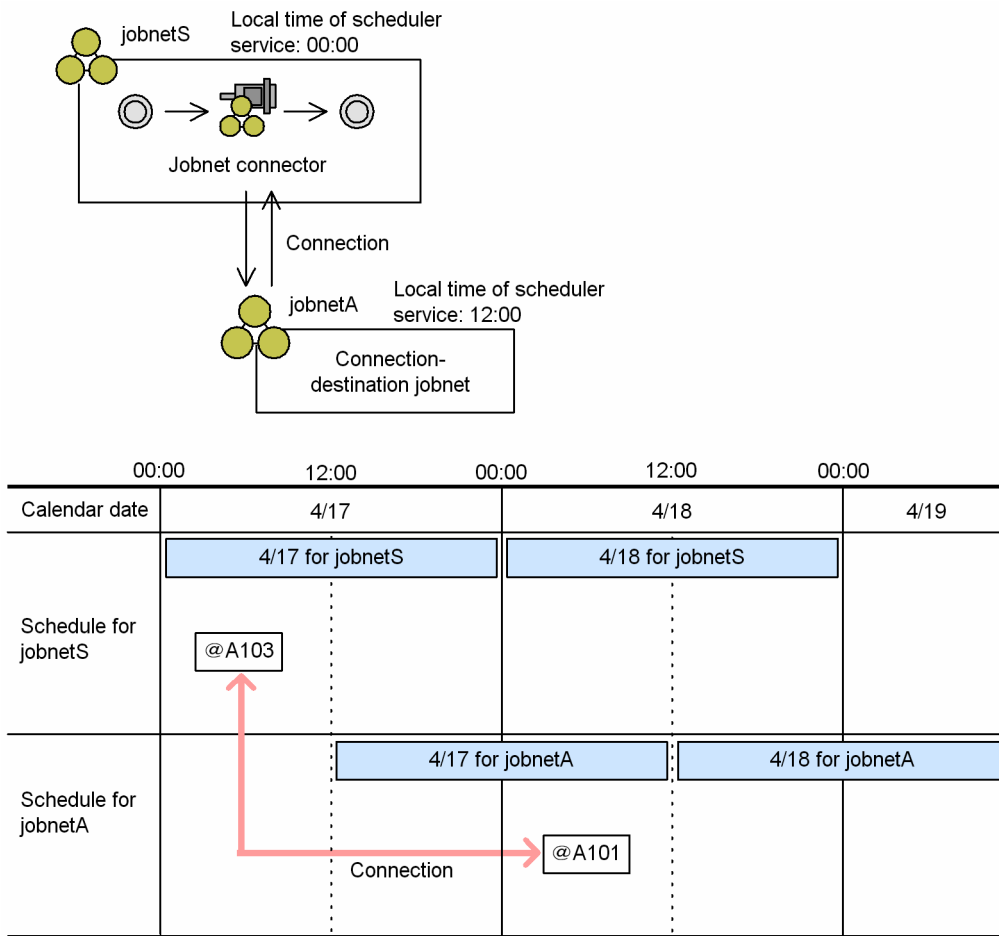
When a root jobnet with a jobnet connector and its connection-destination jobnet are scheduled under different time zones, connections will still be established between generations if their execution dates are the same. However, for the sake of simplicity, we recommend that jobnet connectors and their connection-destination jobnets use the same time zone.

### (g) Behavior when scheduler services have different local times

When the scheduler service governing a root jobnet with a jobnet connector uses a different local time setting from the scheduler service of the connection-destination jobnet, connections will still be established between generations if their execution dates are the same.

The following figure shows an example of the connections established between jobnets when their scheduler services use different local time settings.

Figure 2–22: Example of connections between jobnet connector and connection-destination jobnet (when scheduler services have different local time settings)



Although generation @A103 of jobnetS and generation @A101 of jobnetA are scheduled to execute on different calendar days, a connection is established between them because both are scheduled for execution on 4/17 at their respective local times. To avoid a situation in which connections are established between generations scheduled for execution on different days, ensure that the scheduler services responsible for the connection-destination jobnet and the jobnet connector are set to the same local time.

### (3) Methods for controlling the execution order of root jobnets

The method of execution order control governs whether execution of the root jobnet is synchronized with execution of the jobnet connector. Specify the control method in the settings of the connection-destination jobnet. The default is **Asynchro**.

The following describes the behavior at root jobnet execution under each control method.

#### (a) Asynchronous execution

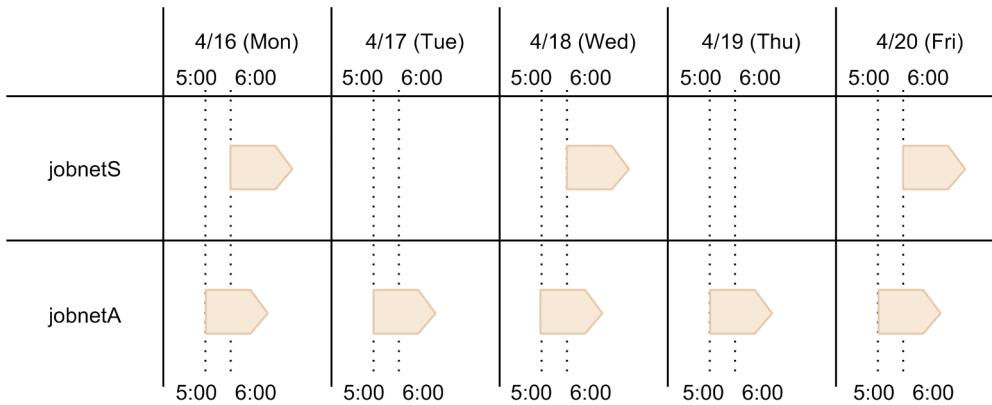
If **Asynchro** is set as the method of execution order control, the connection-destination jobnet executes according to its own schedule regardless of any connections with other jobnets and the status of the jobnet connector.

The following figure shows how the respective jobnets are executed under the asynchronous method of execution order control. Here, jobnetS is the jobnet with the jobnet connector, and jobnetA is the connection-destination jobnet.

Figure 2–23: Operation under asynchronous execution order control

|                      | 4/16 (Mon)    | 4/17 (Tue)    | 4/18 (Wed)    | 4/19 (Thu)    | 4/20 (Fri)    |
|----------------------|---------------|---------------|---------------|---------------|---------------|
| Schedule for jobnetS | 6:00<br>@A106 |               | 6:00<br>@A107 |               | 6:00<br>@A108 |
| Schedule for jobnetA | @A101<br>5:00 | @A102<br>5:00 | @A103<br>5:00 | @A104<br>5:00 | @A105<br>5:00 |

■ Execution schedule when *Asyncho* is specified



Cautionary note

Even when multi-schedule is set as the schedule option, execution will not start at the subsequent execution start time if the previous execution schedule has not started yet. If multi-schedule is set both on the root jobnet for which the jobnet connector is defined and on the connected jobnet, make sure that the previous execution schedule starts before the subsequent execution start time arrives.

**(b) Synchronous execution**

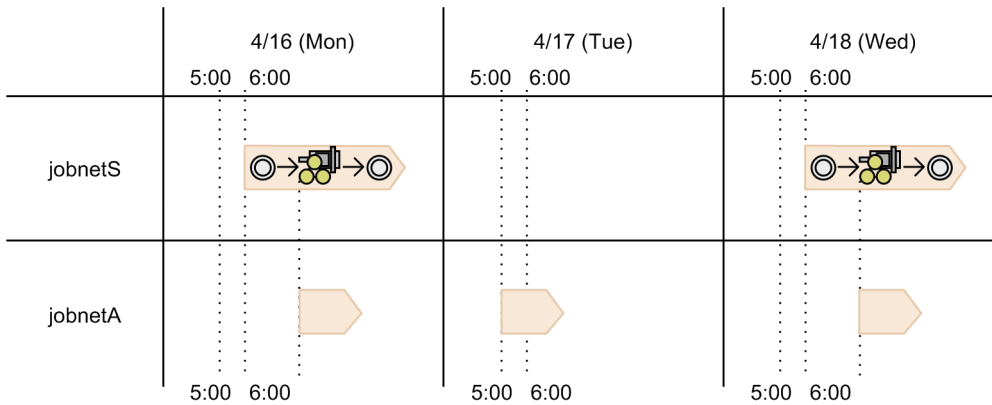
If **Syncho** is set as the method of execution order control, generations in a connected relationship wait until the jobnet connector has executed before starting. In the absence of a corresponding generation with the same execution date, a generation will execute according to its own schedule.

The following figure shows how the respective jobnets are executed under the synchronous method of execution order control. Here, jobnetS is the jobnet with the jobnet connector, and jobnetA is the connection-destination jobnet.

Figure 2–24: Operation under synchronous execution order control

|                      | 4/16 (Mon)    | 4/17 (Tue)    | 4/18 (Wed)    |
|----------------------|---------------|---------------|---------------|
| Schedule for jobnetS | 6:00<br>@A106 |               | 6:00<br>@A107 |
| Schedule for jobnetA | @A101<br>5:00 | @A102<br>5:00 | @A103<br>5:00 |

■ Execution schedule when *Synchro* is specified



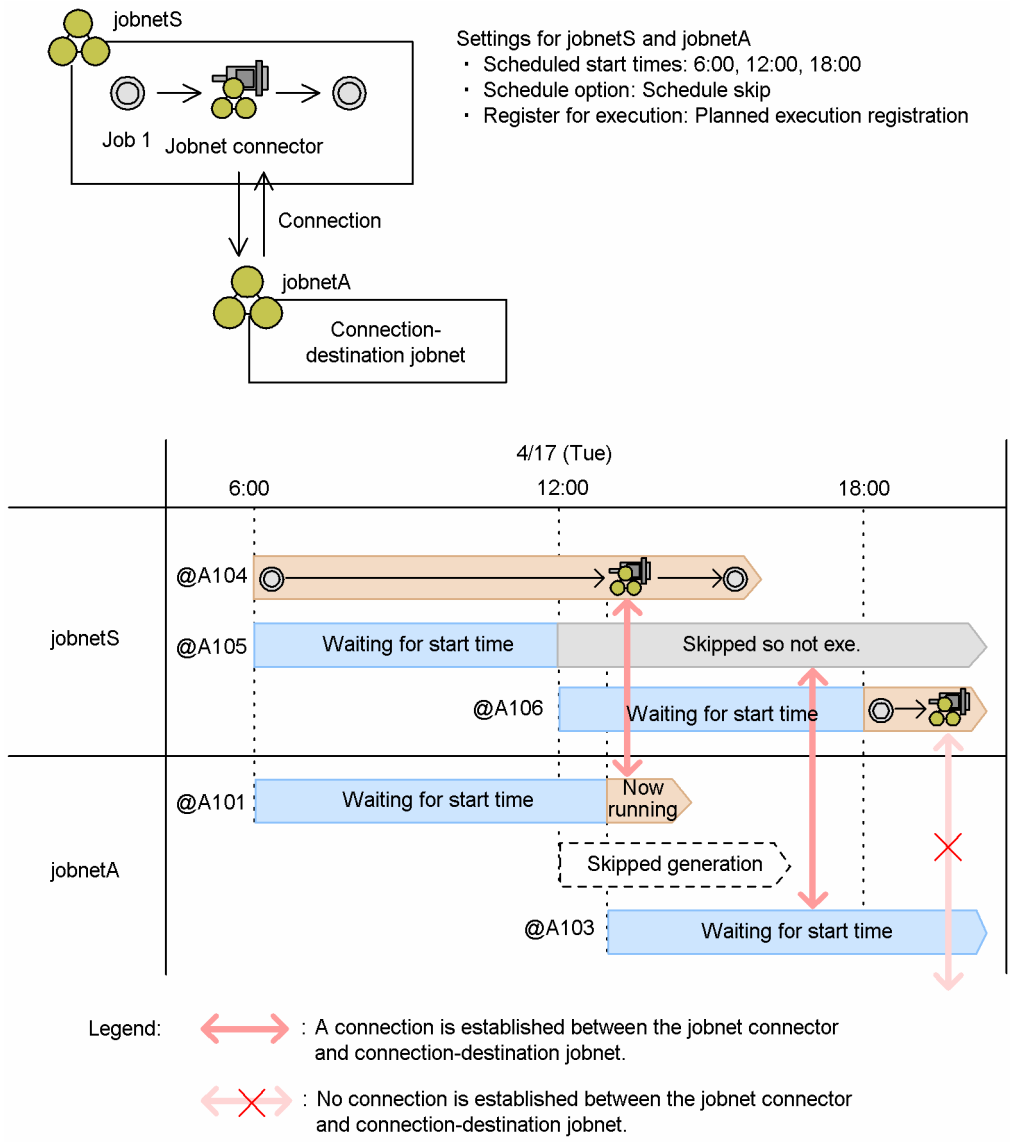
JobnetS and jobnetA have scheduled start times of 6:00 and 5:00, respectively. The connected relationship between the generations that are scheduled for execution on 4/16 (Mon) and 4/18 (Wed) means that on these days, jobnetA starts in synchronization with the jobnet connector, instead of at 5:00 as scheduled. The generation of jobnetA scheduled for execution on 4/17 (Tue), which lacks a connection to a generation of jobnetS, starts at 5:00 as originally scheduled.

The synchronous execution setting applies only to the first execution, not to any subsequent re-executions. To apply synchronous execution to re-executed generations, you can use the function for temporarily changing the method of execution order control.

Cautionary notes

- Even when multi-schedule is set as the schedule option, execution will not start at the subsequent execution start time if the previous execution schedule has not started yet. If multi-schedule is set both on the root jobnet for which the jobnet connector is defined and on the connected jobnet, make sure that the previous execution schedule starts before the subsequent execution start time arrives.
- If execution is delayed for a jobnet with a jobnet connector that is scheduled to execute multiple times per day, and **Synchro** is specified as the method of execution order control, the next scheduled generation of the connection-destination jobnet might not be generated at all. Because the relationships of a delayed jobnet can be complicated, we recommend that you schedule jobnets with a jobnet connector to execute only once per day. The following figure shows an example in which jobnets are executed more than once per day with the execution order control method set to **Synchro**.

Figure 2–25: Example of executing jobnets multiple times per day based on synchronous execution order control



In this example, the scheduled execution start times of the root jobnet for which the jobnet connector is defined (jobnetS) and the connection-destination jobnet (jobnetA) are 6:00, 12:00, and 18:00. Schedule skip is set as the schedule option, and planned execution registration is used as the execution registration method.

The following describes the behavior of jobnetS and jobnetA with respect to generation @A104 of jobnetS (6:00 start time), when the preceding "Job 1" of the jobnet connector has not completed by the time jobnetS is next scheduled for execution (12:00).

1. Generation @A105 of jobnetS scheduled for execution at 12:00 enters *Skipped so not executed* status.
 

Because *schedule skip* is set as the schedule option for jobnetS, generations whose execution overlaps with that of an earlier scheduled generation enter *Skipped so not executed* status and their execution is skipped.

Because generation @A101 has not started by 12:00, the generation of jobnetA scheduled for execution at that time is removed from the dummy schedule.
2. After Job 1 has finished executing, the jobnet connector associated with generation @A104 of jobnetS is executed.
 

Because synchronous execution is specified, generation @A101 of jobnetA also begins executing. At the same time, JP1/AJS3 creates the next scheduled generation of jobnetA (scheduled for 18:00) in *Wait for start time* status. This generation is assigned the execution ID @A103. At this point, a connection is established

between generation @A105 of jobnetS, which was placed in *Skipped so not executed* status, and generation @A103 of jobnetA. Because generation @A103 of jobnetA is connected to a jobnet in *Skipped so not executed* status, it remains in *Wait for start time* status and is not executed. For an explanation of the status transitions undergone by the jobnet connector and connection-destination jobnet, see (4) *Transitions of jobnet connector and connection-destination jobnet statuses*.

3. At 18:00, the generation of jobnetS scheduled for execution at that time (@A106) is executed.

At this stage, generation @A106 of jobnetS has no connection to any other generation. Hence, generation @A106 of jobnetS waits in *Now running* status until generation @A103 of jobnetA has finished and it is clear whether any generations of jobnetA can serve as a connection target.

If you change to asynchronous execution by using the function for temporarily changing the execution order control method, generation @A106 will be executed regardless of whether any connected generations exist. For details on this feature, see (c) *Temporarily changing the method of execution order control*.

If generation @A105 of jobnetS is re-executed from its *Skipped so not executed* status before 18:00, the connected generation @A103 of jobnetA is also executed. If a new generation of jobnetA is created after generation @A103 has finished, a connection is established between generation @A106 of jobnetS and this new generation, and generation @A106 is executed. If no new generation is created, the jobnet connector associated with generation @A106 of jobnetS enters *Bypassed* status.

### (c) Temporarily changing the method of execution order control

JP1/AJS3 provides a feature for temporarily changing the method of execution order control. You can use this feature to synchronize re-execution of a connection-destination jobnet with a jobnet connector, or to apply a different method of execution order control to a particular generation.

To access this feature, choose **Operations** and then **Change Execution Order Method** in the following windows:

- Daily Schedule (Hierarchy) window
- Monthly Schedule window
- Jobnet Monitor window

Supplementary note

You cannot change the method of execution order control for a dummy schedule.

## (4) Transitions of jobnet connector and connection-destination jobnet statuses

This section describes the status transitions that apply to jobnet connectors and their connection-destination jobnets. For details on the statuses that these units can acquire, see 6.1 *Status levels of jobnets, jobs, and jobnet connectors* in the manual *JP1/Automatic Job Management System 3 Overview*.

### (a) Jobnet connector status transitions

When you register a jobnet with a jobnet connector for execution, the root jobnet associated with the jobnet connector begins executing. At the point when all preceding units in the jobnet have ended, the status of the jobnet connector changes according to the status of the connection-destination jobnet.

The following table lists the status transitions that apply to jobnet connectors.

Table 2–6: Jobnet connector status transitions

| Status of connection-destination jobnet | Status of jobnet connector |
|---|----------------------------|
| Not registered                          | Running + Abend            |
| (No generations)#1                      | Bypassed                   |
| Not sched. to exe.                      | Bypassed                   |
| Wait for start time                     | Now running                |
| Being held                              | Now running                |
| Skipped so not exe.                     | Running + Abend#2          |
| Ended normally                          | Ended normally             |
| Ended with warning                      | Ended with warning         |
| Ended abnormally                        | Running + Abend#2          |
| Interrupted                             | Running + Abend#2          |
| Killed                                  | Running + Abend#2          |
| Shutdown                                | Running + Abend            |
| Invalid exe. seq.                       | Running + Abend#2          |
| Now running                             | Now running                |
| Running + Warning                       | Running + Warning          |
| Running + Abend                         | Running + Abend            |

#1

The connection-destination jobnet is registered for execution, but there are no generations that connect to the jobnet connector. For details on the connection rules, see (2) *Rules governing connections between jobnet connectors and connection-destination jobnets*.

#2

If, while the status of the jobnet connector is *Running + Abend*, a generation of the connection-destination jobnet that is in the end status is deleted by managing saved generations or by cancelling registration, the status of the jobnet connector changes to *Ended abnormally*.

The jobnet connector immediately enters *Ended abnormally* status if the definition of the jobnet connector is invalid. Possible reasons are as follows:

- No connection-destination unit is specified.
- A non-existent unit is specified as the connection-destination jobnet.
- The unit specified as the connection-destination jobnet is neither a root jobnet nor a planning group.
- Execution order control is disabled in the definition of the unit specified as the connection-destination jobnet.
- The **Jobnet Connector** is specified incorrectly in the definition of the unit specified as the connection-destination jobnet.
- The **Connection range** setting of the jobnet connector is different from that of the connection-destination jobnet.
- One of the following problems renders the definition invalid when **Other service** is specified for **Connection range**.
  - JP1/AJS3 cannot connect to the host specified in **Connection host**.

- The scheduler service specified in **Connection service** does not exist.
- The host specified in **Connection host** for the connection-destination jobnet is different from that of the jobnet connector.
- The scheduler service specified in **Connection service** for the connection-destination jobnet is different from that of the jobnet connector.

## (b) Connection-destination jobnet status transitions

When a connection-destination jobnet is registered for execution, its behavior at execution changes according to the status of connected generations, as well as which execution order control method (synchronous or asynchronous) is in effect.

The following table lists the status transitions that apply to connection-destination jobnets.

Table 2–7: Connection-destination jobnet status transitions

| Status of jobnet connector     | Status of connection-destination jobnet |              |
|--------------------------------|---|--------------|
|                                | Synchronous                             | Asynchronous |
| Not registered                 | Wait for start time                     | Now running  |
| (No generations) <sup>#1</sup> | Now running                             | Now running  |
| Not sched. To exe.             | Wait for start time                     | Now running  |
| Wait for prev. to end          | Wait for start time                     | Now running  |
| Not executed + Ended           | Wait for start time <sup>#2</sup>       | Now running  |
| Bypassed                       | Now running                             | Now running  |
| Now running                    | Now running                             | Now running  |
| Ended normally                 | Wait for start time <sup>#2</sup>       | Now running  |
| Ended with warning             | Wait for start time <sup>#2</sup>       | Now running  |
| Ended abnormally               | Wait for start time <sup>#2</sup>       | Now running  |
| Killed                         | Wait for start time <sup>#2</sup>       | Now running  |
| Shutdown                       | Wait for start time <sup>#2</sup>       | Now running  |
| Unknown end status             | Wait for start time <sup>#2</sup>       | Now running  |

#1

The jobnet connector is registered for execution, but there are no generations that connect to connection-destination jobnets. For details on the connection rules, see (2) *Rules governing connections between jobnet connectors and connection-destination jobnets*.

#2

If the jobnet connector has already terminated as a result of user intervention by the time the connection-destination jobnet starts, the connection-destination jobnet will stay in *Wait for start time* status until the jobnet connector is executed. You can resolve this issue by re-executing the jobnet connector, or by temporarily changing the method of execution order control for the connection-destination jobnet to *asynchronous*.

The connection-destination jobnet immediately enters *Ended abnormally* status if the definition of the jobnet contains an error, such as one of those listed below. However, if the jobnet is subject to asynchronous control and **Other service** is specified for **Connection range**, the jobnet executes as normal without entering *Ended abnormally* status even if the



jobnet definition contains an error. For this reason, we recommend that you use the definition pre-check function to check the definition of the connection-destination jobnet before you register it for execution.

- Execution order control is enabled, but no jobnet connector name is specified.
- A non-existent unit is specified as the jobnet connector.
- The unit specified as the jobnet connector is not a jobnet connector.
- The specified jobnet connector is associated with a different connection target.
- The **Connection range** setting of the jobnet connector is different from that of the connection-destination jobnet.
- One of the following problems renders the definition invalid when **Other service** is specified for **Connection range**:
  - JP1/AJS3 cannot connect to the host specified in **Connection host**.
  - The scheduler service specified in **Connection service** does not exist.
  - The host specified in **Connection host** for the jobnet connector is different from that of the connection-destination jobnet.
  - The scheduler service specified in **Connection service** for the jobnet connector is different from that of the connection-destination jobnet.

### (c) When communication between scheduler services fails

When the execution order of root jobnets governed by different scheduler services is controlled, communication between the respective scheduler services takes place at the following times:

- When the jobnet connector starts executing
- When the connection-destination jobnet starts executing
- When the status of the connection-destination jobnet changes

This subsection describes how the statuses of the jobnet connector and the connection-destination jobnet are affected when a communication error<sup>#</sup> occurs at each of these stages. For details about how to troubleshoot errors, see *2.6.5 Troubleshooting problems related to jobnet connectors* in the manual *JP1/Automatic Job Management System 3 Troubleshooting*.

#### ■ If a communication error occurs when the jobnet connector starts executing

When the jobnet connector starts executing, connections are established with generations of the connection-destination jobnet. For information about this process, see *(2) Rules governing connections between jobnet connectors and connection-destination jobnets*.

If a communication error occurs during this process, the jobnet connector enters *Ended abnormally* status. The status of the connection-destination jobnet remains unchanged.

#### ■ If a communication error occurs when the connection-destination jobnet starts executing

When synchronous control is used for the connection-destination jobnet, connections are established with generations of the jobnet connector when the connection-destination jobnet starts executing. For information about this process, see *(2) Rules governing connections between jobnet connectors and connection-destination jobnets*.

If a communication error occurs during this process, the connection-destination jobnet enters *Ended abnormally* status. The status of the jobnet connector remains unchanged.

#### ■ If a communication error occurs when the status of the connection-destination jobnet changes

The jobnet connector remains unaware of the change to the connection-destination jobnet's status. Consequently, the jobnet connector does not undergo a corresponding status change.

#

A communication error occurs even when communication processing is performed for a stopped scheduler service. For example, a communication error occurs if there is linkage between different scheduler services in the same host, the JP1/AJS service is started by a hot start and this communication processing is performed before one of the scheduler services starts.

## (5) Re-execution of jobnet connectors and connection-destination jobnets

The following describes the re-execution of jobnet connectors and connection-destination jobnets.

### (a) Re-executing jobnet connectors

In the same manner as with a job, you can re-execute a jobnet connector only after it has finished executing. Note that because a jobnet connector does not take a hold attribute, you cannot have it immediately placed in *Held* status upon re-execution.

### (b) Re-executing connection-destination jobnets

A connection-destination jobnet can be re-executed in the same manner as a standard jobnet. However, after a jobnet controlled by the synchronous method has been executed once, subsequent executions run according to the asynchronous method. Consequently, such a connection-destination jobnet will not synchronize with the jobnet connector at re-execution. You can synchronize re-execution with a jobnet connector by changing the execution order control method to synchronous by using the feature provided for this purpose.

### (c) Effect on re-execution when connections between generations are severed

A generation that is in a connected relationship with another generation might be removed from the schedule for some reason. This might be due to a cancellation of registration, execution cancellation, or changes to the number of logs to keep. In this case, you will be unable to re-execute the generation that was in the connected relationship with the deleted generation. The following table shows how JP1/AJS3 behaves when one of a pair of connected generations is deleted, and describes what action the user must take.

Table 2–8: Handling re-execution when a connected relationship is severed

| Generation  | Behavior at re-execution   | Action to take   |
|---|--|--|
| The remaining generation is the jobnet connector              | Upon re-execution, the jobnet connector enters <i>Ended abnormally</i> <sup>#</sup> status.  | Resume the job flow from the unit succeeding the jobnet connector.   |
| The remaining generation is the connection-destination jobnet | When you re-execute the connection-destination jobnet after changing the execution order control method to synchronous, the jobnet enters <i>Ended abnormally</i> <sup>#</sup> status. | Temporarily change the execution order control method to asynchronous, and re-execute the connection-destination jobnet. |

#

The jobnet connector or connection-destination jobnet will enter *Ended abnormally* status only if you re-execute it after it has already finished. You will be able to re-execute the unit without it entering *Ended abnormally* status if you do so while it is still running.

## (6) Jobnet connector polling

When you control the execution order of root jobnets governed by different scheduler services, jobnet connectors and connection-destination jobnets under synchronous control check the status of their counterparts by regular polling. This polling takes place during the following periods:

- The time from when the jobnet connector starts until the jobnet connector has ended
- The time from when the connection-destination jobnet reaches its start time until it starts

The jobnet connector or connection-destination jobnet will end abnormally if polling fails for some reason, such as an interruption to the scheduler service or a communication error, and it cannot confirm the status of its counterpart. Also, for example, there is linkage between different scheduler services in the same host, the JP1/AJS service is started by a hot start, polling is performed before one of the scheduler services starts, and a jobnet connector or connection-destination jobnet ends abnormally.

## 2.2.5 Using wait conditions to control the order of unit execution

You can use a *wait condition* to control the execution sequence of units in different jobnets. A unit assigned a wait condition does not start to execute until a specified unit has finished.

To use wait conditions, you need to configure the `PREWAITUSE` environment setting parameter in advance to enable the use of wait conditions. If you want to expand the types of units that can be waited for, configure the `PREWAITEXTEND` environment setting parameter as necessary.

### (1) Overview of using wait conditions to control the order of unit execution

The following gives an overview of execution sequence control using wait conditions, and describes the flow of using wait conditions to control the execution sequence among units.

#### (a) Units with wait conditions and units whose ends are being waited for

When assigning a wait condition, you specify a target unit. The unit with the wait condition waits for the target unit to finish executing. The status of the target unit (that is, whether it has finished executing) depends on whether the unit has a start condition<sup>#</sup>.

For units without a start condition<sup>#</sup>

You can assume that the unit whose end is being waited for has finished executing when its status is one of the following:

- Ended normally
- Ended with warning
- Bypassed

For units with a start condition<sup>#</sup>

If the default settings are in effect, you can assume that the unit whose end is being waited for has finished executing when the statuses of the monitoring generation and all the execution generations of the unit are as follows:

Monitoring generation

- Monitor-end normal

Execution generation

- Ended normally
- Ended with warning

By setting a start condition for the unit whose end is being waited for, you can change the status that the unit needs to have when it finishes executing. For details, see *(5)(c) Behavior of units with wait conditions when start conditions are used for the units whose ends are being waited for*.

#

To be able to use a start condition, the root jobnet must satisfy the following criteria:

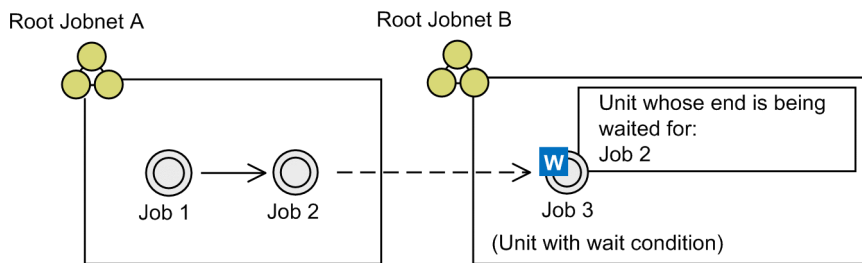
- A start condition is set for the root jobnet.
- When the root jobnet is registered for execution, the use of a start condition is specified.  
For immediate execution, in the Register for Execution dialog box, in the **Start condition** section, click **Use if defined**.  
For planned execution and fixed execution, in the Schedule Rule dialog box, in the **Start condition** section, click **Use if defined**.
- Start conditions are not disabled by the `-j` option of the `ajsp1an` command.

The unit with the wait condition starts to execute when the unit specified in the wait condition finishes executing.

A unit with a wait condition assigned is called a *unit with wait conditions*. A unit specified in a wait condition is called a *unit whose end is being waited for*.

The following figure shows how a wait condition controls the order in which jobs are executed.

Figure 2–26: Controlling execution order using wait conditions



Legend:

- > : Relation line
- -> : Flow of wait processing

In this example, a wait condition is assigned to Job 3 in Root Jobnet B, and Job 2 in Root Jobnet A is specified for the wait condition as the unit whose end is being waited for. When the system executes Root Jobnet A and Root Jobnet B, Job 3 waits for Job 2 to finish executing. When Job 2 finishes executing, Job 3 then starts.

You can use wait conditions to control the execution sequence for a variety of units. The following table describes the units to which wait conditions can be assigned, and the units you can specify as units whose end is being waited for.

Table 2–9: Units capable of accepting wait conditions and units specifiable as a unit whose end is being waited for

| No. | Unit type   |   | Capable of accepting wait conditions | Specifiable as unit whose end is being waited for |
|-----|-------------|---|--------------------------------------|---|
| 1   | Job group   | Job group itself                                    | N                                    | N   |
| 2   |             | Jobs immediately under a job group                  | Y#1                                  | N   |
| 3   | Root jobnet | Root jobnet without a start condition               | Y                                    | Y   |
| 4   |             | Units under a root jobnet without a start condition | Y#2                                  | Y#2   |

| No. | Unit type                       |  | Capable of accepting wait conditions | Specifiable as unit whose end is being waited for |
|-----|---------------------------------|--|--------------------------------------|---|
| 5   | Root jobnet                     | Root jobnet with a start condition                             | Y#3                                  | Y#3   |
| 6   |                                 | Units under a root jobnet with a start condition               | N#2                                  | N#2   |
| 7   | Start condition (.CONDITION)    |  | N                                    | N   |
| 8   | Nested jobnet                   |  | Y                                    | Y   |
| 9   | Remote jobnet                   | Remote jobnet  | N                                    | N   |
| 10  |                                 | Lower-level units in remote jobnet                             | N                                    | N   |
| 11  | Planning group                  | Planning group   | N                                    | Y   |
| 12  |                                 | Root jobnets directly under planning group                     | Y                                    | N   |
| 13  |                                 | Lower-level units in root jobnet directly under planning group | Y                                    | Y#4   |
| 14  | Standard job                    |  | Y                                    | Y   |
| 15  | Jobnet connector                |  | Y                                    | Y   |
| 16  | OR job                          |  | N                                    | N   |
| 17  | Judgment job                    |  | N                                    | N   |
| 18  | Event job                       | Event job in a jobnet  | Y                                    | Y   |
| 19  |                                 | Event job in a start condition (.CONDITION)                    | N                                    | N   |
| 20  | Action job                      |  | Y                                    | Y   |
| 21  | Custom job                      |  | Y                                    | Y   |
| 22  | Passing information setting job |  | Y                                    | Y   |
| 23  | HTTP connection job             |  | Y                                    | Y   |
| 24  | Custom event job                | Custom event job in a jobnet                                   | Y                                    | Y   |
| 25  |                                 | Custom event job in a start condition (.CONDITION)             | N                                    | N   |

Legend:

Y: Can be assigned a wait condition or specified as a unit whose end is being waited for

N: Cannot be assigned a wait condition or specified as a unit whose end is being waited for

#1

You can assign wait conditions to these jobs, but you cannot register them for execution.

#2

Even if a root jobnet has a start condition, you can disable the start condition by clicking **Do not use** in the **Start condition** section of the Register for Execution dialog box or the Schedule Rule dialog box. This means that you do not know whether the start condition will be applied for the units under the root jobnet with the start condition until the root jobnet is executed. Accordingly, you can assign wait conditions to the units under a root jobnet with a start condition. Also, you can specify a unit under a root jobnet with a start condition as the unit whose end is being waited for. Note, however, that if the start condition is used at execution, an execution error occurs.

#3

The following conditions must be satisfied if you want to assign wait conditions to a root jobnet with a start condition or to specify that root jobnet as the unit whose end is being waited for:

- JP1/AJS3 - Manager is version 10-00 or later.
- `condition` is set for the `PREWAITTEXTEND` environment setting parameter.

#4

When designating a unit in a root jobnet directly under a planning group as a unit whose end is being waited for, specify the name of the unit without the root jobnet name. That is, specify */planning group/job* rather than */planning group/root-jobnet/job*.

A wait condition controls the execution sequence of units governed by a single scheduler service. That is, the unit with the wait condition and the unit whose end is being waited for must be under the control of the same scheduler service.

#### Cautionary notes

- Job flows that incorporate wait conditions can be hard to comprehend, and tend to obscure the flow of work tasks and the jobnet hierarchy. When you create a job flow, to the extent possible, avoid using wait conditions. Use relation lines, nest jobnets, and combine multiple jobs instead.
- When you use wait conditions, space out when waits begin so that multiple waits are not concentrated at the same time. Also specify different scheduled start times for jobnets.

If multiple units with wait conditions must start waiting at the same time, design the job flow taking into account the following points regarding the time required from the time a unit whose end is being waited for ends until the unit with a wait condition starts executing:

- A unit with a wait condition checks the status of the unit whose end is being waited for. Accordingly, some time is necessary after the unit whose end is being waited for ends until the unit with the wait condition starts executing. How much time is required depends on the number of units with wait conditions that are waiting at the same time. These units include other units with wait conditions in the same scheduler service.

- One scheduler service can process approximately 4 to 20 units with wait conditions every second. For example, if 100 units with wait conditions simultaneously start waiting for one unit whose end is being waited for, approximately 5 to 25 seconds is required from the time the unit whose end is being waited for ends until the 100th unit with a wait condition starts executing.

- When you specify a root jobnet with a start condition as the unit whose end is being waited for, the time required until the unit with the wait condition starts executing depends on the number of execution generations of the root jobnet with the start condition. For example, if 100 units with wait conditions simultaneously start waiting for 100 root jobnets with start conditions and 10 execution generations (for each root jobnet), approximately 5 to 20 seconds is required after all the root jobnets with start conditions finish executing until the units with wait conditions start executing. If each of the 100 root jobnets with start conditions has 100 execution generations, the time is approximately 50 to 100 seconds.

- The length of time required until a unit with a wait condition starts executing depends on the performance of the computer and the jobnet configuration of the unit whose end is being waited for and the unit with the wait condition.

- You can use wait conditions from version 09-50 of JP1/AJS3 - View and JP1/AJS3 - Manager. However, you cannot use wait conditions if the database uses a compatible ISAM configuration, even when running JP1/AJS3 - Manager 09-50 or later.

## (b) Wait statuses

The *wait status* of a unit with wait conditions indicates whether the unit is still waiting for the unit whose end is being waited for to finish executing. The following table describes each wait status.

Table 2–10: List of wait statuses

| No. | Wait status              | Description  |
|-----|--------------------------|--|
| 1   | Wait incomplete          | The unit is still waiting for the unit whose end is being waited for to finish executing.  |
| 2   | Wait complete            | The unit is no longer waiting for the unit whose end is being waited for to finish executing.  |
| 3   | Wait incomplete (manual) | The unit is still waiting for the unit whose end is being waited for to finish executing (the wait condition was activated, causing the wait status to change to <i>Wait incomplete</i> )  |
| 4   | Wait complete (manual)   | The unit is no longer waiting for the unit whose end is being waited for to finish executing (the wait condition was deactivated, causing the wait status to change to <i>Wait complete</i> )  |
| 5   | Wait incomplete (rerun)  | The unit is still waiting for the unit whose end is being waited for to finish executing (the succeeding unit for an abnormally terminated unit was re-executed, and the status of the abnormally terminated unit changed to an end status). |

When the wait status transitions to *Wait complete* or *Wait complete (manual)*, the wait condition is satisfied and the unit with the wait condition begins to execute.

You can configure the behavior of a unit with a wait condition. For details, see (5) *Configuring the behavior of units with wait conditions*.

### (c) Flow of execution order control

To use wait conditions to control the execution order of units:

1. Define the unit whose end is being waited for and the unit to which the wait condition is to be assigned.

Create a detailed definition and schedule for the units.

2. Assign a wait condition to the unit.

The following settings define a wait condition:

- Name of the unit whose end is being waited for
- Wait method
- Behavior when not waiting for any generations

For details on the wait method and behavior when not waiting for any generations, see (5) *Configuring the behavior of units with wait conditions*.

For details on how to assign a wait condition, see 5.2.6 *Assigning wait conditions* in the *JPI/Automatic Job Management System 3 Operator's Guide*.

3. Register the root jobnets that manage the related units, namely the unit with wait conditions and the unit whose end is being waited for, for execution.

The unit with the wait condition waits for the unit whose end is being waited for to finish executing.

For details on how to register a root jobnet for execution, see 7. *Executing Jobnets* in the *JPI/Automatic Job Management System 3 Operator's Guide*.

## (2) Rules governing waiting regarding units with wait conditions and units whose ends are being waited for

When you register a unit with a wait condition for execution, the unit with the wait condition waits for the unit whose end is being waited for to finish executing, subject to specific rules.

The rules differ for each of the following scenarios:

- The unit with the wait condition is in the same root jobnet as the unit whose end is being waited for

- The unit with the wait condition and the unit whose end is being waited for are in different root jobnets
- The unit with the wait condition or the unit whose end is being waited for uses a start condition
- A planning group is specified as the unit whose end is being waited for

Each of these scenarios is described below.

**(a) The unit with the wait condition is in the same root jobnet as the unit whose end is being waited for**

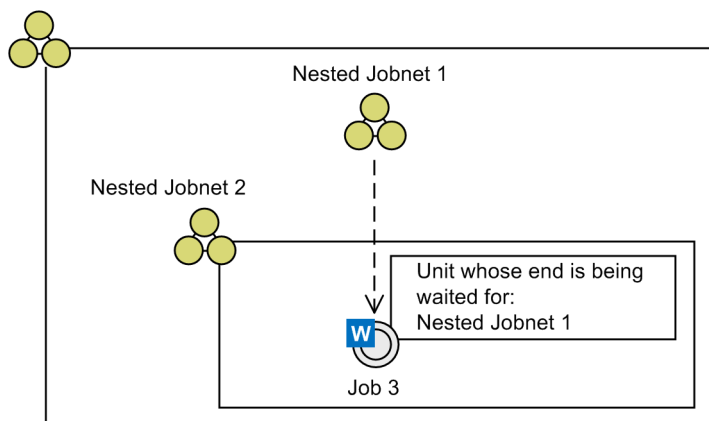
When the unit with the wait condition is associated with the same root jobnet as the unit whose end is being waited for, both units are executed as part of the same generation. Therefore, the unit with the wait condition will wait for the unit whose end is being waited for that is executed in the same generation to finish executing.

The following figure shows an example in which the unit with the wait condition and the unit whose end is being waited for are associated with the same root jobnet.

**Figure 2–27: Example where the unit with wait conditions and the unit whose end is being waited for are in the same root jobnet**

■ Unit configuration

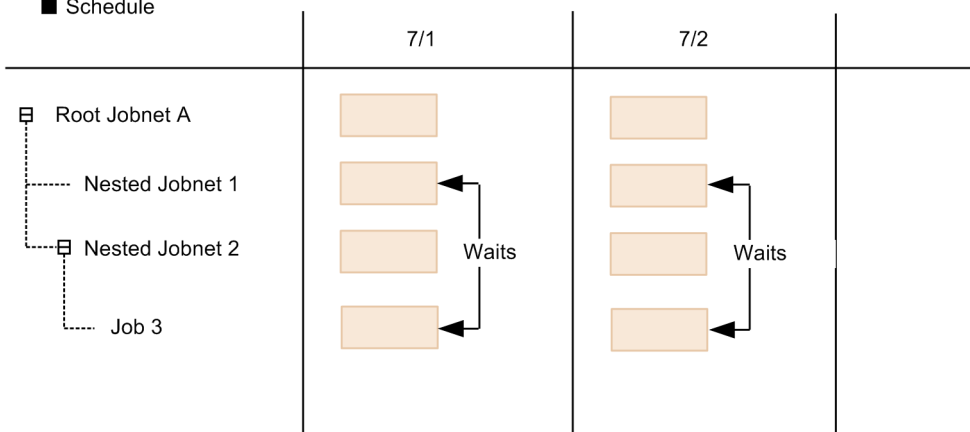
Root Jobnet A



Legend:

--> : Flow of wait processing

■ Schedule



In this example, Job 3 (the unit with wait conditions) and Nested Jobnet 1 (the unit whose end is being waited for) are both defined under Root Jobnet A. If Root Jobnet A is scheduled to execute once per day, generations of Root Jobnet A



are scheduled for 7/1 and 7/2 when the jobnet is registered for execution. In this scenario, Job 3 waits for the instance of Nested Jobnet 1 that is in the same generation to finish executing.

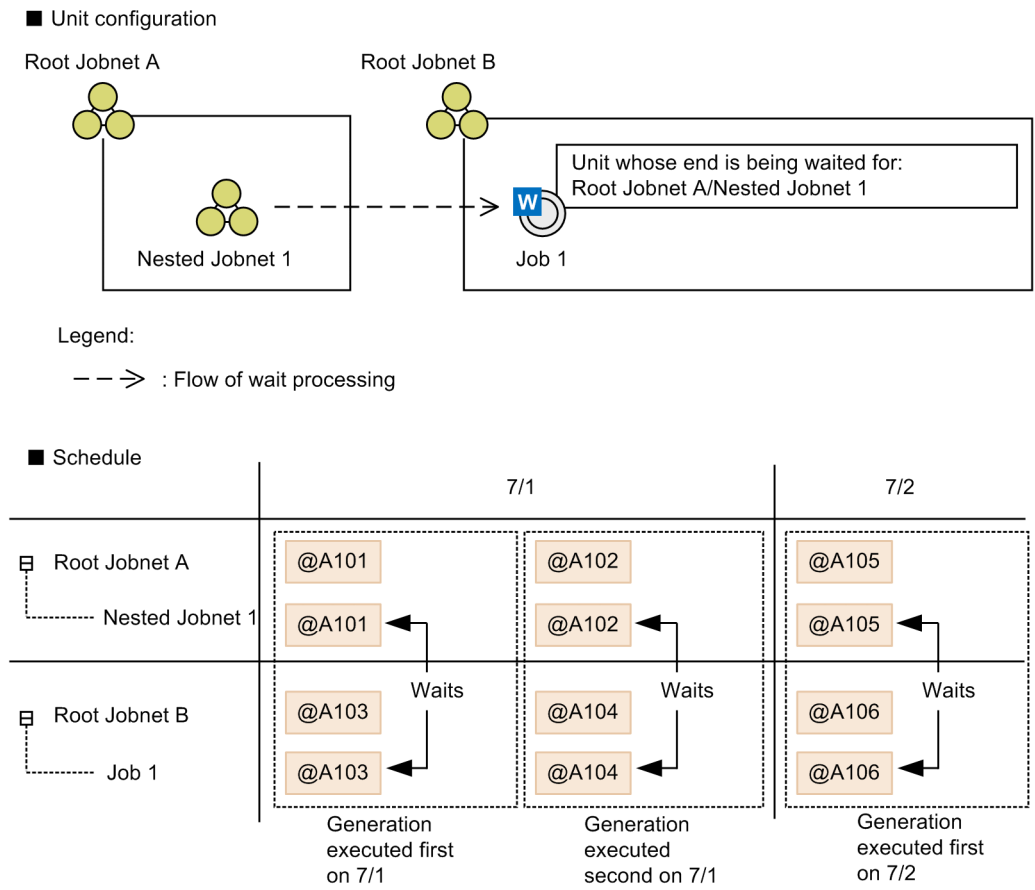
**(b) The unit with the wait condition and the unit whose end is being waited for are in different root jobnets**

When the unit with the wait condition and the unit whose end is being waited for are associated with different root jobnets, generations of each root jobnet are scheduled when you register the root jobnets for execution. In this scenario, the unit with wait conditions waits for the generation of the unit whose end is being waited for that satisfies both of the following conditions:

- Shares the same execution date
- Appears in the same position in the sequence when the generations are sorted in order of execution start time

The following figure shows an example in which the unit with wait conditions and the unit whose end is being waited for are in different root jobnets.

Figure 2–28: Example where the unit with wait conditions and the unit whose end is being waited for are in different root jobnets



In this example, Job 1 (the unit with wait conditions) and Nested Jobnet 1 (the unit whose end is being waited for) are defined under different root jobnets. If Root Jobnet A and Root Jobnet B are scheduled for execution twice on 7/1, two generations of Root Jobnet A are generated for 7/1 with the execution IDs @A101 and @A102. Similarly, two generations of Root Jobnet B are generated for 7/1 with the execution IDs @A103 and @A104. Generations @A103 and @A104 of Job 1 each wait for the generation of Nested Jobnet 1 that shares the same execution date and whose start time appears in the equivalent position. Therefore, the wait condition of the first generation of Job 1 (@A103) applies to the generation of Nested Jobnet 1 that executes first (@A101). Similarly, the wait condition of @A104, the second

generation of Job 1, applies to @A102, the second generation of Nested Jobnet 1. Of the generations of Job 1 scheduled for 7/2, each generation waits for the generation of Nested Jobnet 1 that starts in the corresponding position.

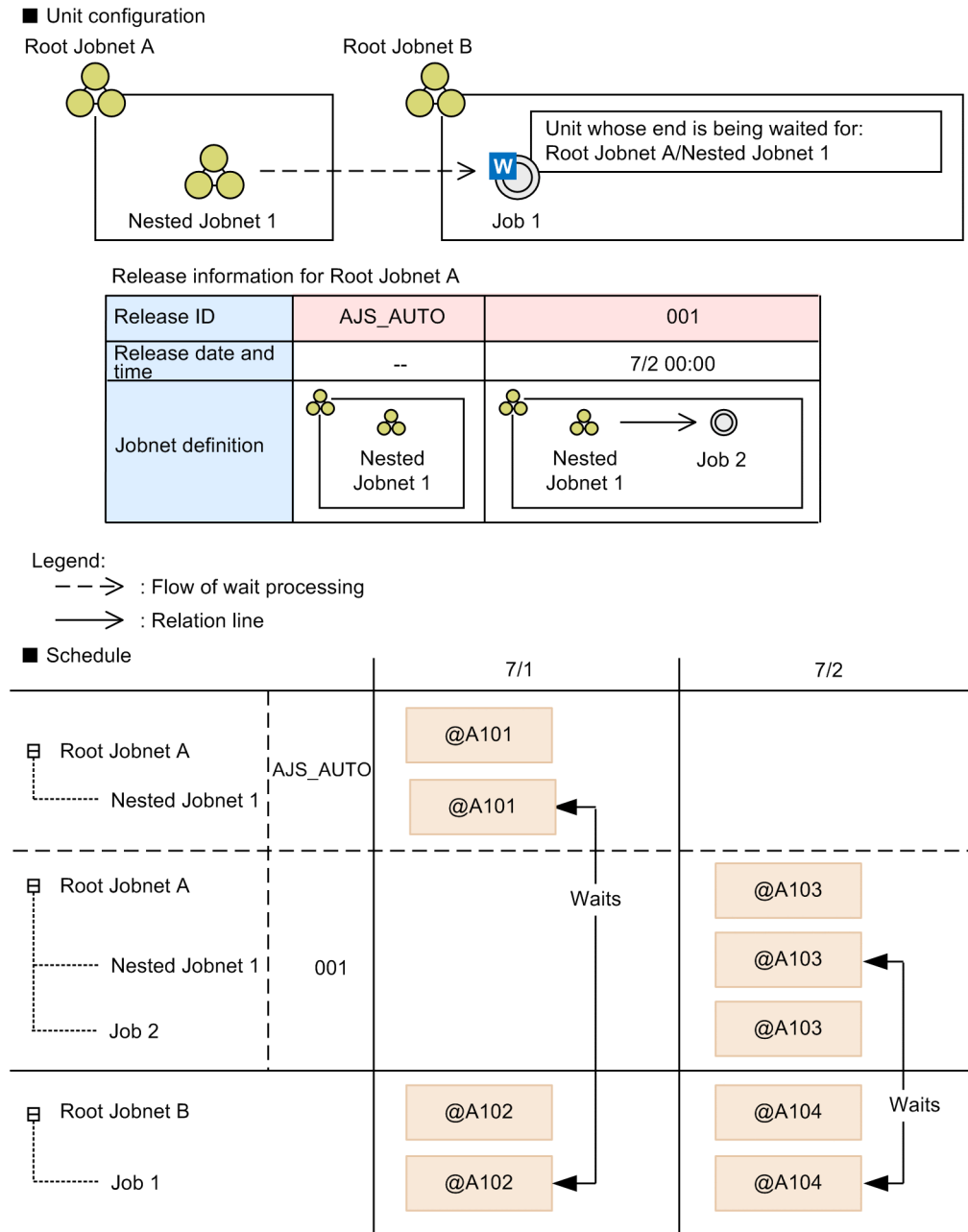
Below are examples of using wait conditions in conjunction with JP1/AJS3 functionality.

Using the jobnet release function to replace the jobnet definition of the unit whose end is being waited for

If you use the jobnet release function to switch the jobnet definition of a unit whose end is being waited for without stopping the jobnet, the jobnet targeted by the wait condition switches automatically.

The following figure shows an example of releasing a jobnet serving as a unit whose end is being waited for.

Figure 2–29: Releasing a jobnet serving as a unit whose end is being waited for



In this example, Root Jobnet A is subject to a release registration that releases the jobnet definition with the release ID 001 at the specified release time of 00:00 on 7/2. When Root Jobnet A is registered for execution, the generation scheduled for 7/1 (@A101) is subject to the jobnet definition associated with release ID AJS\_AUTO, and the generation scheduled for 7/2 (@A103) is subject to the jobnet definition associated with release ID 001. On 7/1, Job

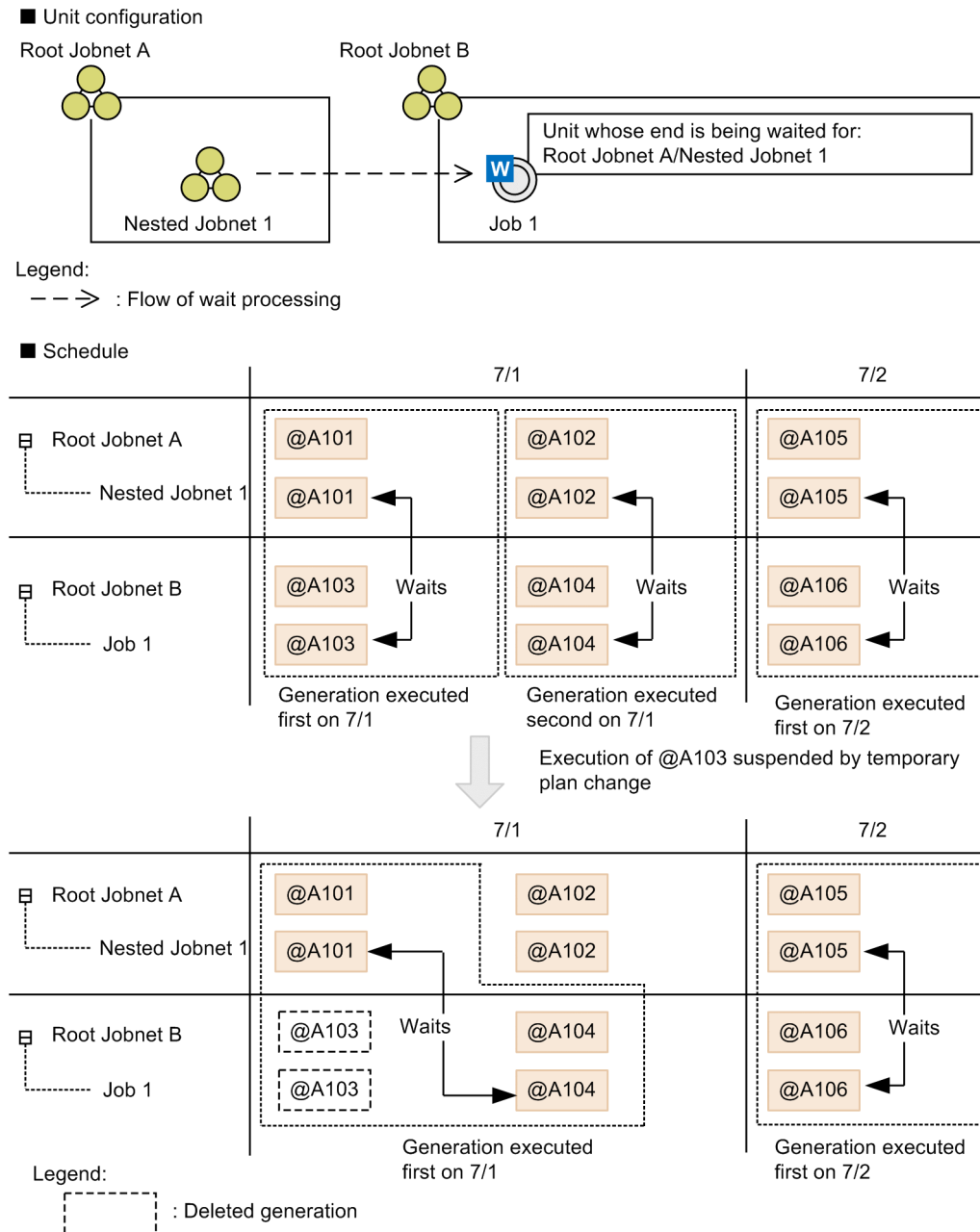
1 waits for generation @A101 of the Nested Jobnet 1 associated with release ID `AJS_AUTO`. From 7/2, Job 1 waits for generation @A103 of the Nested Jobnet 1 associated with release ID 001.

### Changing the start order of scheduled generations of a jobnet registered for execution

If you perform a temporary plan change or other operation that changes the order in which generations of a registered jobnet are scheduled to start, the generations targeted by the wait condition automatically change to reflect the order in the new schedule.

The following figure shows an example in which a temporary plan change alters the order in which scheduled generations start. Note that, if execution is prohibited for a root jobnet that was registered for planned execution, the wait status is passed to the next generation. For details on prohibiting execution, see 4.5.5 *Prohibiting execution of a job or jobnet* in the manual *JP1/Automatic Job Management System 3 Overview*.

Figure 2–30: Example of using a temporary plan change to change the start sequence of scheduled generations



In this example, Root Jobnet A which manages Nested Jobnet 1 (the unit whose end is being waited for), and Root Jobnet B which manages Job 1 (the unit with a wait condition), are both scheduled to execute twice on 7/1. Although

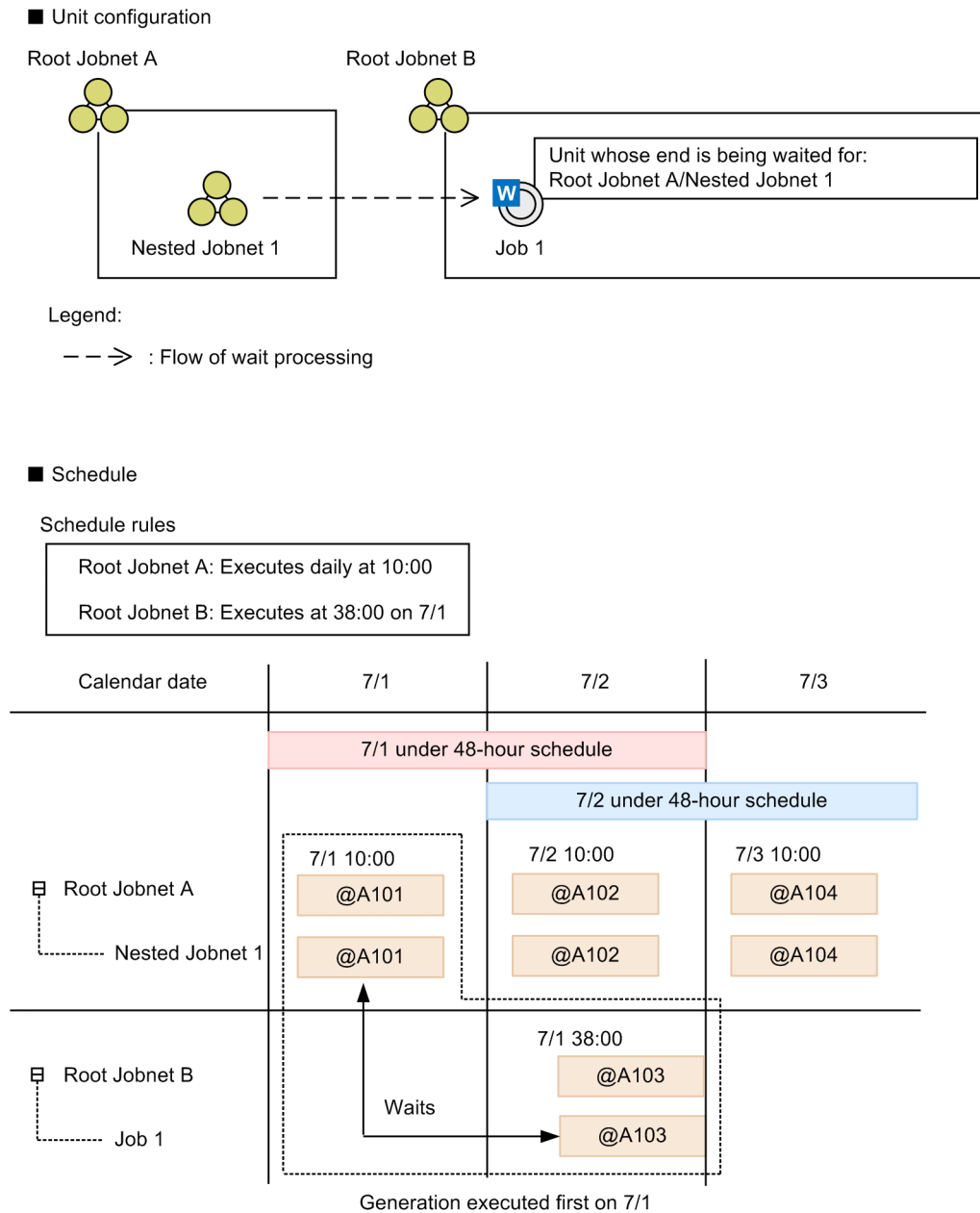
generation @A103 of Root Jobnet B was scheduled to execute first on 7/1, a temporary plan change caused its execution to be canceled, and @A104 became the generation executed first. Thus, generation @A104 of Job 1 waits for generation @A101 of Nested Jobnet 1.

### Effects of a 48-hour schedule on wait conditions

If you use a 48-hour schedule, the target generation of a wait condition is determined by its execution date in the 48-hour schedule.

The following figure shows an example of wait conditions in a 48-hour schedule.

Figure 2–31: Example of wait conditions in a 48-hour schedule



In this example, the scheduler services for Root Jobnet A and Root Jobnet B use the 48-hour schedule. Root Jobnet A, which governs Nested Jobnet 1 (the unit whose end is being waited for), is scheduled to execute daily at 10:00. Root Jobnet B, which governs Job 1 (the unit with wait conditions), is scheduled to execute at 38:00 on 7/1. In this scenario, the first generation of Job 1 to be executed on 7/1 is @A103, scheduled for 38:00 (actually 14:00 on 7/2). Thus, generation @A103 of Job 1 waits for generation @A101 of Nested Jobnet 1, which is the first generation of Nested Jobnet 1 scheduled for 7/1.

## Cautionary notes

- You can set a different base time for the unit with wait conditions and the unit whose end is being waited for. However, doing so can complicate the process of matching wait conditions to their targets. We recommend that you coordinate the base times of units with wait conditions and their units whose ends are being waited for.
- In UNIX, you can specify different time zones for the unit with the wait condition and the unit whose end is being waited for. However, doing so can complicate the process of matching wait conditions to their targets. We recommend that you coordinate the time zones of units with wait conditions and their units whose ends are being waited for.
- Make sure that the saved generations for the unit with the wait condition and the unit whose end is being waited for cover at least one day of operation. This is to ensure that the unit with the wait condition waits as intended and starts executing when the unit whose end is being waited for finishes. If you cannot save one day's generations of the unit whose end is being waited for, there is a chance that the unit might not start executing. On the other hand, if you cannot save one day's generations of the unit with the wait condition, the condition might target a unit whose end is being waited for other than the one intended.

## (c) The unit with the wait condition or the unit whose end is being waited for uses a start condition

### ■ When the unit whose end is being waited for uses a start condition

When a root jobnet with a start condition specified as the unit whose end is being waited for starts executing, both a monitoring generation and execution generations are generated. The scheduled execution generation of the unit with the wait condition waits for both the monitoring generation and execution generations of the unit whose end is being waited for to finish executing.

In this scenario, the scheduled execution generation of the unit with wait conditions waits for the monitoring and execution generations of the unit whose end is being waited for that satisfy all of the following criteria to finish executing.

#### Monitoring generation

- The monitoring generation shares the same execution date as the scheduled execution generation of the unit with wait conditions.
- The monitoring generation appears in the same position in the sequence as the scheduled execution of the unit with wait conditions when all the monitoring generations are sorted in order of scheduled execution start time.

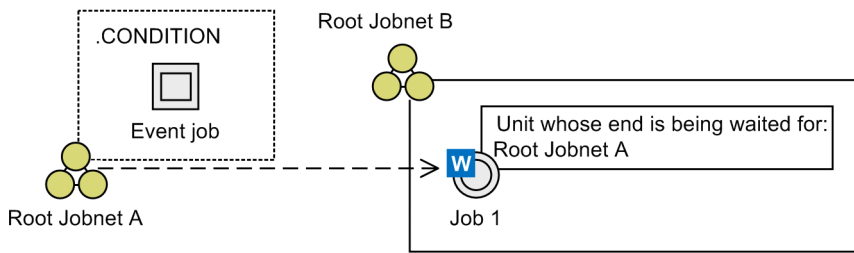
#### Execution generation

- All execution generations that are generated together with a monitoring generation

The following figure shows an example in which the root jobnet with a start condition is the unit whose end is being waited for.

Figure 2–32: Example where the root jobnet with a start condition is the unit whose end is being waited for

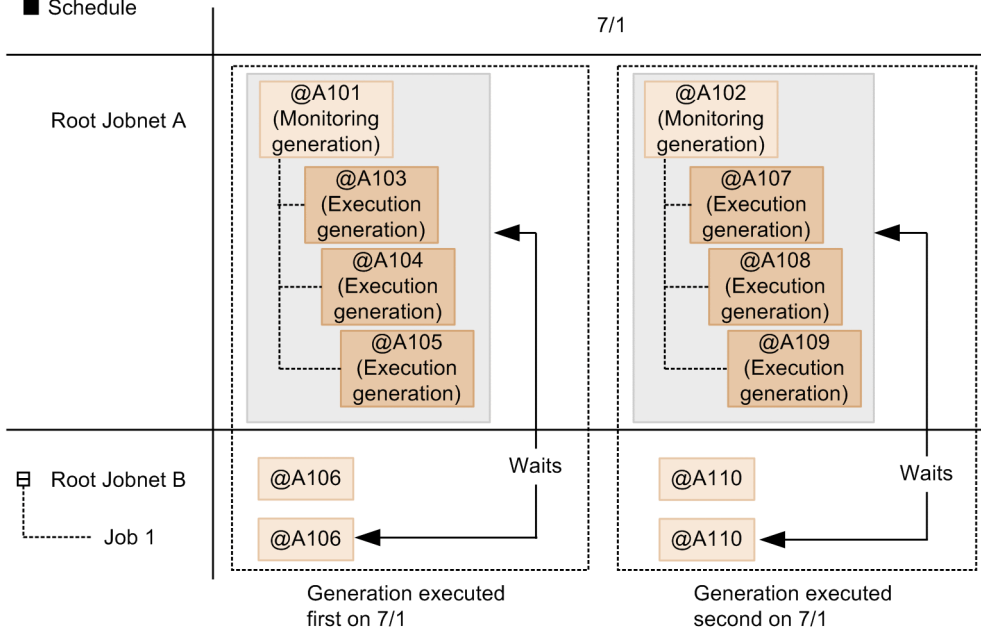
■ Unit configuration



Legend:

--> : Flow of wait processing

■ Schedule



In this example, Root Jobnet A (the root jobnet with a start condition) is defined as the unit whose end is being waited for. Generations @A101 and @A102 of Root Jobnet A use the start condition.

If Root Jobnet A and Root Jobnet B are scheduled and registered for execution twice on 7/1, a monitoring generation with execution ID @A101, an execution generation with execution ID @A103, a monitoring generation with execution ID @A102, and an execution generation with execution ID @A107 are generated for Root Jobnet A for 7/1.

When the start condition is satisfied during monitoring by monitoring generation @A101 and execution generation @A103 starts executing, execution generation @A104 is generated. When the start condition is satisfied a second time, execution generation @A104 starts executing and execution generation @A105 is generated. For Root Jobnet B for 7/1, however, generations with execution IDs @A106 and @A110 are scheduled. Generation @A106 of Job 1 waits for the monitoring generation and the execution generations of Root Jobnet A in the same position of the execution sequence for the same execution date. Accordingly, generation @A106 of Job 1 waits until monitoring generation @A101 and execution generations @A103 to @A105 of Root Jobnet A finish executing.

Similarly, generation @A110 of Job 1 (the second generation to be executed) waits until monitoring generation @A102 and execution generations @A107 to @A109 of Root Jobnet A finish executing.

## Cautionary notes

- When you specify a root jobnet with a start condition as the unit whose end is being waited for, explicitly specify numbers for the valid range (either the number of times or a period) of the start condition. If the valid range for a start condition is unlimited for both the number of times and the period, an abnormal end occurs when the unit with the wait condition starts waiting. The reason for the abnormal end is to prevent the unit with the wait condition from continuing to wait when the monitoring generation does not finish executing.
- When you specify a root jobnet with a start condition as the unit whose end is being waited for, specify the number of logs to keep after reviewing the number of execution generations that will be generated. Also make sure that generations will not be deleted during waiting because the number of specified logs has been exceeded. For details, see *(5)(c) Behavior of units with wait conditions when start conditions are used for the units whose ends are being waited for*.
- When you specify a jobnet that has start conditions as a unit whose end is being waited for, make sure not to mix generations that have start conditions with generations that do not have start conditions. For example, if you specify multiple schedule rules for a jobnet that has start conditions, all the schedule rules must either use or not use start conditions. If there are generations that have start conditions and that do not have start conditions, wait conditions will not work properly.

### ■ When the unit with a wait condition uses a start condition

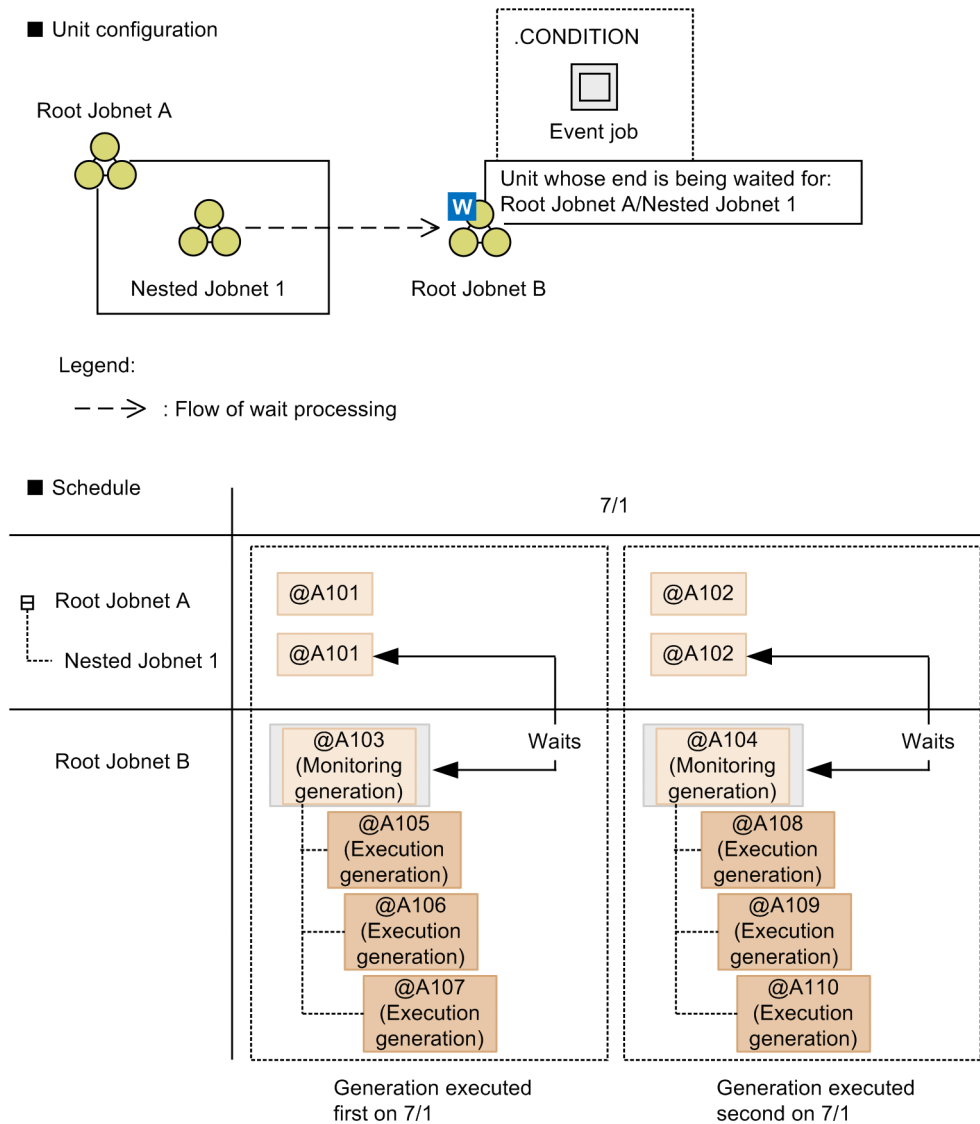
When a root jobnet with a start condition starts executing, both a monitoring generation and execution generations are generated. The monitoring generation of the unit with the wait condition waits for the unit whose end is being waited for to finish executing.

In this scenario, the monitoring generation of the unit with the wait condition waits for the generation of the unit whose end is being waited for that satisfies both of the criteria below to finish executing. Note that the execution generations of the unit with the wait condition do not wait. The execution generations start executing when the start condition is satisfied.

- The generation of the unit whose end is being waited for shares the same execution date as the monitoring generation of the unit with the wait condition.
- The generation of the unit whose end is being waited for appears in the same position in the sequence as the monitoring generation of the unit with the wait condition when all the generations of the unit whose end is being waited for and all the monitoring generations of the unit with the wait condition are sorted in order of scheduled start time.

The following figure shows an example in which a root jobnet with a start condition is specified as the unit with a wait condition.

Figure 2–33: Example where the root jobnet with a start condition is specified as the unit with a wait condition



In this example, Root Jobnet B (the root jobnet with a start condition) is defined as the unit with a wait condition.

If Root Jobnet A and Root Jobnet B are scheduled and registered for execution twice on 7/1, generations with execution IDs @A101 and @A102 are generated for Root Jobnet A for 7/1. For Root Jobnet B, for 7/1, a monitoring generation with execution ID @A103, an execution generation with execution ID @A105, a monitoring generation with execution ID @A104, and an execution generation with execution ID @A108 are generated. Monitoring generation @A103 waits for the end of scheduled execution generation @A101 of Nested Jobnet 1, which has the same execution date and which is in the same position in the execution sequence of Root Jobnet A.

When generation @A101 finishes executing, the wait condition is satisfied. When the wait condition is satisfied, monitoring generation @A103 starts monitoring for the start condition. When the start condition is satisfied and execution generation @A105 starts executing, execution generation @A106 is generated. When the start condition is satisfied a second time, execution generation @A106 starts executing and execution generation @A107 is generated.

Similarly, monitoring generation @A104 of Root Jobnet B waits for the end of execution of generation @A102, which is executed second in Nested Jobnet 1.

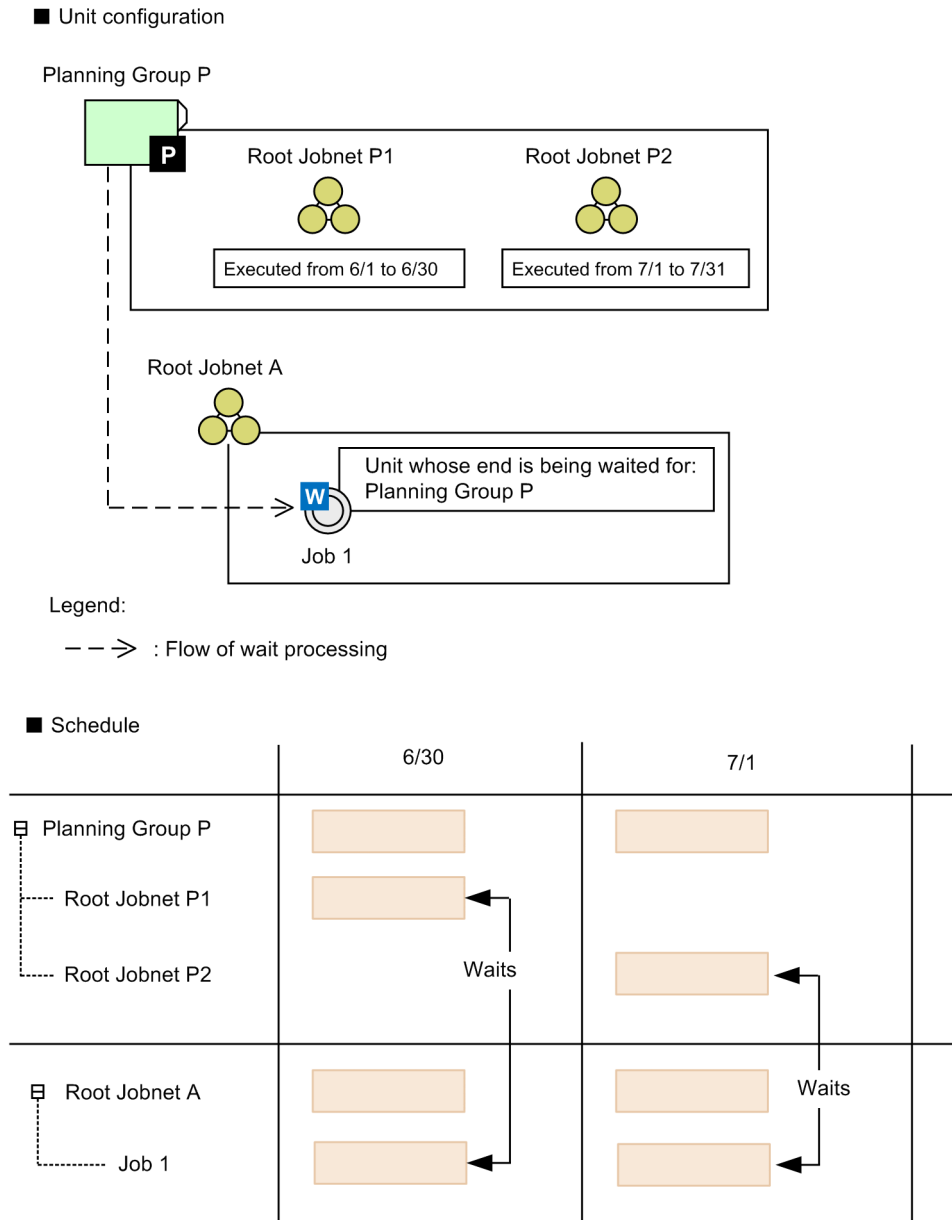


### (d) A planning group is specified as the unit whose end is being waited for

When you specify a planning group as a unit whose end is being waited for, the target of the wait condition is the root jobnet running under the planning group. When the planning group switches execution from one root jobnet to another, the target of the wait condition changes automatically.

The following figure shows an example in which a planning group is specified as the unit whose end is being waited for.

Figure 2–34: Example where planning group is specified as unit whose end is being waited for



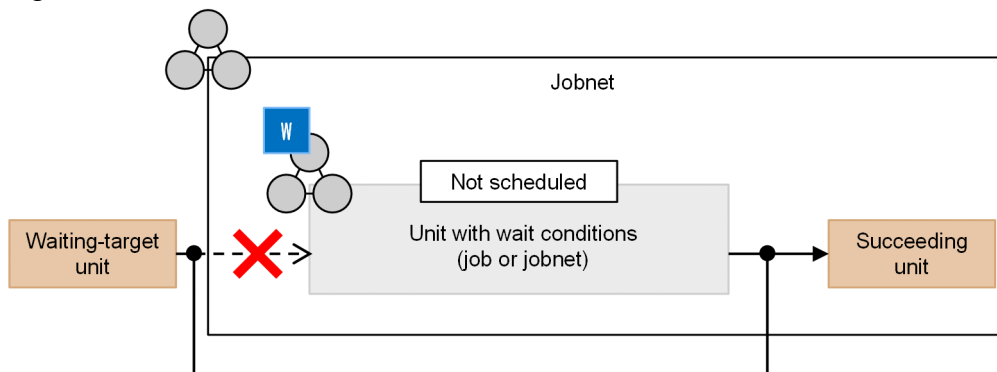
In this example, Planning Group P is specified as the unit whose end is being waited for, and defines two root jobnets: Root Jobnet P1 which runs between 6/1 and 6/30, and Root Jobnet P2 which runs between 7/1 and 7/31. From 6/1 to 6/30, Job 1 waits for Root Jobnet P1. When 7/1 arrives and the root jobnet running under the planning group switches to Root Jobnet P2, Root Jobnet P2 becomes the target of Job 1's wait condition.

### (3) Behavior in the case where a unit with wait conditions is not scheduled for execution

For a unit that has wait conditions and is not scheduled for execution, by default, no wait occurs. The succeeding unit runs without waiting for the execution of the waiting-target units to end.

The following figure shows the behavior in the case where a unit with wait conditions is not scheduled for execution.

Figure 2–35: Behavior in the case where a unit with wait conditions is not scheduled for execution




If the unit with wait conditions is not scheduled for execution, it does not wait for the waiting-target unit to end.

If the unit with wait conditions is not scheduled for execution, the succeeding unit starts without waiting for the waiting-target units to end.

Legend:

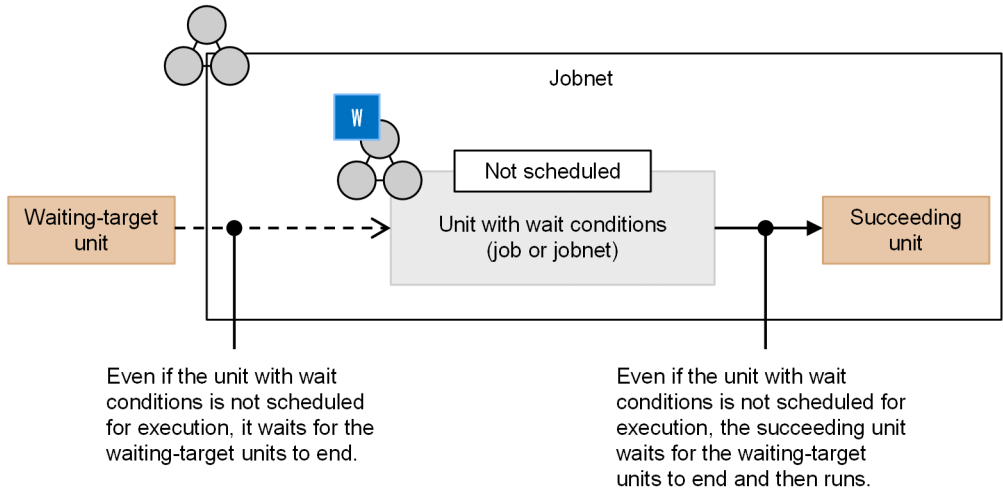
Relation line: 

Wait (for the left-side unit to end) that does not occur: 

This behavior can be changed by using the `PREWAITNOSCHUNITS` environment setting parameter. If `yes` is specified for the `PREWAITNOSCHUNITS` environment setting parameter, a wait occurs even if the unit with wait conditions is not scheduled for execution. For details about the `PREWAITNOSCHUNITS` environment setting parameter, see 20.4.2(122) *PREWAITNOSCHUNITS* in the *JP1/Automatic Job Management System 3 Configuration Guide*.

The following figure shows the behavior in the case where `yes` is specified for the `PREWAITNOSCHUNITS` environment setting parameter.

Figure 2–36: Behavior in the case where yes is specified for the PREWAITNOSCHUNITS environment setting parameter



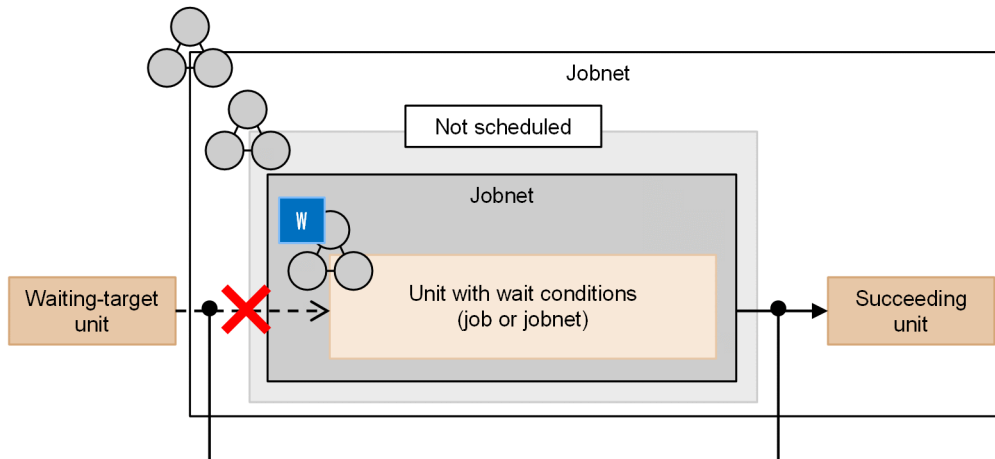
Legend:

- Relation line:
- Flow of wait processing:

If the jobnet that contains a unit with wait conditions is not scheduled for execution, no wait occurs regardless of the value specified for the PREWAITNOSCHUNITS environment setting parameter. An upper jobnet that is not scheduled for execution is placed in *Bypassed* status when its own execution conditions are met (for example, when execution of the preceding job or jobnet ends). All units that belong to the jobnet (including units with wait conditions) will be placed in *Bypassed* status.

The following figure shows the behavior in the case where the jobnet that contains a unit with wait conditions is not scheduled for execution.

Figure 2–37: Behavior in the case where the jobnet that contains a unit with wait conditions is not scheduled for execution




If the upper-level jobnet of the unit with wait conditions is not scheduled for execution, the unit does not wait for the waiting-target unit to end.

If the upper-level jobnet of the unit with wait conditions is not scheduled for execution, the succeeding unit starts without waiting for the waiting-target units to end.

Legend:

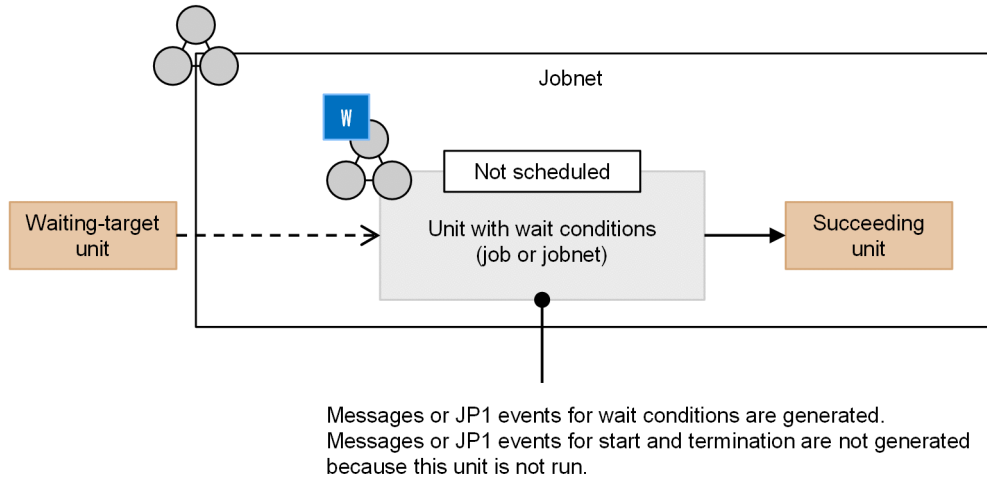
Relation line: 

Wait (for the left-side unit to end) that does not occur: 

If `yes` is specified for the `PREWAITNOSCHUNITS` environment setting parameter, the wait conditions of a unit take effect even if the unit is not scheduled for execution, and messages and JP1 events are output according to the wait status. When the wait conditions are met, because the unit that has wait conditions and is not scheduled for execution is not run, a start message, end message, and JP1 events are not output. However, if a wait error occurs, the status of the unit with wait conditions changes to *Ended abnormally*, and messages and JP1 events are output.

The following figure shows the behavior of messages and JP1 events in the case where `yes` is specified for the `PREWAITNOSCHUNITS` environment setting parameter.

Figure 2–38: Behavior of messages and JP1 events in the case where yes is specified for the PREWAITNOSCHUNITS environment setting parameter



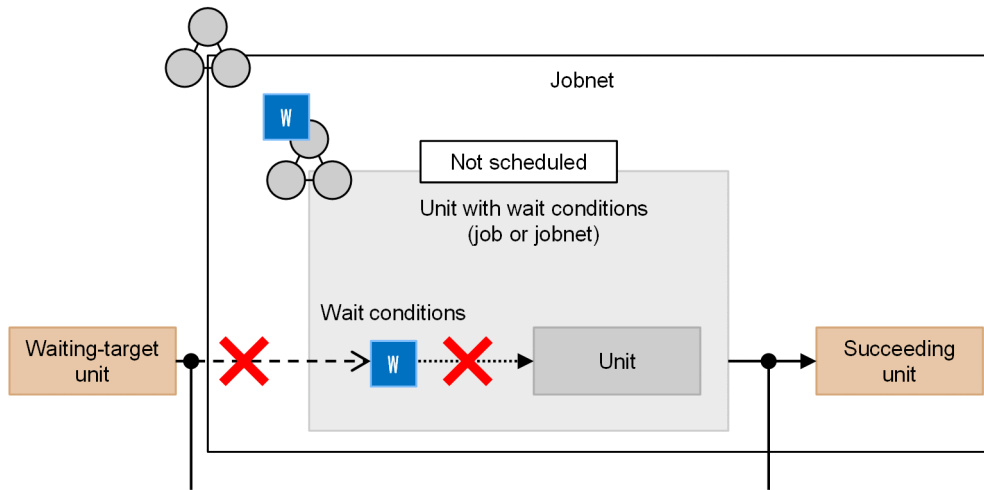
Legend:

- Relation line: →
- Wait for the left-side unit to end: →

### Supplementary note

If a unit with wait conditions is not scheduled for execution, by default, the processing to evaluate the wait conditions also ends without being performed, as well as the processing to run the unit.

Figure 2–39: Flow of processing in the case where the PREWAITNOSCHUNITS environment setting parameter is omitted



If the unit with wait conditions is not scheduled for execution, it does not wait for the waiting-target unit to end.

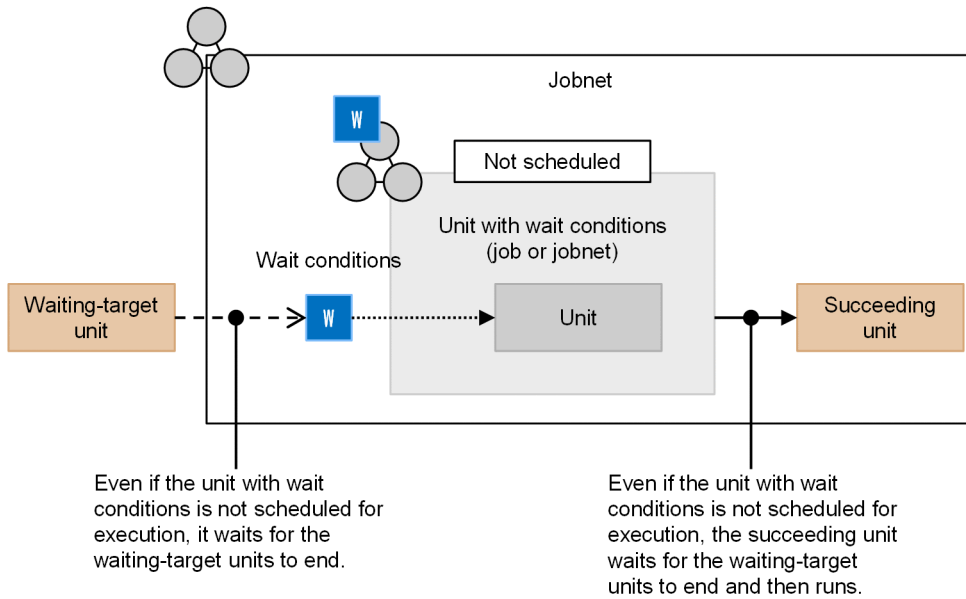
If the unit with wait conditions is not scheduled for execution, the succeeding unit starts without waiting for the waiting-target units to end.

Legend:

- Relation line:
- Wait (for the left-side unit to end) that does not occur:
- Processing flow that is not executed:

If wait conditions are provided outside the range in which the processing of the unit with wait conditions is not performed when `yes` is specified for the `PREWAITNOSCHUNITS` environment setting parameter, only the wait processing is performed. However, even when `yes` is specified for the `PREWAITNOSCHUNITS` environment setting parameter, if the upper-level jobnet of the unit with wait conditions is not scheduled for execution, no wait is performed. This is because these conditions are included in the range in which the processing of the upper-level jobnet is not performed.

Figure 2–40: Flow of processing in the case where yes is specified for the PREWAITNOSCHUNITS environment setting parameter



Legend:

- Relation line:
- Flow of wait processing:
- Processing flow:

#### Cautionary note

Even if the environment setting parameter PREWAITNOSCHUNITS is *yes*, units are not made to wait in the following cases. The succeeding unit is executed without waiting for the end of the unit whose end is being waited for.

- When a unit with a wait condition that is not scheduled for execution is a dependent unit, and the status of the Judgment job is *Normal end + False*.
- When a unit with a wait condition that is not scheduled for execution is the preceding event job or custom event job of an OR job, and another preceding event job or custom event job ends.

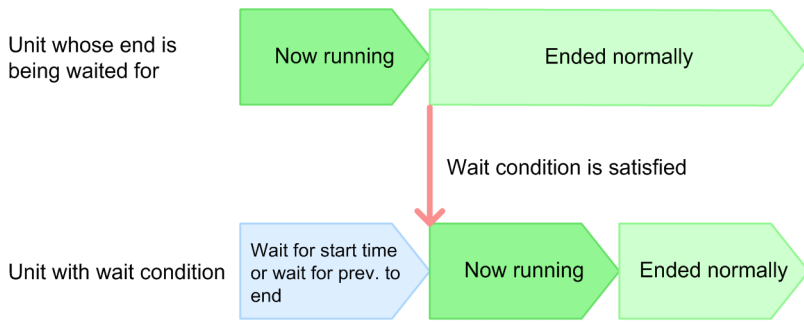
## (4) Status transitions of units with wait conditions and units whose ends are being waited for

### (a) Status transitions of units with wait conditions and units whose ends are being waited for when no start condition is used

#### ■ Status transitions of a unit with wait conditions that is scheduled for execution

The following figure shows the status transitions of a unit when its wait conditions are met.

Figure 2–41: Status transitions of a unit that is scheduled for execution when its wait conditions are met (when no start condition is used)



While the status of the unit whose end is being waited for is *Now running*, the unit with wait conditions waits in *Wait for start time* or *Wait for prev. to end* status until the wait condition is satisfied. Its status while waiting depends on the unit type.

The table below lists, for each unit type, the status that the unit with wait conditions adopts before the wait condition is met. For details on the types of units that can accept wait conditions, see (1)(a) *Units with wait conditions and units whose ends are being waited for*.

Table 2–11: Status of units with wait conditions before the wait condition is satisfied by unit type (if the units are scheduled for execution)

| No. | Type of unit with wait conditions   | Status before wait condition is satisfied |
|-----|---|---|
| 1   | Root jobnet   | Wait for start time                       |
| 2   | Nested jobnet   | Wait for prev. to end                     |
| 3   | <ul style="list-style-type: none"> <li>• Standard job</li> <li>• Jobnet connector</li> <li>• Event job</li> <li>• Action job</li> <li>• Custom job</li> <li>• Passing information setting job</li> <li>• HTTP connection job</li> <li>• Custom event job</li> </ul> | Wait for prev. to end                     |

When the unit with wait conditions is in *Wait for start time* or *Wait for prev. to end* status, the wait condition is satisfied when the unit whose end is being waited for transitions to one of the following statuses:

- Ended normally
- Ended with warning
- Bypassed

When the wait condition is satisfied, the unit with wait conditions enters *Now running* status and starts executing.

If the unit whose end is being waited for ends abnormally, the unit with wait conditions remains in *Wait for start time* or *Wait for prev. to end* status and continues to wait for the wait condition to be satisfied. If this occurs, fix the problem that caused the unit whose end is being waited for to terminate abnormally and then rerun the unit.

### ■ Status transitions of a unit with wait conditions that is not scheduled for execution

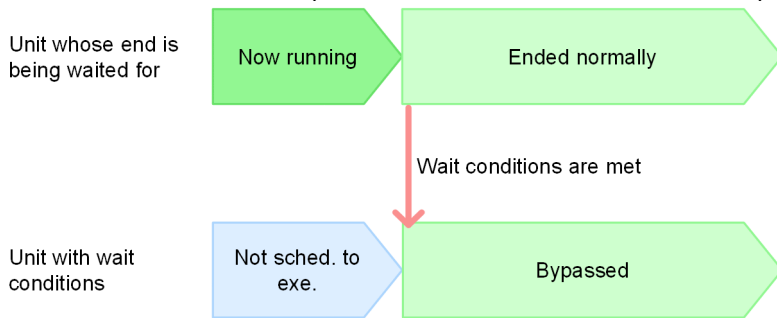
If *yes* is specified for the `PREWAITNOSCHUNITS` environment setting parameter, a unit with wait conditions performs a wait even when it is not scheduled for execution. For details about the `PREWAITNOSCHUNITS`



environment setting parameter, see 20.4.2(122) *PREWAITNOSCHUNITS* in the *JP1/Automatic Job Management System 3 Configuration Guide*.

The following figure shows the status transitions of units when a wait condition is satisfied.

Figure 2–42: Status transitions of a unit that is not scheduled for execution when its wait conditions are met (when no start condition is used)



If *yes* is specified for the *PREWAITNOSCHUNITS* environment setting parameter, a unit with wait conditions waits for wait conditions to be met even when it is not scheduled for execution. The unit continues to wait in the *Not sched. to exe.* status while the waiting-target unit (unit whose end is being waited for) is in the *Now running* status.

The table below lists, for each unit type, the status that the unit with wait conditions adopts before the wait condition is met. For details on the types of units that can accept wait conditions, see (1)(a) *Units with wait conditions and units whose ends are being waited for*.

Table 2–12: Status of units with wait conditions before the wait condition is satisfied by unit type (if the units are not scheduled for execution)

| No. | Type of unit with wait conditions   | Status before wait condition is satisfied |
|-----|---|---|
| 1   | Nested jobnet   | Not sched. to exe.                        |
| 2   | <ul style="list-style-type: none"> <li>• Standard job</li> <li>• Event job</li> <li>• Action job</li> <li>• Custom job</li> <li>• Passing information setting job</li> <li>• HTTP connection job</li> <li>• Custom event job</li> </ul> | Not sched. to exe.                        |

When the unit with wait conditions is in *Not sched. to exe.* status, the wait condition is satisfied when the unit whose end is being waited for transitions to one of the following statuses:

- Ended normally
- Ended with warning
- Bypassed

When the wait conditions of a unit are met, the unit enters the *Bypassed* status and the succeeding job begins to run.

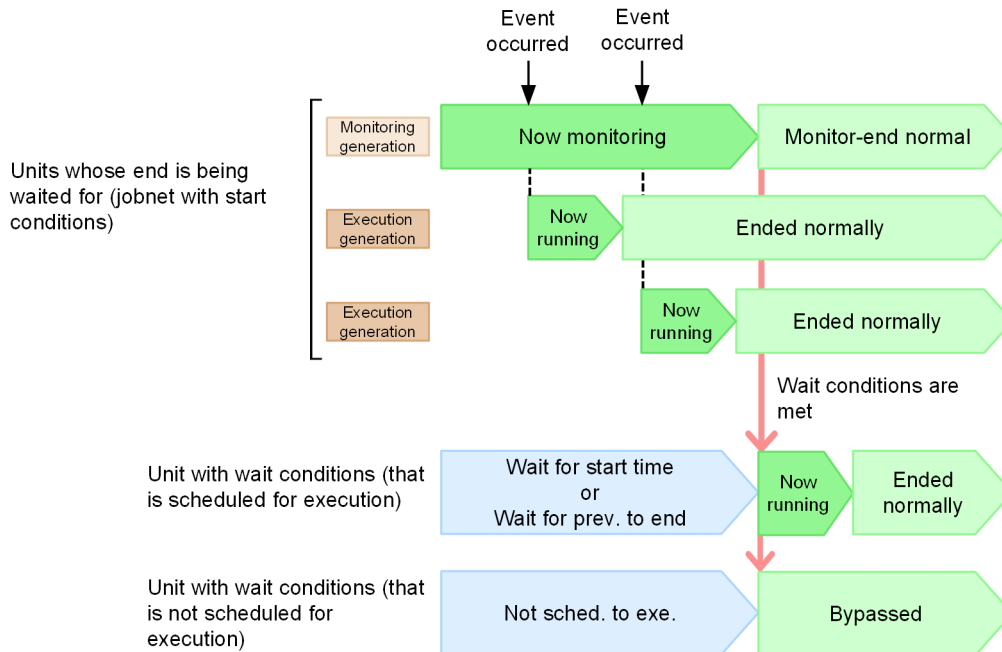
If the unit whose end is being waited for ends abnormally, the unit with wait conditions remains in *Not sched. to exe.* status and continues to wait for the wait condition to be satisfied. If this occurs, fix the problem that caused the unit whose end is being waited for to terminate abnormally and then rerun the unit.

## (b) Status transitions of units with wait conditions and units whose ends are being waited for when a start condition is used

### ■ Status transitions when units whose ends are being waited for use start conditions

The following figure shows the status transitions of units when a wait condition is satisfied.

Figure 2–43: Status transitions of units when a wait condition is satisfied (when the unit whose end is being waited for uses a start condition)



If a unit with wait conditions is scheduled for execution, the unit waits for the wait conditions to be met in the *Wait for start time* or *Wait for prev. to end* status under either of the following conditions: While the monitoring generation of the waiting-target unit (jobnet with start conditions) is in the *Now monitoring* status or while the execution generation is in the *Now running* status. Which status the unit enters depends on the type of the unit. For details about the status of a unit with a wait condition until the wait condition is satisfied, see [Table 2-11](#).

A unit with wait conditions enters the *Now running* status and begins to run when execution of the jobnet with start conditions ends and the wait conditions are met, if the unit is scheduled for execution.

A unit that has wait conditions and is not scheduled for execution waits for the conditions to be met in the *Not sched. to exe.* status, if environment setting parameter `PREWAITNOSCHUNITS` is `yes`, under either of the following conditions: While the monitoring generation of the waiting-target unit (jobnet with start conditions) is in the *Now monitoring* status or while the execution generation is in the *Now running* status. For details about the `PREWAITNOSCHUNITS` environment setting parameter, see [20.4.2\(122\) PREWAITNOSCHUNITS](#) in the *JPI/Automatic Job Management System 3 Configuration Guide*. For details about the status of a unit with a wait condition until the wait condition is satisfied, see [Table 2-12](#).

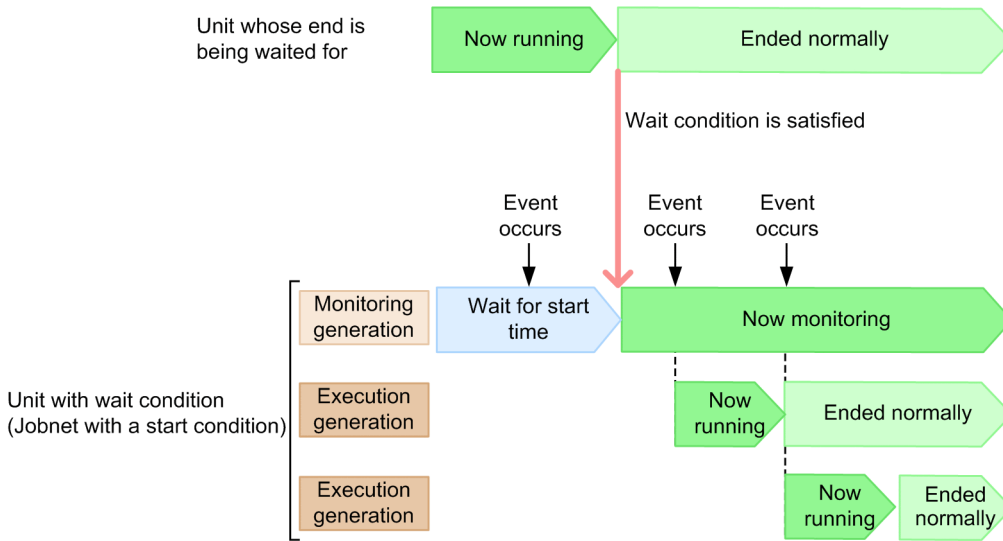
When a jobnet with start conditions ends and wait conditions are met, a unit with wait conditions that is not scheduled to be executed enters the *Bypassed* status and the succeeding unit runs.

You can change the status of the monitoring generation or an execution generation that marks the end of execution of the root jobnet with the start condition by changing the wait condition. For details, see [\(5\)\(c\) Behavior of units with wait conditions when start conditions are used for the units whose ends are being waited for](#).

## ■ Status transitions when units with wait conditions use start conditions

The following figure shows the status transitions of units when a wait condition is satisfied.

Figure 2–44: Status transitions of units when a wait condition is satisfied (when the unit with a wait condition uses a start condition)



While the unit whose end is being waited for is in the *Now running* status, the monitoring generation of the unit with a wait condition (that is, the root jobnet with a start condition) waits in the *Wait for start time* status for the wait condition to be satisfied. During this time, if an event to be monitored by the start condition occurs, the start condition is not satisfied because monitoring for the start condition has not started yet. As a result, no execution generation starts executing.

### Cautionary note

While a root jobnet with a start condition is waiting for the execution of a unit whose end is being waited for to finish executing, no events are monitored. If the unit whose end is being waited for is delayed, the start of monitoring for events is also delayed. To avoid this problem, make sure that a delay in monitoring for events does not cause any problems when you assign a wait condition to a root jobnet with a start condition.

#### Example of a monitoring delay that does not cause any problems

- The Monitoring Files job is used to satisfy a start condition even if a file exists before monitoring starts. In this case, there are no problems if a unit whose end is being waited for is delayed and the start of monitoring is delayed because the root jobnet with a start condition starts executing at the same time that monitoring starts.

#### Example of a monitoring delay that causes problems

- The Interval Control job is used to repeatedly start a jobnet at a preset interval from a specific point in time. This case might cause either the start or end of a repetition to be later than the expected time because the repetition start time depends on the time that the unit whose end is being waited for finishes executing.

## (c) Status transitions of units when units with wait conditions are not executed even if a wait condition is satisfied

Note that if the unit with wait conditions and the unit whose end is being waited for are defined or run under the following circumstances, the unit with wait conditions might not run even if the wait condition is satisfied:

- The unit with wait conditions connects to a preceding unit by a relation line
- A hold attribute is set for the unit with wait conditions

- Multiple instances of the unit with wait conditions are set to run concurrently
- You rerun the unit with wait conditions or the unit whose end is being waited for

The following describes the status transitions that take place in these scenarios.

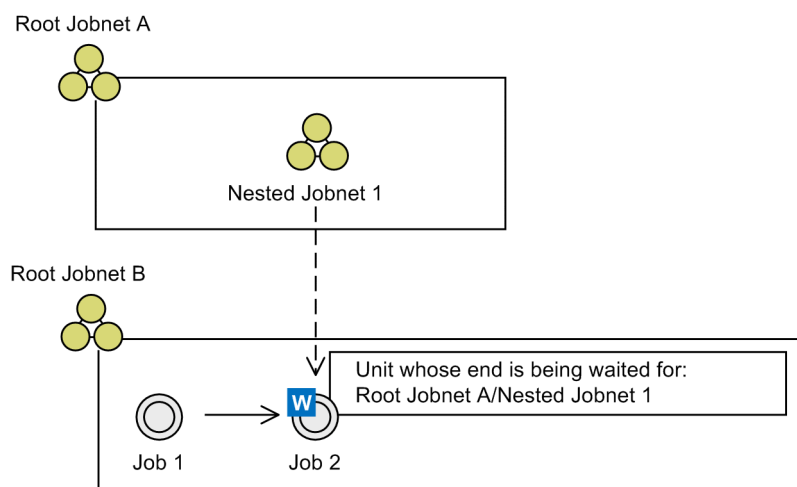
### ■ When the unit with wait conditions connects to a preceding unit by a relation line

When the unit with wait conditions is connected to a preceding unit by a relation line, the unit with wait conditions does not check whether the wait condition is satisfied until the preceding unit has finished executing. If the wait condition is satisfied before the preceding unit finishes, the unit with wait conditions does not start to execute because it does not check whether the wait condition is satisfied.

The following figure shows an example in which a relation line connects the unit with wait conditions to a preceding unit.

Figure 2–45: Example with preceding unit connected by relation line

#### ■ Unit configuration

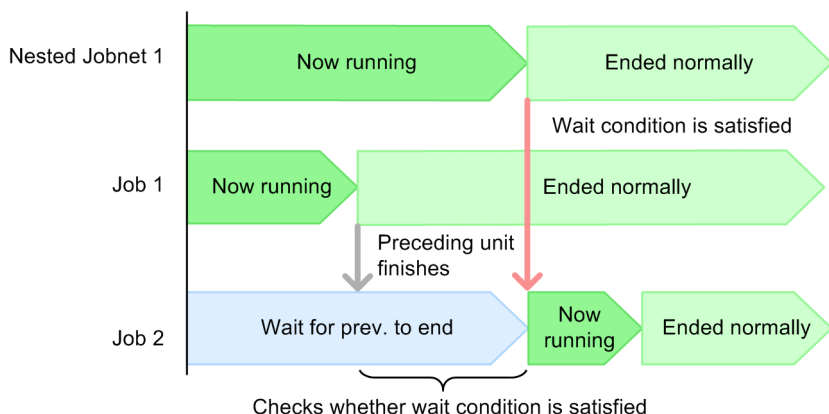


Legend:

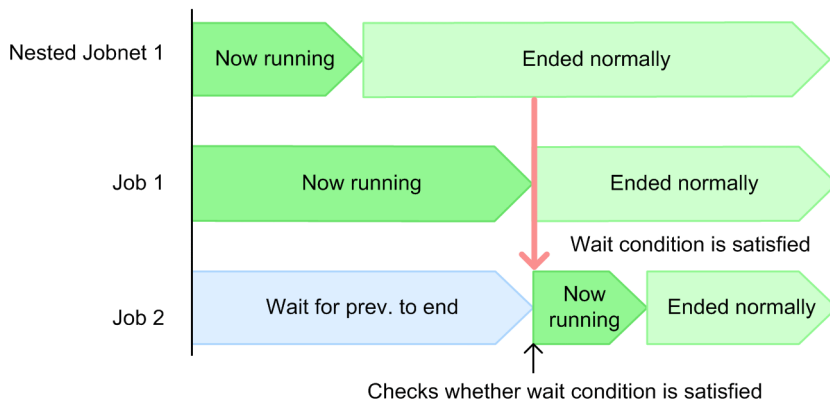
- : Relation line
- > : Flow of wait processing

#### ■ Status transitions

Example 1: The preceding unit finishes executing before the unit whose end is being waited for



Example 2: Preceding unit finishes executing after unit whose end is being waited for



In these examples, Nested Jobnet 1 is defined as the unit whose end is being waited for, for Job 2. Also, Job 1 is defined as a preceding unit of Job 2.

In Example 1, Job 1 finishes executing before Nested Jobnet 1. In this scenario, Job 2 begins to check whether the wait condition is satisfied as soon as Job 1 finishes executing. When Nested Jobnet 1 finishes executing and satisfies the wait condition, Job 2 starts executing.

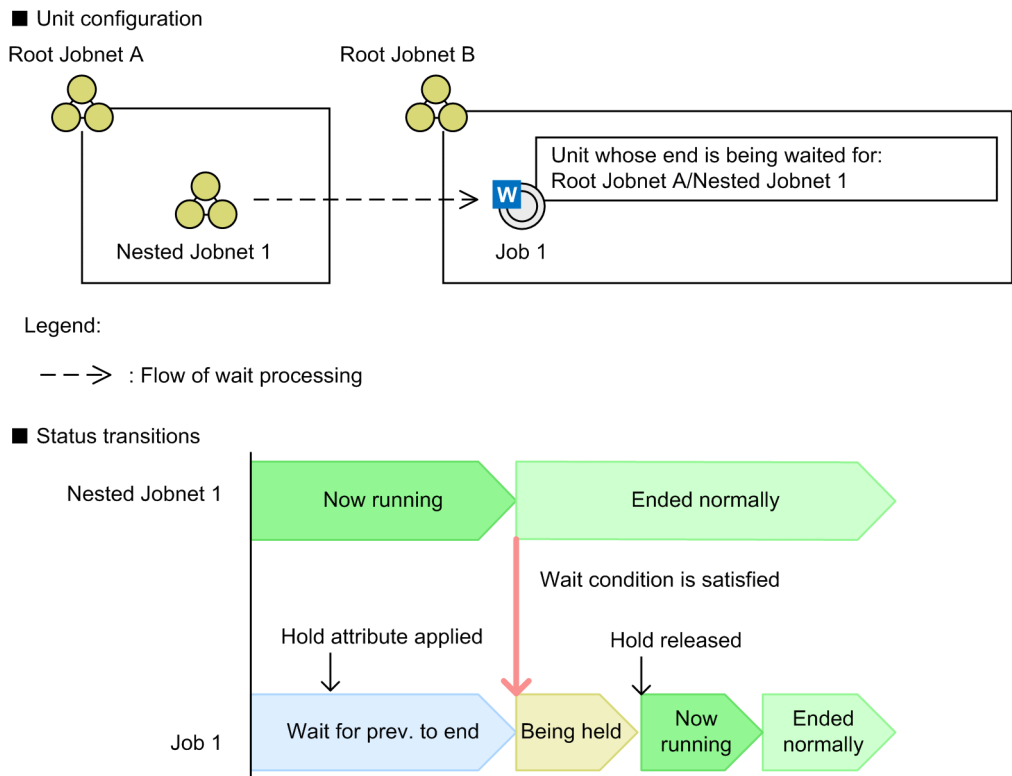
In Example 2, Nested Jobnet 1 finishes executing before Job 1. In this scenario, even if Nested Jobnet 1 ends normally, Job 2 is not executed because it has not begun to check the status of the wait condition. Only after Job 1 ends normally does Job 2 begin to check whether the condition is satisfied. If Nested Jobnet 1 has ended normally by the time Job 2 begins checking the status of the wait condition, the wait condition is satisfied and Job 2 starts immediately.

■ **When a hold attribute is set for the unit with wait conditions**

If a hold attribute is set for the unit with wait conditions, the unit does not start executing when the wait condition is satisfied, instead entering a held state.

The following figure shows an example in which a hold attribute is set.

Figure 2–46: Example with hold attribute set



In this example, a hold attribute is set for Job 1 (the unit with wait conditions) while the job is in *Wait for prev. to end* status. In this scenario, if the wait condition is satisfied, the status of Job 1 changes to *Being held* and the job does not start executing until its held status is released.

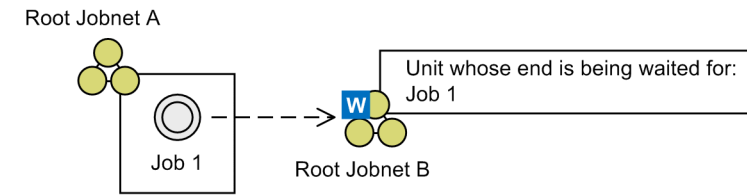
■ **Multiple instances of a unit with wait conditions are run concurrently**

If you run multiple concurrent instances of the unit with wait conditions, depending on the concurrent execution and schedule option settings of the root jobnet, a generation of the unit with wait conditions might wait for the current generation to end despite the wait condition being satisfied, or the next scheduled execution of the root jobnet might enter *Bypassed* status.

The following figure shows examples of the status transitions that apply under different concurrent execution and schedule option settings.

Figure 2–47: Examples of status transitions for different concurrent execution and schedule option settings

■ Unit configuration



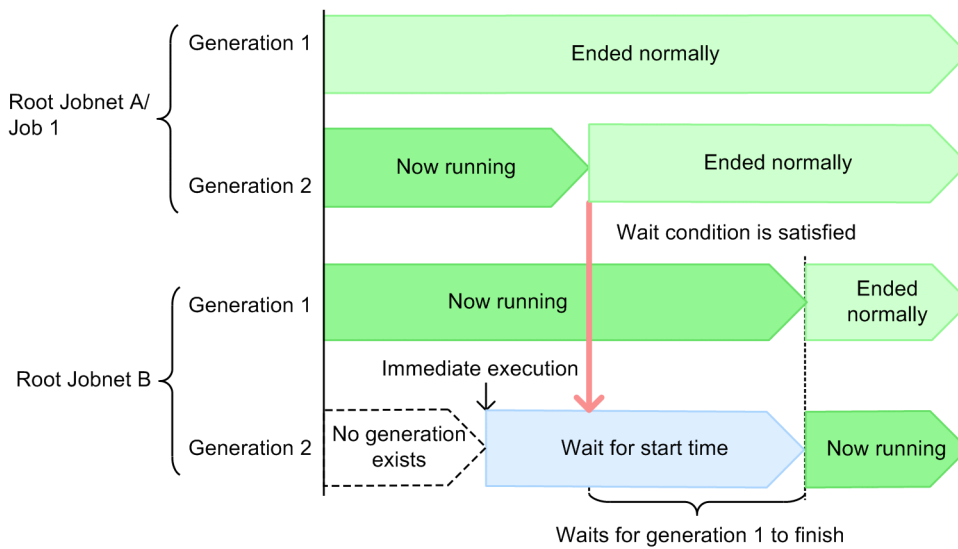
Legend:

--> : Flow of wait processing

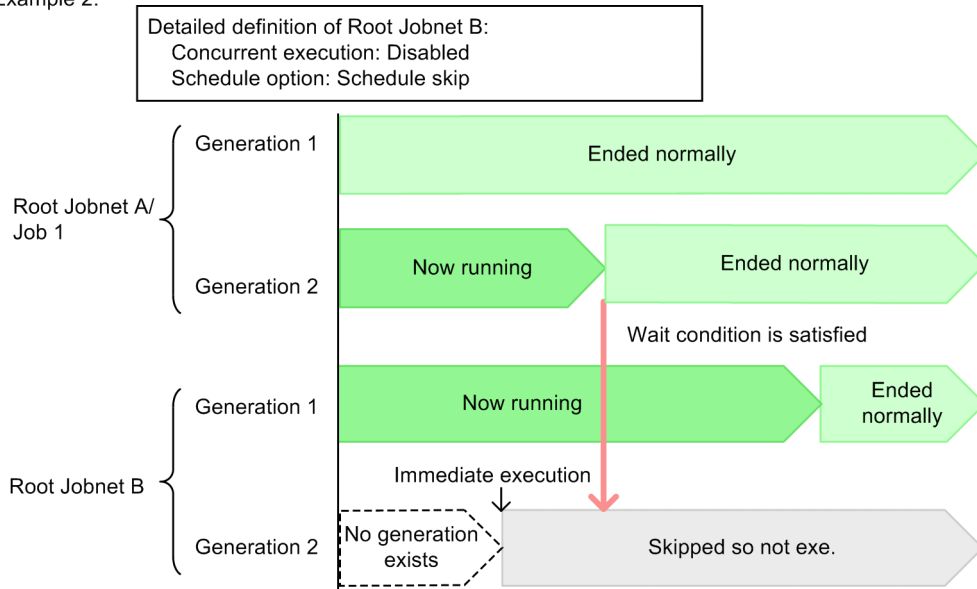
■ Status transitions

Example 1:

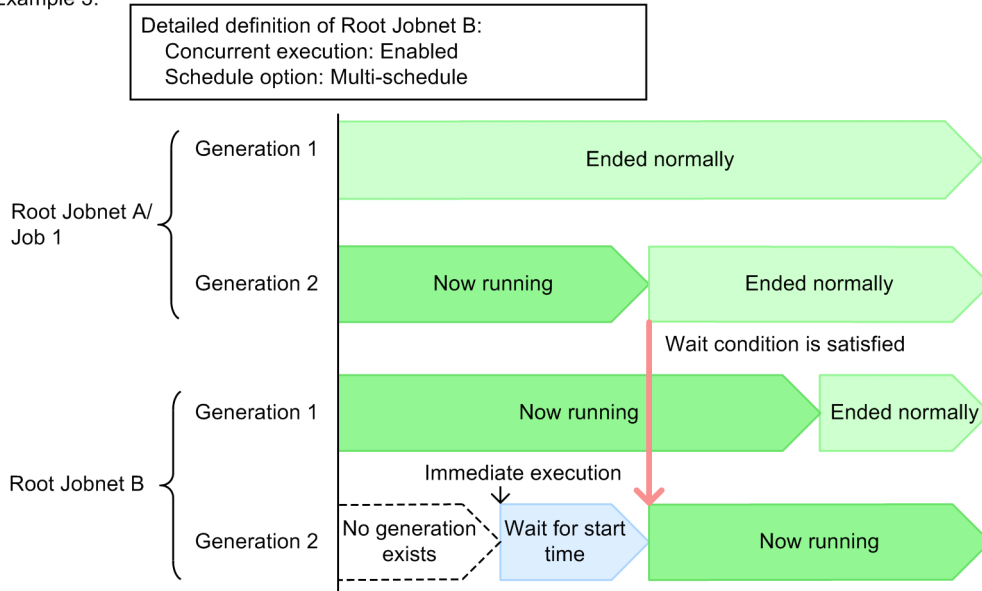
Detailed definition of Root Jobnet B:  
 Concurrent execution: Disabled  
 Schedule option: Multi-schedule



Example 2:



Example 3:



In Example 1, concurrent execution is disabled and multi-schedule is set as the schedule option in the detailed definition of Root Jobnet B (the unit with wait conditions). If Root Jobnet B is registered for immediate execution while the status of Generation 1 of Root Jobnet B is *Now running*, Generation 2 of Root Jobnet B is generated and waits in *Wait for start time* status for Generation 2 of Root Jobnet A/Job 1 to finish executing. When Generation 2 of Job 1 finishes executing and satisfies the wait condition, because concurrent execution is disabled, Generation 2 of Root Jobnet B remains in *Wait for start time* status and waits for Generation 1 of Root Jobnet B to finish executing. Generation 2 of Root Jobnet B starts executing after Generation 1 of Root Jobnet B has finished.

In Example 2, concurrent execution is disabled and schedule skip is set as the schedule option in the detailed definition of Root Jobnet B. If Root Jobnet B is registered for immediate execution while the status of Generation 1 of Root Jobnet B is *Now running*, Generation 2 of Root Jobnet B is generated. However, because Generation 1 is still running, Generation 2 is skipped and enters *Skipped so not exe.* status. Subsequently, even if Generation 2 of Job 1 of Root Jobnet A ends normally, Generation 2 of Root Jobnet B does not start executing because it has already been skipped.

In Example 3, concurrent execution is enabled and multi-schedule is set as the schedule option in the detailed definition of Root Jobnet B. If Root Jobnet B is registered for immediate execution while the status of Generation 1 of Root Jobnet



B is *Now running*, Generation 2 of Root Jobnet B is generated and waits in *Wait for start time* status for Generation 2 of Root Jobnet A/Job 1 to finish executing. When Generation 2 of Job 1 finishes executing and satisfies the wait condition, Generation 2 of Root Jobnet B begins executing.

**Cautionary note**

When assigning wait conditions for the root jobnet, make sure that the wait conditions for the previous execution schedule are satisfied and that the unit begins to execute before the subsequent execution start time arrives. Even when concurrent execution is enabled and multi-schedule is set, if the previous execution schedule has not yet started, execution will not start when the wait conditions are satisfied.

■ **Rerunning units with wait conditions or units whose ends are being waited for**

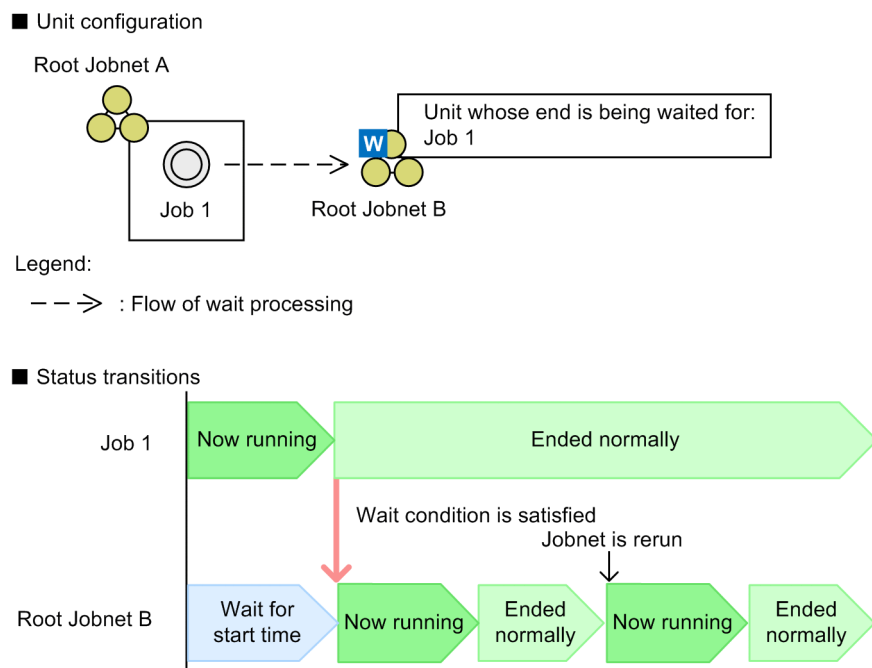
The following describes the status transitions that occur when you rerun a unit with wait conditions or a unit whose end is being waited for.

**Behavior when a unit with wait conditions is rerun**

If you rerun a unit whose wait condition is already satisfied, the wait condition does not revert to an unsatisfied state. If you rerun a unit that finished executing after its wait condition was satisfied, the unit enters a waiting state. However, because the wait condition is already satisfied, the unit then starts executing immediately.

The following figure shows an example of the status transitions that occur when a unit with wait conditions is rerun.

**Figure 2–48: Example of rerunning a unit with wait conditions**



In this example, Root Jobnet B is re-executed after Job 1 (the unit whose end is being waited for) has ended normally and satisfied the wait condition. When Root Jobnet B is rerun, it starts executing immediately without waiting for Job 1 to finish because the wait condition is already satisfied.

**Supplementary note**

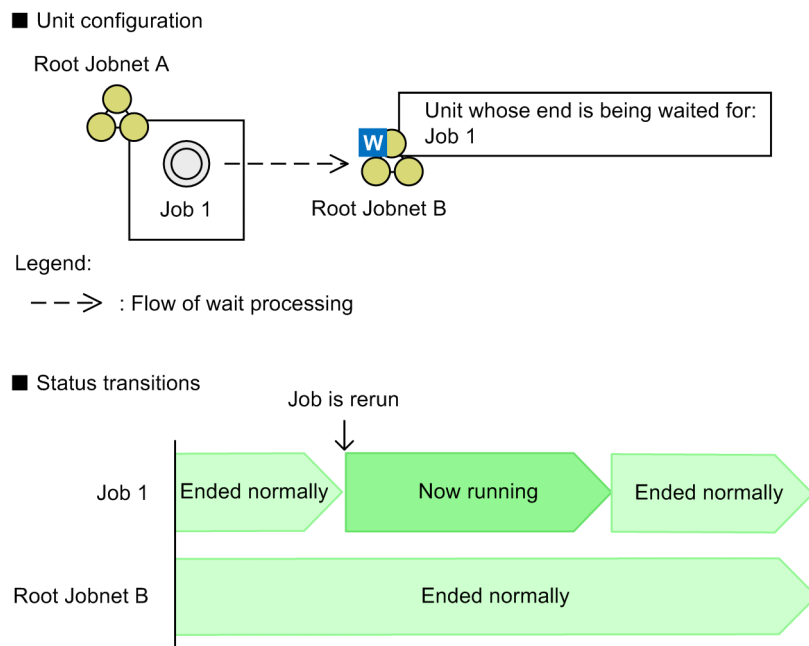
When you rerun a unit that has wait conditions and is not scheduled for execution if `yes` is specified for the `PREWAITNOSCHUNITS` environment setting parameter, the unit enters the *Bypassed* status and the succeeding unit runs.

**Behavior when unit whose end is being waited for is rerun**

When you rerun a unit whose end is being waited for, only the unit whose end is being waited for is rerun. The unit with the wait condition remains unaffected.

The following figure shows an example of the status transitions that occur when a unit whose end is being waited for is rerun.

Figure 2–49: Example of rerunning a unit whose end is being waited for



In this example, Job 1 (the unit whose end is being waited for) is rerun after Job 1 and Root Jobnet B (the unit with the wait condition) have both ended normally. In this scenario, Job 1 starts executing, but Root Jobnet B remains in *Ended normally* and does not restart.

**Cautionary note**

When you use a start condition for a unit whose end is being waited for, do not re-execute the monitoring generation. Re-executing the monitoring generation will not start monitoring again, and might unexpectedly cause a wait condition to be satisfied. If the monitoring generation has ended in the *Monitor terminated* or *Unmonitored + Ended* status when no wait condition is satisfied, disable the wait condition to start execution of the unit with the wait condition. For details about making temporary changes to wait conditions, see 4.5.15 *Temporarily changing the wait condition settings for a jobnet or job* in the manual *JP1/Automatic Job Management System 3 Overview*.

**(d) Treatment of a wait condition when the succeeding unit of a unit, whose end is being waited for, is rerun**

The status of a unit whose end is being waited for by the succeeding unit might change when the succeeding unit is rerun. By using the `PREWAITRERUNSTATUS` environment setting parameter, whether to meet the wait condition can be controlled based on this status change. In a newly installed JP1/AJS3, the environment setting parameter is initially set to meet the wait condition.

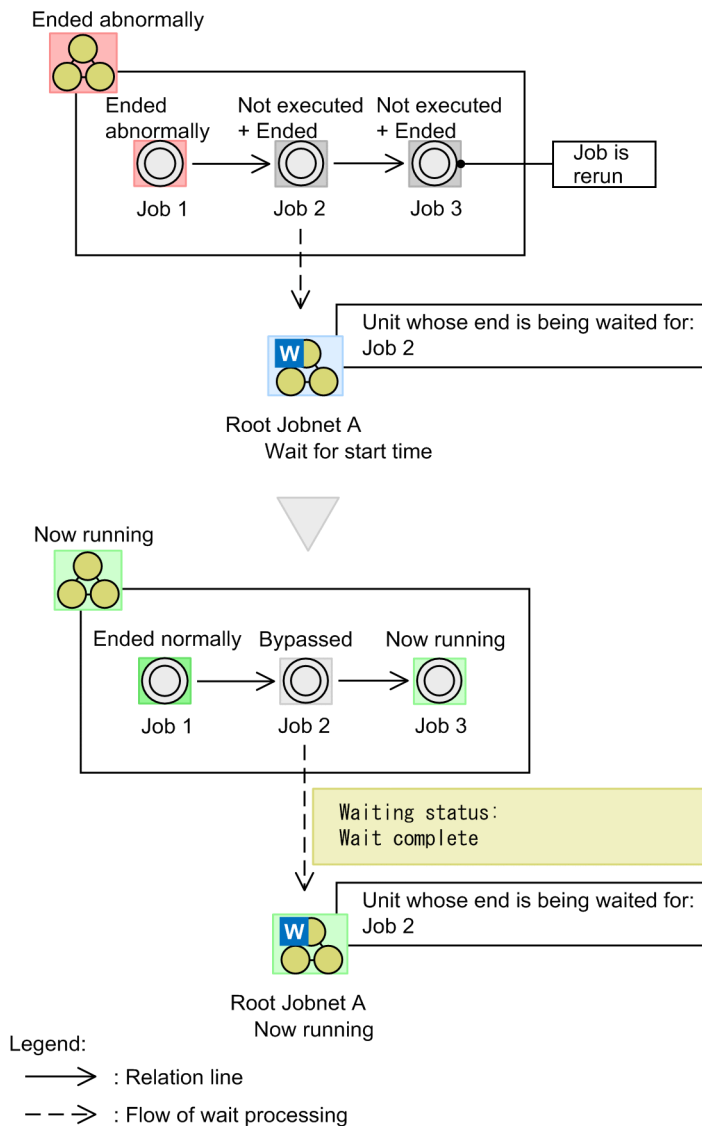
For details about the `PREWAITRERUNSTATUS` environment setting parameter, see 20.4.2(110) `PREWAITRERUNSTATUS` in the *JP1/Automatic Job Management System 3 Configuration Guide*.

**Configuring so that the wait condition is met:**

If the `PREWAITRERUNSTATUS` environment setting parameter is set to `no` and running the succeeding unit changes the status of the unit whose end is being waited for, the wait condition is met and the unit with wait conditions then starts running.

The following figure shows an example of rerunning the succeeding job of a unit whose end is being waited for, while the `PREWAITRERUNSTATUS` environment setting parameter is set to `no`.

Figure 2–50: Example of rerunning a succeeding unit of a unit whose end is being waited for (configuring so that the wait condition is met)



In this example, Job 2 is the unit whose end is being waited for and Job 3 is a succeeding unit of Job 2. If Job 3 is re-executed while both Job 3 and Job 2 are in *Not executed + Ended* status, the status of Job 2 changes from *Not executed + Ended* to *Bypassed*.

If the `PREWAITRERUNSTATUS` environment setting parameter is set to `no`, the wait status changes to *Wait complete*, causing the wait condition to be met. As a result, root jobnet A, which is a unit with wait conditions, starts running.

If the `PREWAITRERUNSTATUS` environment setting parameter is set to `no`, be careful if rerunning of the succeeding unit changes the status of a unit whose end is being waited for. If you intend to rerun a succeeding unit of a unit whose end is being waited for, make sure that you are fully aware of the effects such an action will have. You can do so by conducting a search to make sure that the unit whose end is being waited for is not one of the units whose status will be affected. For details on rerunning a unit, see 4.5.11 *Rerunning a job or jobnet* in the manual *JPI/Automatic Job Management System 3 Overview*. For details on how to search for wait conditions, see 10.2 *Finding and displaying units* in the *JPI/Automatic Job Management System 3 Operator's Guide*.

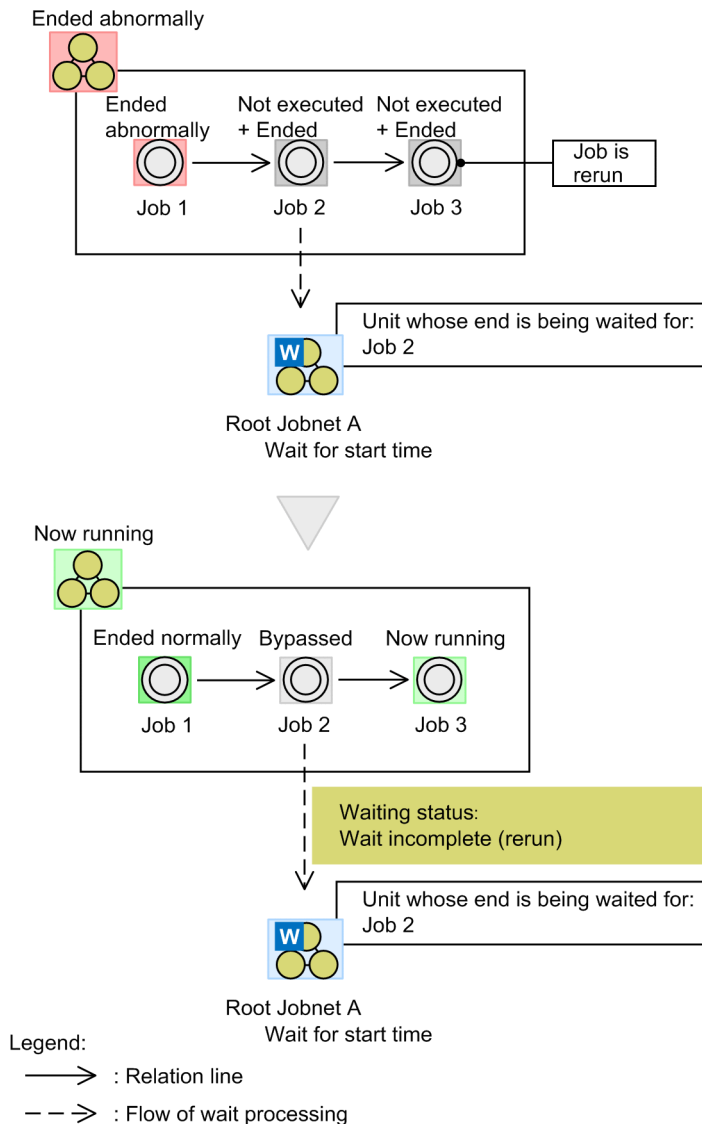
Configuring so that wait condition is not met:

If the `PREWAITRERUNSTATUS` environment setting parameter is set to `yes`, the wait condition is not met when the status of a unit whose end is being waited for is changed by rerunning the succeeding unit. Therefore, the status of the unit with wait conditions does not change.

This setting can prevent a wait condition from being unintentionally met when the status of the preceding unit is changed by rerunning a unit.

The following figure shows an example of rerunning the succeeding job of a unit whose end is being waited for if the `PREWAITRERUNSTATUS` environment setting parameter is set to `yes`.

Figure 2–51: Example of rerunning a succeeding unit of a unit whose end is being waited for (configuring so that the wait condition is not met)



The unit definitions in the above example are the same as those in the example in the case where the wait condition is met. When job 3 is rerun, the status of job 2 changes from *Not executed + Ended* to *Bypassed*.

However, if the `PREWAITRERUNSTATUS` environment setting parameter is set to `yes`, the wait condition changes to *Wait incomplete*. As a result, the wait condition is not met. Therefore, the status of root jobnet A, which is a unit with wait conditions, does not change.

Note that if job 2, which is a unit whose end is being waited for, is re-executed and the execution ends, the wait condition is met, causing root jobnet A to start running.

If the unit whose end is being waited for is a jobnet and subordinate units are individually rerun, a wait is completed when the jobnet ends. If necessary, set the hold attribute or change the job status so that the wait condition is not met.

## (5) Configuring the behavior of units with wait conditions

In addition to the name of the unit whose end is being waited for, the following attributes can be set for a wait condition:

- The wait method for conditions with multiple units whose ends are being waited for
- How units with wait conditions behave when there are no generations to wait for
- How units with wait conditions behave when start conditions are used for the units whose ends are being waited for

These attributes are described below.

### **(a) Wait methods for conditions with multiple units whose ends are being waited for**

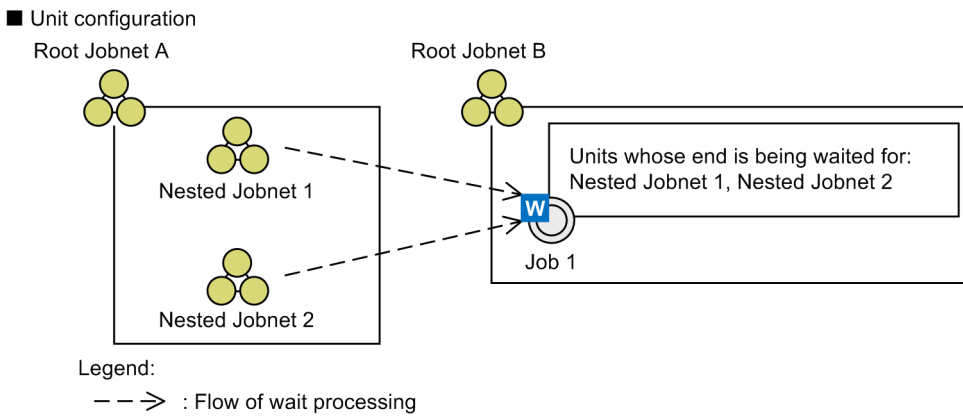
You can specify, for each unit with wait conditions, a maximum of 32 units whose ends are being waited for. If you specify more than one unit whose end is being waited for, you can specify whether the condition is satisfied when all of the units whose ends are being waited for finish executing, or just one.

You can choose either of the following wait methods:

- AND  
The wait condition is satisfied when all specified units whose ends are being waited for finish executing.
- OR  
The wait condition is satisfied when any one of the specified units whose ends are being waited for finishes executing.

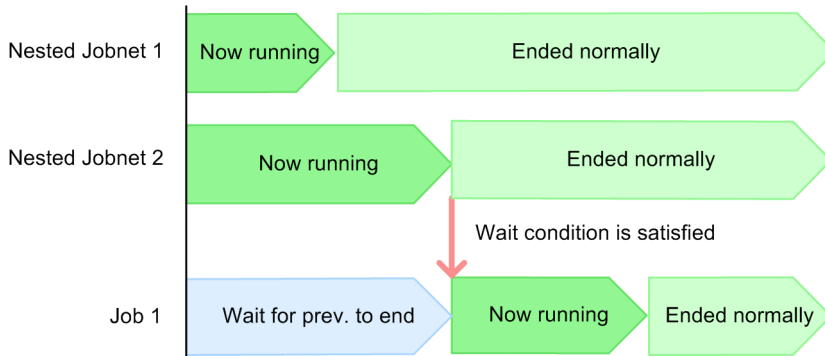
The following figure shows the difference between the two wait methods.

Figure 2–52: Difference between AND and OR wait methods

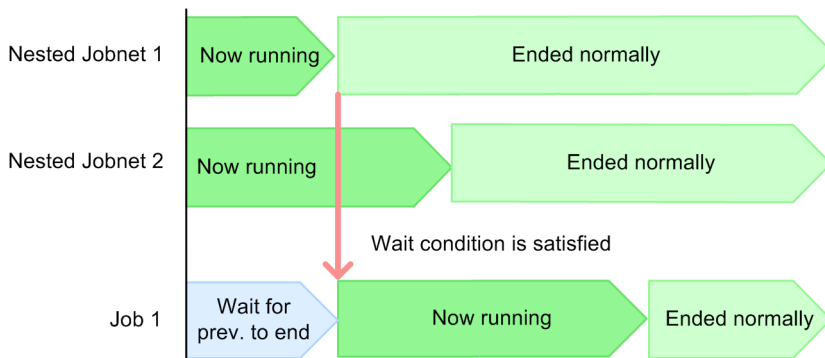


■ Status transitions

Example 1: AND wait method



Example 2: OR wait method



In these examples, Nested Jobnet 1 and Nested Jobnet 2 are specified for Job 1, the unit with wait conditions, as the units whose ends are being waited for.

In Example 1, which uses the AND wait method, the condition is satisfied and Job 1 starts executing when Nested Jobnet 1 and Nested Jobnet 2 have both finished executing.

In Example 2, which uses the OR wait method, the condition is satisfied and Job 1 starts executing when Nested Jobnet 1 finishes executing.

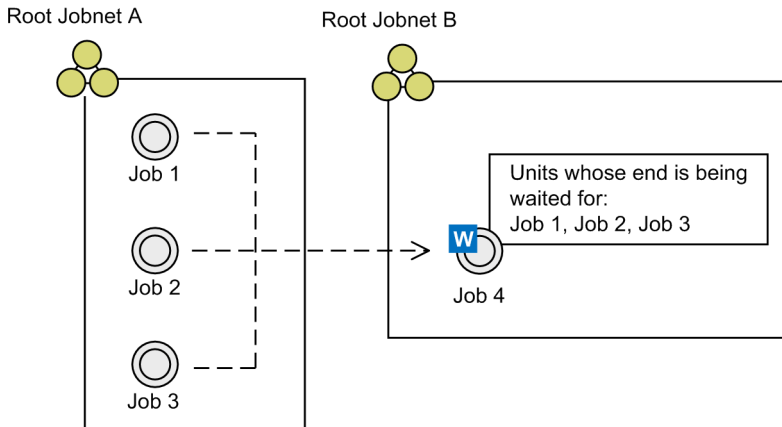
Cautionary note

Assigning a large number of wait conditions to a single unit can slow down the processing speed of the JP1/AJS3 system. When designing a unit with wait conditions, try to use as few units whose ends are being waited for as possible.

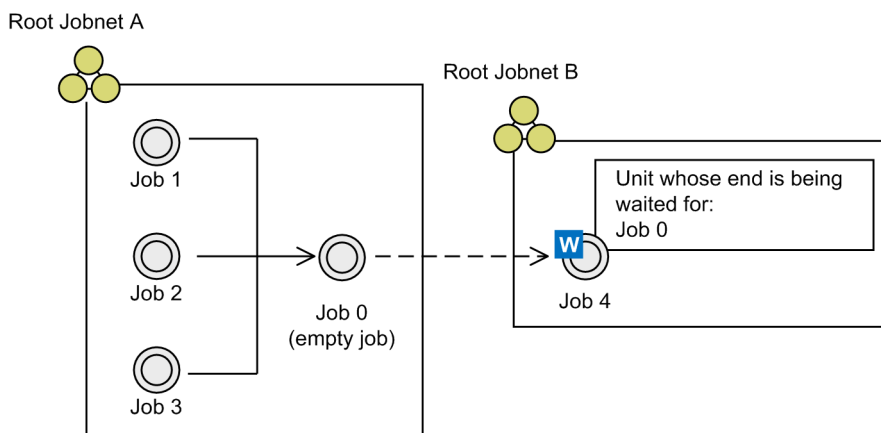
The following figure shows an example of how you can reduce the number of units whose ends are being waited for.

Figure 2–53: Example of reducing the number of units whose ends are being waited for

■ Example of job flow with more units whose end is being waited for



■ Example with fewer units whose end is being waited for



Legend:

→ : Relation line

- - → : Flow of wait processing

In the example with the greater number of units whose ends are being waited for, Job 1, Job 2, and Job 3 are specified for job 4 (the unit with wait conditions) as the units whose ends are being waited for. In this example, the number of units whose ends are being waited for is reduced by defining an empty job (Job 0) as a succeeding unit of Jobs 1, 2, and 3, and connecting each of Jobs 1, 2, and 3 to Job 0 by relation lines. For Job 4, you can then specify Job 0 as the unit whose end is being waited for, achieving the same results with fewer units whose ends are being waited for. However, defining empty jobs like the one pictured above can complicate job management. For this reason, we recommend that you avoid defining superfluous units unless performance considerations demand it.

Supplementary note

When using the AND wait method, if you perform a hot restart of the scheduler service after some of the units whose ends are being waited for have finished executing, the status of the wait condition is retained after the scheduler service restarts. This means that the wait condition will be satisfied when the remaining units whose ends are being waited for finish executing.

For example, suppose that in Example 1 of *Figure 2-52* you perform a hot restart of the scheduler service after Nested Jobnet 1 has ended normally and Nested Jobnet 2 is running. In this scenario, the wait condition is satisfied when Nested Jobnet 2 finishes executing after the scheduler service restarts.

## (b) Behavior of units with wait conditions when there are no applicable generations of units whose ends are being waited for

Units with wait conditions wait for specific generations of their units whose end is being waited for to finish executing, according to the rules described in (2) *Rules governing waiting regarding units with wait conditions and units whose ends are being waited for*. However, if there are no generations of the unit whose end is being waited for that satisfy the wait rule, the unit with the wait condition can be configured to behave in either of the following ways:

- Do not start executing

When there are no applicable generations of the unit whose end is being waited for, the unit with the wait condition continues waiting for the wait condition to be satisfied with a status of either *Wait for start time*, *Wait for prev. to end*, or *Not sched. to exe.*<sup>#</sup>.

#

If `yes` is specified for the `PREWAITNOSCHUNITS` environment setting parameter, the unit with wait conditions performs a wait even if the unit is in the *Not sched. to exe.* status. For details about the `PREWAITNOSCHUNITS` environment setting parameter, see 20.4.2(122) `PREWAITNOSCHUNITS` in the *JPI/Automatic Job Management System 3 Configuration Guide*.

- Start executing

When there are no applicable generations of the unit whose end is being waited for, the unit with the wait condition immediately starts executing. If the wait condition applies to multiple units whose ends are being waited for, the unit starts executing when there are no applicable generations of any units whose ends are being waited for that have not finished executing.

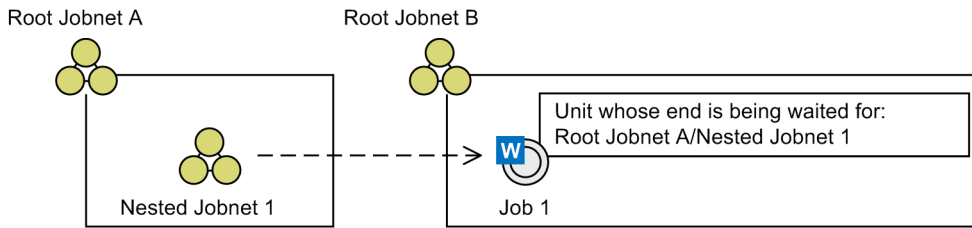
This setting applies when the unit with wait conditions and the unit whose end is being waited for are associated with different root jobnets. If both units are under the same root jobnet, they run as part of the same generation, which means there is always a generation of the unit whose end is being waited for that satisfies the wait rule.

The following figure shows the differences between the settings that govern how wait conditions behave when there are no applicable generations of units whose ends are being waited for.



Figure 2–54: Differences in behavior when there are no settings specifying applicable generations of units whose ends are being waited for

■ Unit configuration

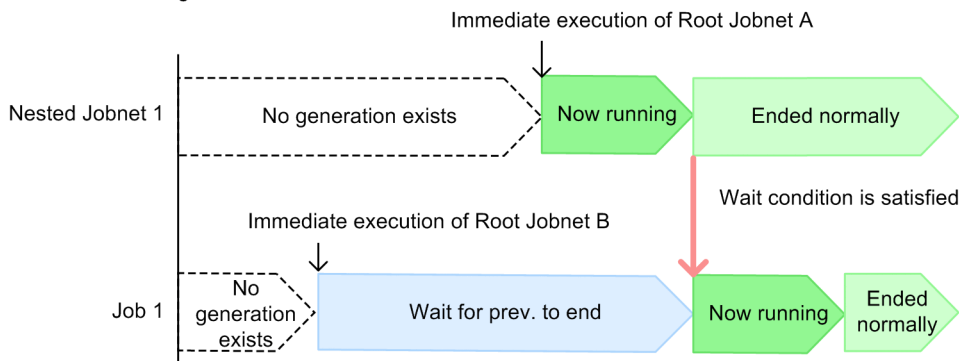


Legend:

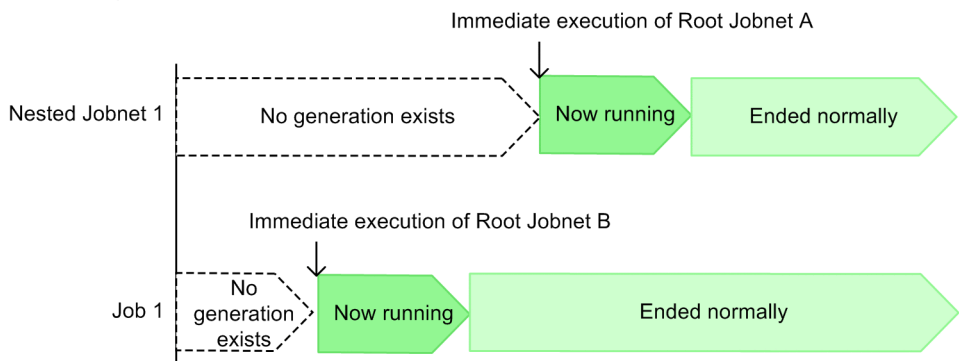
--> : Flow of wait processing

■ Status transitions

Example 1: Setting for when there are no applicable generations of units whose end is being waited for: Do not start executing



Example 2: Setting for when there are no applicable generations of units whose end is being waited for: Start executing



In Example 1, the setting for when there are no applicable generations of units whose ends are being waited for is *Do not start executing*. When Root Jobnet B, an upper-level unit of Job 1 (the unit with wait conditions), is registered for immediate execution, a generation of Job 1 is generated. Because there is no generation of Nested Jobnet 1 (the unit whose end is being waited for), Job 1 enters *Wait for prev. to end* status and waits for the wait condition to be satisfied. If Root Jobnet A, an upper-level unit of Nested Jobnet 1, is registered for immediate execution, and then Nested Jobnet 1 ends normally, the wait condition is satisfied and Job 1 starts executing.

In Example 2, the setting for when there are no applicable generations of units whose ends are being waited for is *Start executing*. When Root Jobnet B is registered for immediate execution, a generation of Job 1 is generated. Because no generation of Nested Jobnet 1 exists at this time, Job 1 immediately starts executing. If Root Jobnet A, an upper-level unit of Nested Jobnet 1, is registered for immediate execution after Job 1 has already started executing, a normal termination of Nested Jobnet 1 will not trigger another generation of Job 1.

## Supplementary notes

- If the status of the unit whose end is being waited for is *Not sched. to exe.*, the setting governing behavior when no applicable generations are present does not apply if the root jobnet associated with the unit whose end is being waited for has an execution schedule. In this scenario, the unit whose end is being waited for enters *Bypassed* status when its root jobnet is executed. This causes the unit whose end is being waited for to be seen as having finished executing, which satisfies the wait condition.
- If the status of the root jobnet of the unit whose end is being waited for is *Not registered*, the *Start executing* setting does not take effect and the unit with wait conditions does not start to execute.

### (c) Behavior of units with wait conditions when start conditions are used for the units whose ends are being waited for

When you use a start condition for a unit whose end is being waited for, you can change the wait condition to change the status the unit whose end is being waited for must have so it can finish executing.

- When the monitoring generation of the unit whose end is being waited for ends in the *Unmonitored + Ended* status  
The wait condition can be satisfied when the monitoring generation ends in the *Unmonitored + Ended* status.
- When an execution generation of the unit whose end is being waited for ends abnormally  
The wait condition can be satisfied when an execution generation ends abnormally.  
The wait condition can also be satisfied when the abnormal termination status of an execution generation is the *Skipped so not exe.*

The following examples describe how you can change a wait condition:

- If you want a unit to end without satisfying a start condition during event monitoring  
If you want operation to be treated as normal even if no start condition at all is satisfied during the time period specified as the valid range for the start condition, you can cause the wait condition to be satisfied when the monitoring generation enters the *Unmonitored + Ended* status.
- If you want operation to continue when an execution generation enters the *Ended abnormally* or *Killed* status  
When a start condition is satisfied multiple times, if you want operation to be treated as normal even if some of generated execution generations end abnormally, you can cause the wait condition to be satisfied when execution generations end abnormally.
- If generations waiting to be executed are set not to be retained even when many start conditions are to be satisfied while the monitoring generation is in the *Now monitoring* status  
If execution generations waiting to be executed are set not to be retained even when many start conditions are satisfied, the execution generations that have not started executing enter the *Skipped so not exe.* status. If you want operation to be treated as normal even when execution generations enter the *Skipped so not exe.* status, you can cause the wait condition to be satisfied when execution generations enter the *Skipped so not exe.* status.

#### ■ Configuring the end of execution for the monitoring generation when a unit whose end is being waited for uses a start condition

When a unit whose end is being waited for uses a start condition, you can set that the monitoring generation transitions to the *Unmonitored + Ended* status as the end of execution of the monitoring generation.

In the Waiting Condition Settings dialog box, in the **When Unmonitored + Ended** section, choose either of the following:

- **Do not execute** (default)  
The unit with a wait condition does not start executing even if the monitoring generation enters the *Unmonitored + Ended* status.

- **Execute**

The unit with a wait condition starts executing when the monitoring generation enters the *Unmonitored + Ended* status.

The following table describes how to end the execution of a root jobnet with a start condition by using these options.

**Table 2–13: Settings in the When Unmonitored + Ended section and statuses that end execution of a root jobnet with a start condition**

| Do not execute (default)  | Execute  |
|---|--|
| Monitoring generation <ul style="list-style-type: none"> <li>• Monitor-end normal</li> </ul> Execution generation <sup>#</sup> <ul style="list-style-type: none"> <li>• Ended normally</li> <li>• Ended with warning</li> </ul> | Monitoring generation <ul style="list-style-type: none"> <li>• Monitor-end normal</li> <li>• Unmonitored + Ended</li> </ul> Execution generation <sup>#</sup> <ul style="list-style-type: none"> <li>• Ended normally</li> <li>• Ended with warning</li> </ul> |

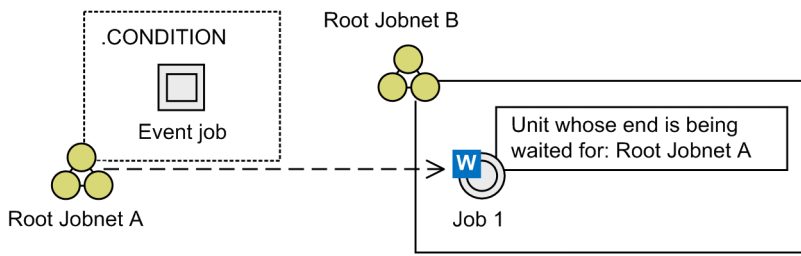
#

You can also change the status an execution generation must have to end its execution by changing the behavior of the unit with a wait condition. This table assumes that the default settings are used.

The following figure explains the difference in the behavior of units with wait conditions resulting from the option chosen in the **When Unmonitored + Ended** section.

Figure 2–55: Difference in the behavior of units with wait conditions resulting from the option chosen in the **When Unmonitored + Ended** section

■ Unit configuration

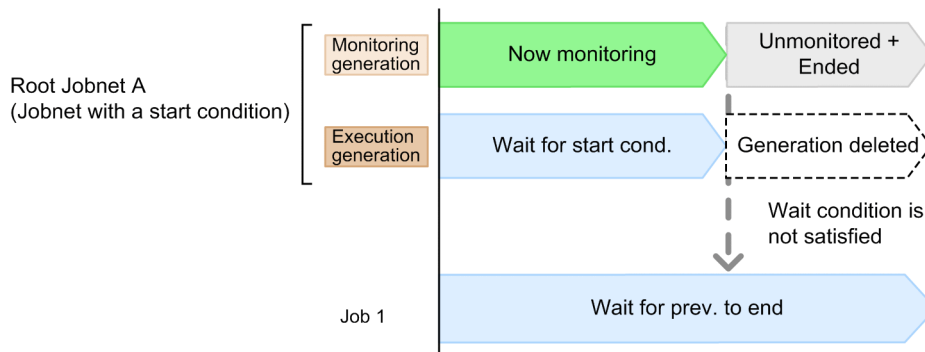


Legend:

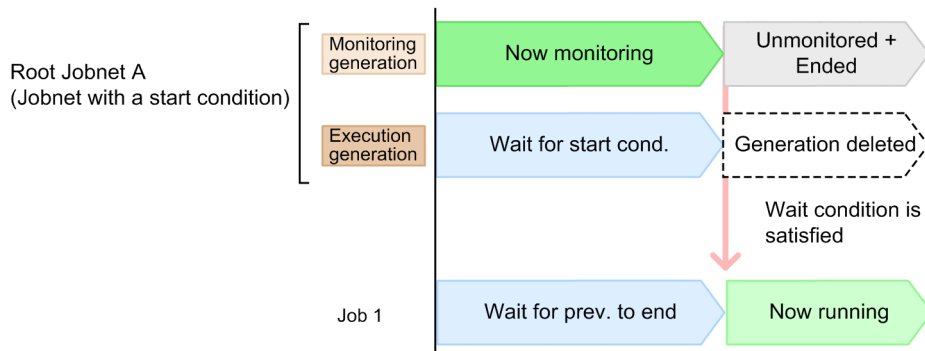
--> : Flow of wait processing

■ Status transitions

Example 1: When **Do not execute** is chosen in the **When Unmonitored + Ended** section (default)



Example 2: When **Execute** is chosen in the **When Unmonitored + Ended** section



In example 1, **Do not execute** is chosen in the **When Unmonitored + Ended** section for the wait condition defined for Job 1. If the start condition is never satisfied while the monitoring generation of Root Jobnet A (the unit whose end is being waited for) is in the *Now monitoring* status, the monitoring generation transitions to the *Unmonitored + Ended* status. Accordingly, Job 1 (the unit with the wait condition) continues to wait in the *Wait for prev. to end* status for the wait condition to be satisfied.

In example 2, **Execute** is chosen in the **When Unmonitored + Ended** section for the wait condition defined for Job 1. If the start condition is never satisfied while the monitoring generation of Root Jobnet A is in the *Now monitoring* status, the monitoring generation transitions to the *Unmonitored + Ended* status. With the transition, the wait condition is satisfied and Job 1 (the unit with the wait condition) starts executing.

■ **Configuring the end of execution for an execution generation when a unit whose end is being waited for uses a start condition**

When a unit whose end is being waited for uses a start condition, you can set that an execution generation transitions to the *Skipped so not exe.* status or any end status, including abnormal termination, as the end of execution of the execution generation.

In the Waiting Condition Settings dialog box, in the **Abnormal end for an execution generation** section, choose either of the following:

- **Do not execute** (default)

If an execution generation ends abnormally, the unit with a wait condition does not start executing. However, if the execution generation is in the *Skipped so not exe.* status, you can choose whether to start execution of the unit. By default, the unit with a wait condition does not start executing when an execution generation is in the *Skipped so not exe.* status.

- **Execute**

Even if an execution generation ends abnormally, the unit with a wait condition starts executing. When you choose this option, all types of end statuses are considered to be the end of execution.

The following table describes how to end the execution of a root jobnet with a start condition by using these options.

Table 2–14: Settings in the Abnormal end for an execution generation section and statuses that end execution of a root jobnet with a start condition

| Do not execute   |   | Execute   |
|--|---|---|
| Do not start execution even if the execution generation is in the Skipped so not exe. status (default)             | Start execution when the execution generation is in the Skipped so not exe. status  |   |
| Monitoring generation#<br>• Monitor-end normal<br>Execution generation<br>• Ended normally<br>• Ended with warning | Monitoring generation#<br>• Monitor-end normal<br>Execution generation<br>• Ended normally<br>• Ended with warning<br>• Skipped so not exe. | Monitoring generation#<br>• Monitor-end normal<br>Execution generation<br>• All types of end statuses |

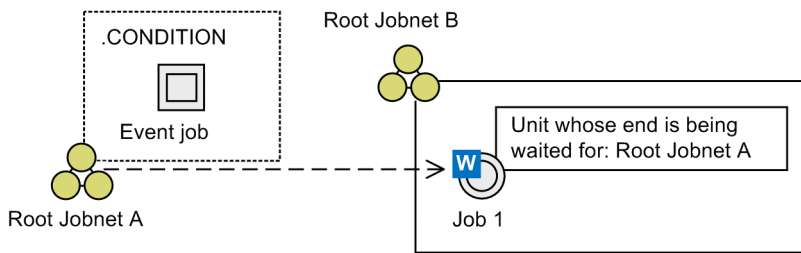
#

You can also change the status the monitoring generation must have to end its execution by changing the behavior of the unit with a wait condition. This table assumes that the default settings are used.

The following figure shows the difference in the behavior of units with wait conditions resulting from the option chosen in the **Abnormal end for an execution generation** section.

Figure 2–56: Difference in the behavior of units with wait conditions resulting from the option chosen in the Abnormal end for an execution generation section

■ Unit configuration

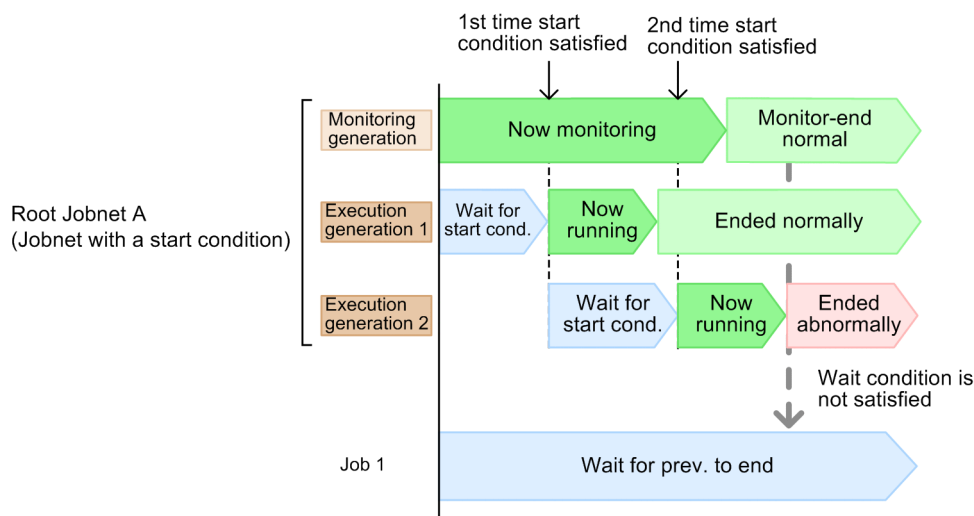


Legend:

--> : Flow of wait processing

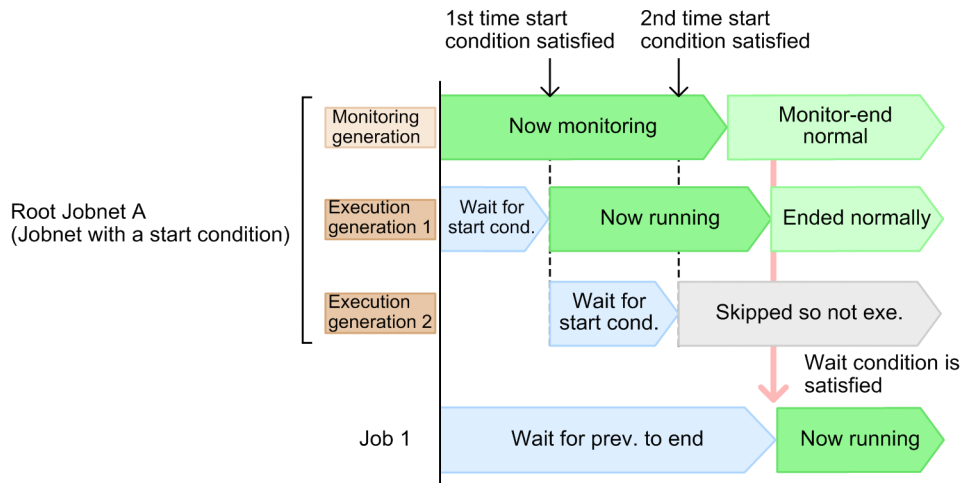
■ Status transitions

Example 1: When **Do not execute** is chosen in the **Abnormal end for an execution generation** section



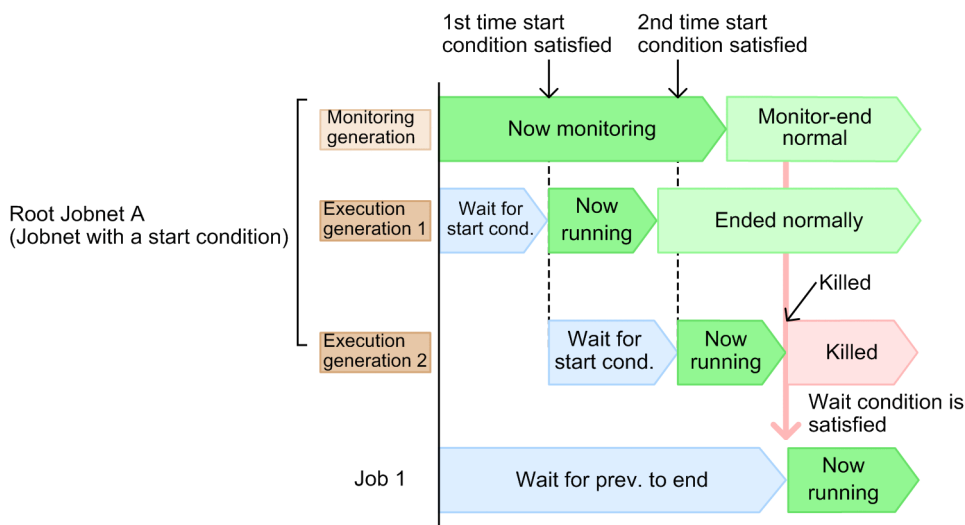
In example 1, **Do not execute** is chosen in the **Abnormal end for an execution generation** section for the wait condition defined for Job 1. When execution generation 1 ends normally and execution generation 2 ends abnormally, Job 1 (the unit with a wait condition) continues to wait in the *Wait for prev. to end* status for the wait condition to be satisfied.

Example 2: When the unit is executed while **Do not execute** is chosen in the **Abnormal end for an execution generation** section and the execution generation is in the *Skipped so not exe.* status



In example 2, **Do not execute** is chosen in the **Abnormal end for an execution generation** section for the wait condition defined for Job 1. However, the unit starts executing when the execution generation is in the *Skipped so not exe.* status. The execution generations of Root Jobnet A are not retained. As a result, if the start condition is satisfied while execution generation 1 is executing and execution generation 2 enters the *Skipped so not exe.* status, the wait condition is satisfied when execution generation 1 enters the *Ended normally* status.

Example 3: When **Execute** is chosen in the **Abnormal end for an execution generation** section



In example 3, **Execute** is chosen in the **Abnormal end for an execution generation** section for the wait condition defined for Job 1. If execution generation 2 is forced to end and enters the *Killed* status while it is executing, the wait condition is satisfied.

#### Cautionary note

The option chosen in the **Abnormal end for an execution generation** section is applied to saved generations. For example, if some saved execution generations have been deleted because the number of logs to keep is exceeded, whether the wait condition is satisfied is determined starting with the remaining execution generations. In other words, even if **Do not execute** is chosen in the **Abnormal end for an execution generation** section, the wait condition is satisfied when all the abnormally-ended execution generations are deleted and all the remaining generations have ended normally.

When you specify a root jobnet with a start condition as the unit whose end is being waited for, estimate the number of logs to keep after taking into consideration the number of execution generations that will be started by the start condition. This precaution ensures that saved execution generations will not be deleted while waiting.

For details about the number of logs to keep for root jobnets with start conditions, see 4.2.3(3) *Examples of managing saved generations for a jobnet with a start condition* in the manual *JPI/Automatic Job Management System 3 Overview* and 7.2 *Relationship between number of logs to keep and performance*. You can then determine the settings.

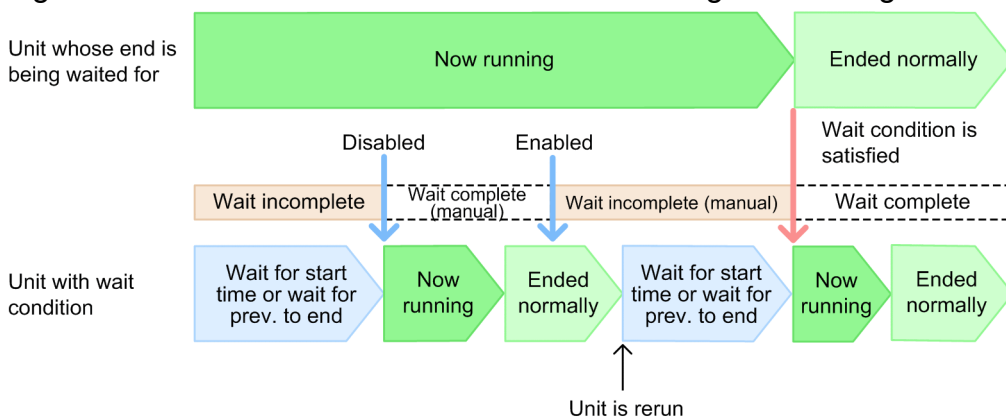
## (6) Temporarily changing a wait condition

You can temporarily enable or disable the waiting process with respect to a unit whose end is being waited for specified in the wait conditions.

When you enable waiting for a unit with wait conditions that has finished executing, its status changes to *Wait incomplete (manual)*. This unit will resume the waiting process if you rerun it. If you disable waiting for a unit that is waiting for a unit whose end is being waited for to finish executing, its status changes to *Wait complete (manual)*. If you change the status to *Wait complete (manual)* for every unit whose end is being waited for that is subject to an AND wait method, the unit with wait conditions will immediately start executing. If the wait method is OR, the unit with wait conditions will start executing if you change the waiting status of any one of the units whose ends are being waited for to *Wait complete (manual)*.

The following figure shows how the wait status is affected by enabling or disabling a wait condition.

Figure 2–57: Effect on wait statuses from enabling or disabling wait conditions



The wait status is *Wait incomplete* while the unit with wait conditions is in *Wait for start time*, *Wait for prev. to end*, or *Not sched. to exe.*<sup>#</sup> status and the unit whose end is being waited for is running. If you disable waiting, the wait status changes to *Wait complete (manual)* and the unit with wait conditions starts executing. If you enable waiting after the unit with wait conditions has finished executing, the wait status changes to *Wait incomplete (manual)*, and the unit will resume the waiting process if you rerun it. In this context, if the unit whose end is being waited for terminates normally, the waiting status changes to *Wait complete* and the unit with wait conditions starts executing.

#

If *yes* is specified for the `PREWAITNOSCHUNITS` environment setting parameter, the unit with wait conditions performs a wait even if the unit is in the *Not sched. to exe.* status. For details about the `PREWAITNOSCHUNITS` environment setting parameter, see 20.4.2(122) `PREWAITNOSCHUNITS` in the *JPI/Automatic Job Management System 3 Configuration Guide*.

To temporarily change waiting condition settings, in the Wait Conditions Statuses window, from the **Operations** menu, choose **Waiting** and then **Wait Enabled** or **Wait Disabled**.



For details on temporarily changing waiting condition settings, see *4.5.15 Temporarily changing the wait condition settings for a jobnet or job* in the manual *JP1/Automatic Job Management System 3 Overview*.

## **(7) Behavior of units with wait conditions when the definition for the unit whose end is being waited for is invalid**

When you use wait conditions to control the order of unit execution, the process might not work as intended if you perform any of the following operations:

- When setting a wait condition, you specify a unit of a type that cannot act as a unit whose end is being waited for.
- You delete a unit whose end is being waited for after setting the wait conditions.
- You change the name of a unit whose end is being waited for after setting the wait conditions.

Such operations result in what is called an invalid definition for the unit whose end is being waited for. The response to an invalid definition for the unit whose end is being waited for depends on whether the unit whose end is being waited for has been registered for execution, and whether the unit whose end is being waited for is suspended.

When the unit whose end is being waited for is unregistered or is suspended

The message KAVS4957-E or KAVS4971-E is output, and the unit with wait conditions does not start executing. If the unit whose end is being waited for is suspended, the waiting process does not conclude when that unit finishes executing. If the unit whose end is being waited for terminates abnormally or is deleted, the unit with wait conditions does not itself terminate abnormally or output a message. The behavior of the unit with wait conditions in this scenario depends on the status of the unit whose end is being waited for when its suspension is released.

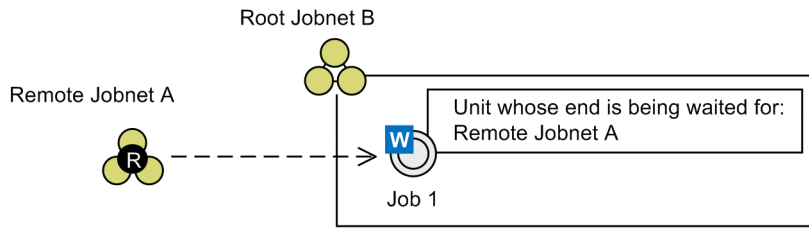
When the unit whose end is being waited for has been registered for execution and is not suspended

The message KAVS4954-E is output and the unit with wait conditions terminates abnormally.

The following figure shows an example of an invalid definition for the unit whose end is being waited for.

Figure 2–58: Example of invalid definition for the unit whose end is being waited for

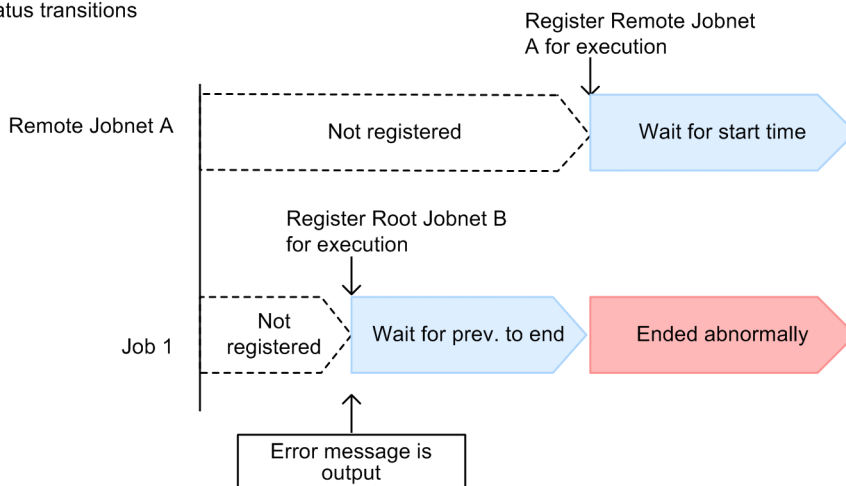
■ Unit configuration



Legend:

--> : Flow of wait processing

■ Status transitions



In this example, the specification of a remote jobnet as the unit whose end is being waited for has caused an invalid definition for the unit whose end is being waited for. If you register Root Jobnet B for execution before registering Remote Jobnet A, Job 1 (the unit with wait conditions) enters *Wait for prev. to end* status and waits for the unit whose end is being waited for to finish executing. At this point, the system outputs an error message indicating that the unit whose end is being waited for has not been registered for execution. If Remote Jobnet A is subsequently registered for execution, although a generation of Remote Jobnet A is scheduled, Job 1 terminates abnormally due to the invalid definition for the unit whose end is being waited for.

An invalid definition also results if you delete, move, or rename a root jobnet or planning group that is serving as a unit whose end is being waited for, after specifying it in a wait condition. In this scenario, the unit with wait conditions undergoes an abnormal termination as soon as it starts executing.

When OR is specified as the wait method, the unit with wait conditions will start executing if another unit whose end is being waited for finishes executing and satisfies the wait condition before the invalid definition is detected. However, if the invalid definition is detected before this can happen, the unit with wait conditions will terminate abnormally.

Preventing invalid definitions for the unit whose end is being waited for

To prevent invalid definitions from occurring, change the wait conditions accordingly each time you rename or delete a unit whose end is being waited for.

You can also use either of the following methods to preemptively check whether an invalid definition for the unit whose end is being waited for exists in a wait condition:

- Definition pre-check

You can use the definition pre-check function to check whether the wait condition is defined correctly. For details on conducting a definition pre-check, see [8. Definition Pre-Check](#).

A definition pre-check does not discover the following problems, which must be identified separately by monitoring the operation of the wait condition:

- An execution sequence loop caused by a wait condition
- An invalid definition caused by specifying a nested jobnet or job under a planning group as a unit whose end is being waited for
- An invalid definition caused by using a start condition
- Check the Wait Conditions Settings List window
  - You can check for invalid definitions for the unit whose end is being waited for in the Wait Conditions Settings List window of JP1/AJS3 - View. For details, see [\(8\) Checking units whose ends are being waited for](#).

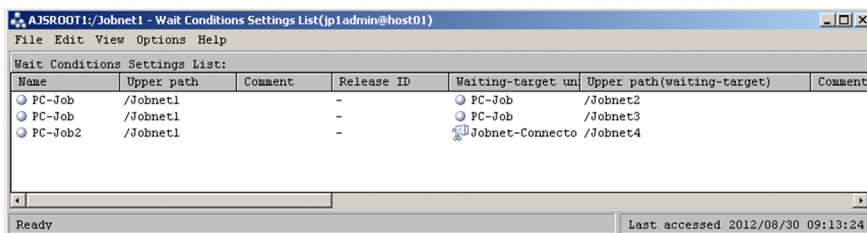
## (8) Checking units whose ends are being waited for

If you use JP1/AJS3 - View or the `ajsshows` command to define or monitor units with wait conditions, you can use the following methods to view a list of units with wait conditions and units whose ends are being waited for.

### Checking wait condition settings

You can use the Wait Conditions Settings List window of JP1/AJS3 - View to view detailed wait condition settings. The figure below shows the Wait Conditions Settings List window.

Figure 2–59: Wait Conditions Settings List window

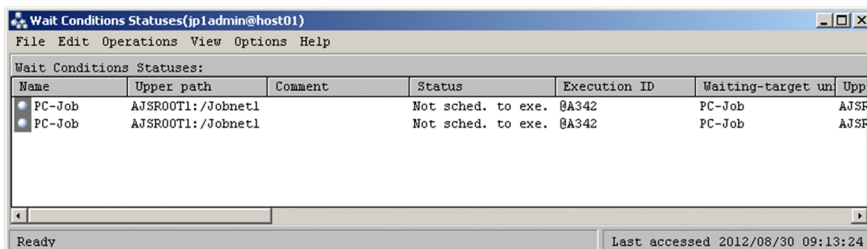


The Wait Conditions Settings List window lists all the units with wait conditions under a given unit, and also lists the units whose ends are being waited for that are specified for those units. You can use this information to check for invalid definitions among the units whose ends are being waited for, and to find out which wait method applies to the unit and its behavior when there are no applicable generations of the target unit.

### Checking the status of wait conditions

You can use the Wait Conditions Statuses window of JP1/AJS3 - View or the `ajsshows` command with the `-xw` option to check the status of the units whose ends are being waited for regarding active units with wait conditions. The following figure shows the Wait Conditions Statuses window.

Figure 2–60: Wait Conditions Statuses window



The Wait Conditions Statuses window displays a list of specific units with wait conditions, and displays the units whose ends are being waited for that are associated with those units. This allows you to check whether a unit whose end is being waited for is executing normally, without displaying the Jobnet Monitor window. You can also use it to check the wait status of a unit with wait conditions.

When you use a start condition for a unit whose end is being waited for, the information about the monitoring generation is displayed as information about the unit whose end is being waited for. Based on the status of the monitoring generation and the wait status, you can determine the execution status of the unit whose end is being waited for. For details, see *8.3.3(1) Checking the wait status when a unit whose end is being waited for uses a start condition* in the *JPI/Automatic Job Management System 3 Administration Guide*.

The Wait Conditions Statuses window also lets you enable or disable waiting for individual units whose ends are being waited for, and open the Jobnet Monitor window to display the status of units whose ends are being waited for.

For details on the `ajsshow` command, see *ajsshow* in *3. Commands Used for Normal Operations* in the manual *JPI/Automatic Job Management System 3 Command Reference*.

#### Cautionary notes

- The Wait Conditions Statuses window might take a long time to display information when a large number of units whose ends are being waited for are assigned to a unit with wait conditions. Refer to the cautionary note in *(5)(a) Wait methods for conditions with multiple units whose ends are being waited for*, and try to use as few units whose ends are being waited for as possible.
- To open the Jobnet Monitor window for a unit whose end is being waited for from the Wait Conditions Statuses window, you need to have `JP1_AJS_Guest` permission for the unit whose end is being waited for and its upper-level jobnet.
- The information displayed in the Wait Conditions Statuses window and the output of the `ajsshow` command is always based on the waiting rules. If you intend to restart the waiting process by enabling waiting for a unit and rerunning it, you can use this information to predict how the unit will behave when the waiting process starts (for example, whether it will wait for the unit whose end is being waited for to finish executing, or start executing immediately).

If you perform an operation that adds or deletes a generation of a unit with wait conditions or a unit whose end is being waited for, the waiting rules will select a different generation as the unit whose end is being waited for. This means that the generation displayed in the Wait Conditions Statuses window or the output of the `ajsshow` command might differ from the one that causes the wait status to transition to *Wait complete*. To ensure that the window always displays information for the true unit whose end is being waited for when the wait status changes to *Wait complete*, configure the system to save one day's generations of the unit with wait conditions and the unit whose end is being waited for, to ensure that the waiting rules always select the right generation of the unit whose end is being waited for. For details on waiting rules, see *(2) Rules governing waiting regarding units with wait conditions and units whose ends are being waited for*.

For details on the Wait Conditions Settings List window and the Wait Conditions Statuses window, see *12. Windows and Dialog Boxes* in the *JPI/Automatic Job Management System 3 Operator's Guide*.

## (9) Choosing between wait conditions or jobnet connectors for execution order control

When you want to control the execution order of units in different root jobnets, you can use wait conditions or jobnet connectors to establish links between units.

Select whether to use wait conditions or jobnet connectors according to your operational requirements.

The following table lists the advantages and disadvantages associated with wait conditions and jobnet connectors in specific contexts.

Table 2–15: Advantages and disadvantages of wait conditions and jobnet connectors

| Context                   | Wait condition   | Jobnet connector  |
|---------------------------|--|---|
| Unit configuration        | <p><b>Advantages</b></p> <p>Can be defined by simply assigning a wait condition to the succeeding unit (unit with wait conditions), requiring no changes to jobnet configuration.</p> <p><b>Disadvantages</b></p> <p>None.</p>   | <p><b>Advantages</b></p> <p>None.</p> <p><b>Disadvantages</b></p> <ul style="list-style-type: none"> <li>• Requires the definition of a new jobnet connector.</li> <li>• You cannot define a nested jobnet as a connection-destination jobnet without redefining the nested jobnet as a root jobnet.</li> </ul>   |
| Detailed unit definitions | <p><b>Advantages</b></p> <p>There is no need to edit the detailed definition of the preceding-side unit (unit whose end is being waited for).</p> <p><b>Disadvantages</b></p> <p>None.</p>   | <p><b>Advantages</b></p> <p>None.</p> <p><b>Disadvantages</b></p> <p>The connection range and jobnet connector name must be specified in the detailed definition of the connection-destination jobnet.</p>  |
| Methods of linking units  | <p><b>Advantages</b></p> <ul style="list-style-type: none"> <li>• Root jobnets, nested jobnets, and various job types can be linked together.</li> <li>• You can assign more than one unit whose end is being waited for to one unit with wait conditions.</li> </ul> <p><b>Disadvantages</b></p> <p>You cannot use wait conditions to link units under the control of different scheduler services. Units under different scheduler services can only be linked by using jobnet connectors or other means to redefine the jobnets, which demands the use of other methods in parallel with wait conditions.</p> | <p><b>Advantages</b></p> <p>Links can be created between root jobnets under the control of different scheduler services, allowing jobnet connectors to be used exclusively.</p> <p><b>Disadvantages</b></p> <ul style="list-style-type: none"> <li>• Only root jobnets can be linked.</li> <li>• You can only link one root jobnet to each jobnet connector.</li> </ul>   |
| Unit monitoring           | <p><b>Advantages</b></p> <p>You can use the Wait Conditions Statuses window to view a list of units whose ends are being waited for and units with wait conditions.</p> <p><b>Disadvantages</b></p> <ul style="list-style-type: none"> <li>• Because no relation lines are drawn between units whose ends are being waited for and units with wait conditions, it can be hard to ascertain the execution order.</li> <li>• The status of the unit whose end is being waited for is not reflected in the status of the unit with wait conditions.</li> </ul>  | <p><b>Advantages</b></p> <ul style="list-style-type: none"> <li>• The execution order is easy to ascertain because relation lines are drawn between jobnet connectors and other units.</li> <li>• The status of the connection-destination jobnet applies to the jobnet connector. This means you can centrally monitor the status of the connection-destination jobnet by monitoring the root jobnet associated with the jobnet connector.</li> </ul> <p><b>Disadvantages</b></p> <p>None.</p> |

## 2.2.6 Considerations regarding the use of macro variables

You can use macro variables to automate tasks because succeeding units can use the values of parameters each time jobs are executed.

### (1) Definition items for which macro variables can be specified

To pass information using macro variables, specify macro variables in the Define Details dialog box or Detailed Definition dialog box of succeeding units.

The following table describes the definition items for which macro variables can be specified.

Table 2–16: Definition items for which macro variables can be specified

| Unit type                       | Definition item   |
|---------------------------------|---|
| Jobnet                          | <b>Exec-agent</b> <sup>#1</sup>   |
| Unix job                        | <b>Exec-agent</b> <sup>#1</sup>   |
|                                 | <b>Command statement</b>  |
|                                 | <b>Script file name</b>   |
|                                 | <b>Parameters</b>   |
|                                 | <b>Environment variables</b>  |
|                                 | <b>Environment file</b> <sup>#1</sup>   |
|                                 | <b>Working path</b> <sup>#1</sup>   |
|                                 | <b>Standard input</b> <sup>#1</sup>   |
|                                 | <b>Standard output</b> <sup>#1</sup>  |
|                                 | <b>Standard error</b> <sup>#1</sup>   |
|                                 | File name for <b>End judgment</b> (when <b>Normal if specified file exists</b> or <b>Normal if file was updated</b> is selected in the <b>Rule</b> box) <sup>#1</sup> |
|                                 | <b>User name</b> <sup>#1</sup>  |
|                                 | <b>File to transfer</b> <sup>#1</sup>   |
|                                 | <b>Destination file</b> <sup>#1</sup>   |
| PC job                          | <b>Exec-agent</b> <sup>#1</sup>   |
|                                 | <b>File name</b>  |
|                                 | <b>Parameters</b>   |
|                                 | <b>Environment variables</b>  |
|                                 | <b>Environment file</b> <sup>#1</sup>   |
|                                 | <b>Working path</b> <sup>#1</sup>   |
|                                 | <b>Standard input</b> <sup>#1</sup>   |
|                                 | <b>Standard output</b> <sup>#1</sup>  |
|                                 | <b>Standard error</b> <sup>#1</sup>   |
|                                 | File name for <b>End judgment</b> (when <b>Normal if specified file exists</b> or <b>Normal if file was updated</b> is selected in the <b>Rule</b> box) <sup>#1</sup> |
|                                 | <b>User name</b> <sup>#1</sup>  |
|                                 | <b>File to transfer</b> <sup>#1</sup>   |
|                                 | <b>Destination file</b> <sup>#1</sup>   |
|                                 | QUEUE job   |
| <b>Queue name</b> <sup>#1</sup> |   |
| <b>Job name</b> <sup>#1</sup>   |   |

| Unit type                                   | Definition item   |
|---|---|
| QUEUE job                                   | <b>File name</b>  |
|   | <b>Parameters</b>   |
|   | <b>File to transfer<sup>#1</sup></b>  |
|   | <b>Destination file<sup>#1</sup></b>  |
| Judgment job                                | <b>Variable</b>   |
| Flexible job                                | <b>Destination</b>  |
|   | <b>Relay agent</b>  |
|   | <b>File name</b>  |
|   | <b>Parameters</b>   |
|   | <b>Environment variables</b>  |
|   | File name for <b>End judgment</b> (when <b>Normal if specified file exists</b> or <b>Normal if file was updated</b> is selected in the <b>Rule</b> box) |
| HTTP connection job                         | <b>Exec-agent</b>   |
|   | <b>Conn. configuration file name</b>  |
|   | <b>Trans. info. file name</b>   |
|   | <b>Trans. info. file name</b> (URL parameter)   |
|   | <b>Trans. info. file name</b> (Message body)  |
|   | <b>Status file name</b>   |
|   | <b>Received header file name</b>  |
|   | <b>Received body file name</b>  |
|   | <b>Standard output</b>  |
|   | <b>Standard error</b>   |
|   | File name for <b>End judgment</b> (when <b>Normal if specified file exists</b> or <b>Normal if file was updated</b> is selected in the <b>Rule</b> box) |
|   | <b>User name</b>  |
| Receive JP1 Event job                       | <b>Exec-agent<sup>#1</sup></b>  |
|   | <b>Host name<sup>#1</sup></b>   |
| Receive JP1 Event job - extended attributes | <b>Arbitrary extended attribute<sup>#1</sup></b>  |
| Monitoring Files job                        | <b>Exec-agent<sup>#1</sup></b>  |
|   | <b>Monitoring file<sup>#1</sup></b>   |
| Receive Mail job                            | <b>Exec-agent<sup>#1</sup></b>  |
| Monitoring Log Files job                    | <b>Exec-agent<sup>#1</sup></b>  |
|   | <b>File to be monitored<sup>#1</sup></b>  |
| Monitoring Event Log job                    | <b>Exec-agent<sup>#1</sup></b>  |
| Interval Control job                        | <b>Exec-agent<sup>#1</sup></b>  |

| Unit type                                 | Definition item   |
|---|---|
| Send JP1 Event job                        | <b>Exec-agent<sup>#1</sup></b>  |
|   | <b>Event destination host</b>   |
|   | <b>Message</b>  |
|   | <b>Extended attribute</b>   |
| Send Mail job                             | <b>Exec-agent<sup>#1</sup></b>  |
|   | <b>Recipients</b>   |
|   | <b>Subject</b>  |
|   | <b>Message text</b>   |
|   | <b>Message text - file name</b>   |
|   | <b>Profile name</b>   |
| Send Mail job - attached files            | <b>Attached file</b>  |
|   | <b>Attached file - list file name</b>   |
| Send MQ Message job - detailed definition | <b>Message data file</b>  |
| Send MSMQ Message job                     | <b>Path name</b>  |
|   | <b>Queue label</b>  |
|   | <b>Message label</b>  |
| Send MSMQ Message job - message text      | <b>Message body file</b>  |
| OpenView Status Report job                | <b>Exec-agent<sup>#1</sup></b>  |
|   | <b>Additional information</b>   |
| Local Power Control job                   | <b>Exec-agent<sup>#1</sup></b>  |
| Remote Power Control job                  | <b>Exec-agent<sup>#1</sup></b>  |
|   | <b>Remote host</b>  |
| Custom Unix job                           | <b>Exec-agent<sup>#1</sup></b>  |
|   | <b>Command statement</b>  |
|   | <b>Script file name</b>   |
|   | <b>Parameters</b>   |
|   | <b>Environment variables</b>  |
|   | <b>Environment file<sup>#1</sup></b>  |
|   | <b>Working path<sup>#1</sup></b>  |
|   | <b>Standard input<sup>#1</sup></b>  |
|   | <b>Standard output<sup>#1</sup></b>   |
|   | <b>Standard error<sup>#1</sup></b>  |
|   | File name for <b>End judgment</b> (when <b>Normal if specified file exists</b> or <b>Normal if file was updated</b> is selected in the <b>Rule</b> box) <sup>#1</sup> |
|   | <b>User name<sup>#1</sup></b>   |



| Unit type                             | Definition item   |
|---------------------------------------|---|
| Custom Unix job                       | <b>File to transfer</b> <sup>#1</sup>   |
|                                       | <b>Destination file</b> <sup>#1</sup>   |
| Custom PC job                         | <b>Exec-agent</b> <sup>#1</sup>   |
|                                       | <b>File name</b>  |
|                                       | <b>Parameters</b>   |
|                                       | <b>Environment variables</b>  |
|                                       | <b>Environment file</b> <sup>#1</sup>   |
|                                       | <b>Working path</b> <sup>#1</sup>   |
|                                       | <b>Standard input</b> <sup>#1</sup>   |
|                                       | <b>Standard output</b> <sup>#1</sup>  |
|                                       | <b>Standard error</b> <sup>#1</sup>   |
|                                       | File name for <b>End judgment</b> (when <b>Normal if specified file exists</b> or <b>Normal if file was updated</b> is selected in the <b>Rule</b> box) <sup>#1</sup> |
|                                       | <b>User name</b> <sup>#1</sup>  |
|                                       | <b>File to transfer</b> <sup>#1</sup>   |
| <b>Destination file</b> <sup>#1</sup> |   |
| Standard custom job                   | <b>Exec-agent</b> <sup>#1</sup>   |
|                                       | <b>Standard output</b> <sup>#1</sup>  |
|                                       | <b>Standard error</b> <sup>#1</sup>   |
|                                       | File name for <b>End judgment</b> (when <b>Normal if specified file exists</b> or <b>Normal if file was updated</b> is selected in the <b>Rule</b> box) <sup>#1</sup> |
|                                       | <b>User name</b> <sup>#1</sup>  |
| Custom event job <sup>#2</sup>        | <b>Exec-agent</b>   |
|                                       | There are items that must be defined in the linked program. <sup>#3</sup>   |

#1

You can specify macro variables when the version of JP1/AJS3 - View and JP1/AJS3 - Manager is 09-50 or later.

#2

You can specify macro variables when the version of JP1/AJS3 - View and JP1/AJS3 - Manager is 13-10 or later.

#3

For details about the items that can be defined, see the applicable documentation for the linked program.

## (2) Checking macro variables

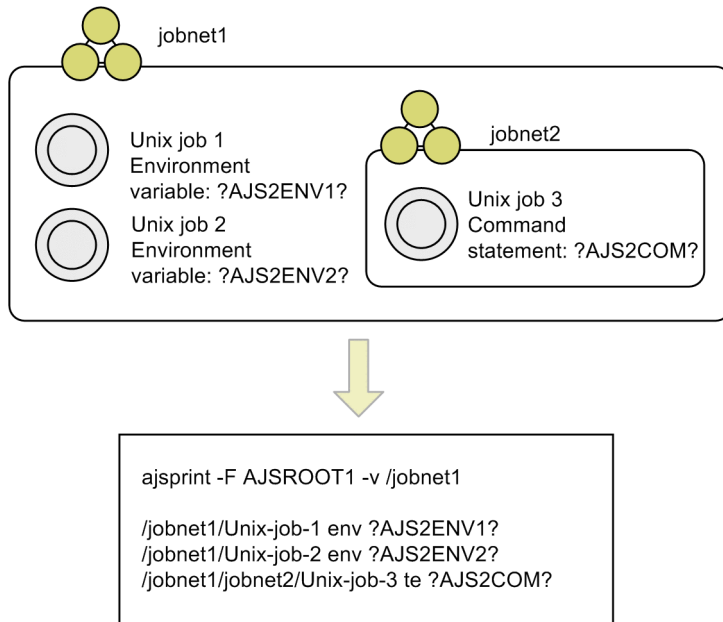
The names of the macro variables used in a job can be checked, and the result of inheriting information via macro variables can be checked after the job has been executed.

## (a) Listing the macro variables being used

You can list the macro variable names used in subordinate units in the Variables Used dialog box of JP1/AJS3 - View or by executing the `ajsprint` command with the `-v` option specified. For details, see *12.3.25 Variables Used dialog box* in the *JP1/Automatic Job Management System 3 Operator's Guide*, and see *ajsprint* in *3. Commands Used for Normal Operations* in the manual *JP1/Automatic Job Management System 3 Command Reference*.

The following figure shows an example of listing macro variables by using the `ajsprint` command.

Figure 2–61: Example of listing macro variables used in subordinate units



## (b) Viewing the result of inheriting information via macro variables

In the Inherit Result dialog box of JP1/AJS3 - View, you can view the macro variable names and passing information specified when a job is registered for execution. You can also view the names of the macro variables that the job inherits during execution and the result of inheriting those values. For details, see *12.7.10 Inherit Result dialog box* in the *JP1/Automatic Job Management System 3 Operator's Guide*.

You can also use the `ajsshow` command to view the same information by specifying the `%MV` two-byte format indicator for the `-i` option. If there are two or more items to be displayed as the result, the items are separated with commas (,). If there are two or more generations to be displayed, the generations are output on separate lines.

The following shows an example of executing the `ajsshow` command.

```
ajsshow -F AJSROOT2 -g 2 -i "%MV" /net

"AJS2ENV:/jp1_data/Job_Report", "AJS2COM:c:\temp\test.exe"
"AJS2ENV:/jp1_data/Job_Report", "AJS2COM:c:\temp\test.exe"

ajsshow -F AJSROOT2 -i "%MV" /net2

"AJS2ENV:/jp1_data2/Job_Report", "AJS2COM:c:\temp\test2.exe"
```

For details about the `ajsshow` command, see the *ajsshow* in *3. Commands Used for Normal Operations* in the manual *JP1/Automatic Job Management System 3 Command Reference*.

Note that whether the results of passing (inheritance) can be displayed differs according to the unit type and the unit status. The following table lists unit types, unit statuses, and whether the results of passing can be displayed.

Table 2–17: Whether the results of passing can be displayed

| Unit type                     | Unit status                     | Whether the results of passing can be displayed |
|-------------------------------|---------------------------------|---|
| Root jobnet                   | <i>Not registered</i>           | No  |
|                               | Other statuses                  | Yes <sup>#1</sup>                               |
| Nested jobnet                 | --                              | No  |
| Job <sup>#2, #3</sup>         | <i>Not registered</i>           | No  |
|                               | <i>Not sched. to exe.</i>       | No  |
|                               | <i>Waiting for prev. to end</i> | No  |
|                               | <i>Being held</i>               | No  |
|                               | <i>Now queuing</i>              | No  |
|                               | <i>Waiting to execute</i>       | No  |
|                               | <i>Now running</i>              | No  |
|                               | <i>Ended normally</i>           | Yes   |
|                               | <i>Normal end + False</i>       | Yes   |
|                               | <i>Ended with warning</i>       | Yes   |
|                               | <i>Ended abnormally</i>         | Yes   |
|                               | <i>Failed to start</i>          | Yes   |
|                               | <i>Unknown end status</i>       | Yes   |
|                               | <i>Killed</i>                   | Yes   |
|                               | <i>Not executed</i>             | No  |
|                               | <i>Bypassed</i>                 | No  |
| <i>Interrupted monitoring</i> | Yes                             |   |
| <i>Shutdown</i>               | No                              |   |

Legend:

Yes: The results can be displayed.

No: The results cannot be displayed.

-- : Not applicable

#1

Remote jobnets are excluded.

#2

The following jobs are excluded:

- The jobs subordinate to remote jobnets
- The jobs deleted during suspension
- The jobs added during suspension (other than those added after suspension is canceled)

#3

If the job status is changed, whether results can be displayed depends on the new status.

When a job that uses a macro variable is re-executed, the information passed via the macro variable during the previous execution is passed again. Therefore, by checking the result of the previous passing operation before re-executing the job, you will know the information that will be passed.

#### Supplementary notes

The results of passing differ from the previous execution in the following cases:

- When the preceding job from which information is passed is being-re-executed:  
The information passed from the preceding job is not displayed.
- When the preceding job from which information is passed was re-executed and the re-execution has finished:  
The passing information after the preceding job is re-executed is displayed.
- When the preceding job from which information is passed was deleted during suspension:  
The information passed from the preceding job is not displayed.
- When two or more passing information setting jobs that specify the same macro variable name are defined in a jobnet  
The passing information set by the last executed passing information setting job is displayed, regardless of the relation lines.
- When re-execution of a passing information setting job terminates after the regular expression specified in the job has been changed  
The passing information extracted based on the new regular expression when the passing information setting job is re-executed is displayed.
- When a passing information setting job is re-executed after the preceding job has been sent to a standard output file different from the one used during the previous execution  
The passing information extracted from the new standard output file when the passing information setting job is re-executed is displayed.

### (3) Determining the JP1/AJS3 behavior if the passing of information via a macro variable fails

If the passing of information via a macro variable fails during execution of a job, JP1/AJS3 either changes the job status to *Failed to start* or continues jobnet execution by using the macro variable name as a string value.

You can use the `MACROCHANGEFAIL` environment setting parameter to determine the JP1/AJS3 behavior when the passing of information via a macro variable has failed.

For details about how to set the `MACROCHANGEFAIL` environment setting parameter, see *6.2.10 Setting the action to be taken if the value of a macro variable cannot be passed in the JP1/Automatic Job Management System 3 Configuration Guide (Windows)* or *15.2.10 Setting the action to be taken if the value of a macro variable cannot be passed in the JP1/Automatic Job Management System 3 Configuration Guide (UNIX)*.

### (4) Using macro variables

The table below describes how to use macro variables. For details, see the subsections listed in the *Reference* column.

Table 2–18: Using macro variables

| No. | Purpose  | Reference   |
|-----|--|---|
| 1   | Passing the event information received by an event job and a custom event job to a succeeding unit | <ul style="list-style-type: none"> <li>• <i>2.4.4(6) Passing information received by an event job and a custom event job</i></li> <li>• <i>7.6.7 Notes on defining passing information</i></li> </ul> |

| No. | Purpose  | Reference  |
|-----|--|--|
| 1   | Passing the event information received by an event job and a custom event job to a succeeding unit   | <ul style="list-style-type: none"> <li>• <a href="#">7.10.1 Notes on defining passing information</a></li> </ul>   |
| 2   | Outputting information that changes according to the circumstances of job execution to the standard output file and passing the information to a succeeding unit | <a href="#">2.4.9 Passing information that changes dynamically to a succeeding unit (example of defining a jobnet that uses a passing information setting job)</a> |
| 3   | Passing to lower-level units the values specified for a jobnet when it was registered for execution  | <a href="#">4.1.2 Specifying macro variables during registration for execution in the manual JP1/Automatic Job Management System 3 Overview</a>                    |

## (5) Cautionary notes

- Make sure that the total size of macro variable names and passing information in one root jobnet does not exceed 4,096 bytes. If the total size exceeds 4,096 bytes, data after the 4,096th byte might not be passed or the passing information setting job might end abnormally. To estimate the total size of macro variable names and passing information in a root jobnet, use the following procedure:
  1. List the names of all macro variables used by all units in the root jobnet.
  2. For all macro variables listed in step 1, consider the maximum size of passing information, and then obtain the result of the following expression for each macro variable:  
 $macro-variable-name-size + maximum-passing-information-size + 5$  (bytes)
  3. Sum all the values obtained in step 2.
- If the same macro variable name is specified in multiple subordinate units of a root jobnet, the value of the macro variable is determined according to the following priority:
  1. The passing information of an event job and a custom event job (other than the one in the start conditions) in the jobnet
  2. The passing information of an event job and a custom event job in the start conditions
  3. The passing information specified during registration for execution
  4. The passing information generated by a passing information setting job<sup>#</sup>

#:

If the same macro variable name is specified in two or more passing information setting jobs in a root jobnet, the passing information generated by the passing information setting job executed later is valid.

We recommend that you do not specify the same macro variable name at multiple locations in one root jobnet.

- For some definition items, the support of macro variables started with JP1/AJS3 version 09-50. If you execute a unit with a macro variable specified for a definition item such as these when the JP1/AJS3 - Manager version is earlier than 09-50, the macro variable is not replaced by the passing information. Instead, the macro variable name itself is passed as the passing information.

For details about the target definition items, see [\(1\) Definition items for which macro variables can be specified](#) or the descriptions of dialog boxes for definitions in [12. Windows and Dialog Boxes](#) in the *JP1/Automatic Job Management System 3 Operator's Guide*.

- Each generation has its own values (passing information) for macro variables, and these values (passing information) cannot be passed among different generations. As a result, passing information cannot be passed to a unit that will be executed by a different generation, and cannot be inherited from a unit executed by a different generation.

The units that can be executed by different generations are as follows:

- Units (root jobnet, or planning group) that are the connection-destination jobnets of jobnet connectors
- Units whose end is being waited for, and that are subordinate to other root jobnets

- When you use a remote jobnet, passing information cannot be passed to a unit that is subordinate to the remote jobnet. In addition, passing information cannot be inherited from a unit that is subordinate to the remote jobnet.

## 2.3 Tips on work task automation

---

This section gives some tips on work task automation that fall outside the categories discussed previously in this chapter.

### 2.3.1 Processing with a distributed load

JP1/AJS3 allows you to specify a group of execution agents as the agent host on which to execute a job or jobnet. This allows the processing load to be distributed among the execution agents in the group. The patterns of load distribution are as follows:

- Jobs are distributed evenly among a number of execution agents.
- Individual execution agents are assigned different limits for the number of concurrently executable jobs.
- When the number of jobs executing concurrently at a specific execution agent reaches the limit, execution is distributed to the other execution agents.

You can specify an execution agent group to execute the following units:

- Root jobnets
- Nested jobnets
- PC jobs
- Unix jobs
- Flexible jobs<sup>#</sup>
- HTTP connection jobs
- Action jobs
- Custom jobs

#

For a flexible job, you can specify an execution agent group as a relay agent.

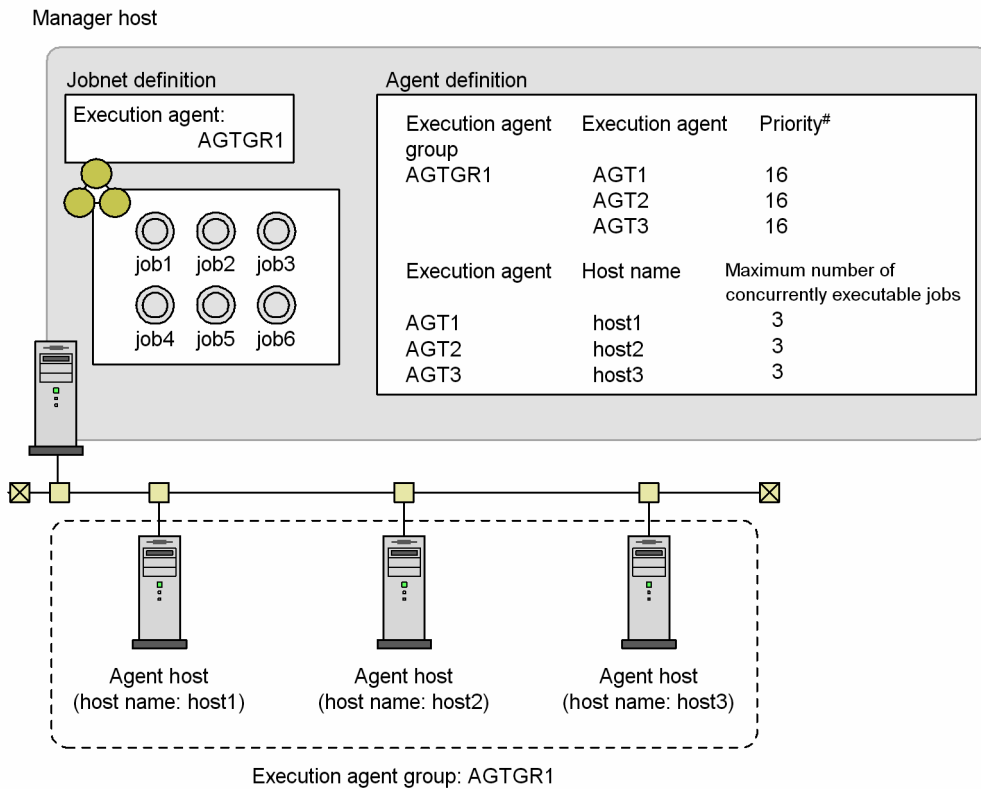
This section describes how load distribution processing takes place in each pattern. In this example, the agent group AGTGR1 is specified as the execution agent group for a jobnet containing six jobs (job1 to job6).

#### (1) Equal load distribution

In this pattern, the agents in the execution agent group have the same priority and the same maximum number of concurrently executable jobs.

The following figure shows an example of the pattern in which the load is distributed evenly among execution agents.

Figure 2–62: Example of equal load distribution



#: Specify the execution agent priority in the range 1 to 16, where 16 is the highest priority. The default is 16 (highest priority).

In this example, the job execution order is determined as follows, where the priorities of the three execution agents (AGT1, AGT2, and AGT3) in execution agent group AGTGR1 are 16, and a maximum of three jobs can be executed concurrently by each agent.

1. Determine the execution agent for job1.

Ratio: *number of active jobs / maximum number of concurrently executable jobs*:

- Execution agent AGT1: 0 / 3
- Execution agent AGT2: 0 / 3
- Execution agent AGT3: 0 / 3

Because all execution agents have the same agent usage rate<sup>#</sup> and the same priority, one of AGT1 to AGT3, which are defined in the execution agent group AGTGR1, becomes the execution agent for job1. In this example, AGT1 becomes the execution agent for job1.

2. Determine the execution agent for job2.

Because job1 has been assigned to execution agent AGT1, the ratio (*active jobs / maximum concurrent jobs*) is now:

- Execution agent AGT1: 1 / 3
- Execution agent AGT2: 0 / 3
- Execution agent AGT3: 0 / 3

Execution agents AGT2 and AGT3 now have the same agent usage rate<sup>#</sup>. Because execution agents AGT2 and AGT3 also have the same priority, AGT2 or AGT3, which are defined in the execution agent group AGTGR1, becomes the execution agent for job2. In this example, AGT2 becomes the execution agent for job2.

3. Determine the execution agent for job3.



Because job2 has been assigned to execution agent AGT2, the ratio (*active jobs / maximum concurrent jobs*) is now:

- Execution agent AGT1: 1 / 3
- Execution agent AGT2: 1 / 3
- Execution agent AGT3: 0 / 3

Execution agent AGT3 now has the lowest agent usage rate<sup>#</sup>. Therefore, job3 is assigned to execution agent AGT3.

#### 4. Determine the execution agent for job4.

Now that job1 through job3 have been assigned to execution agents, the ratio (*active jobs / maximum concurrent jobs*) is now:

- Execution agent AGT1: 1 / 3
- Execution agent AGT2: 1 / 3
- Execution agent AGT3: 1 / 3

Because all execution agents now have the same agent usage rate<sup>#</sup> again, one of AGT1 to AGT3, which are defined in the execution agent group AGTGR1, becomes the execution agent for job4. In this example, AGT1 becomes the execution agent for job4.

#### 5. Determine the execution agent for job5.

Because job4 has been assigned to execution agent AGT1, the ratio (*active jobs / maximum concurrent jobs*) is now:

- Execution agent AGT1: 2 / 3
- Execution agent AGT2: 1 / 3
- Execution agent AGT3: 1 / 3

Because execution agents AGT2 and AGT3 now have the same agent usage rate<sup>#</sup>, AGT2 or AGT3, which are defined in the execution agent group AGTGR1, becomes the execution agent for job5. In this example, AGT2 becomes the execution agent for job5.

#### 6. Determine the execution agent for job6.

Because job5 has been assigned to execution agent AGT2, the ratio (*active jobs / maximum concurrent jobs*) is now:

- Execution agent AGT1: 2 / 3
- Execution agent AGT2: 2 / 3
- Execution agent AGT3: 1 / 3

Because execution agent AGT3 now has the lowest agent usage rate<sup>#</sup>, AGT3 becomes the execution agent for job6.

#

The agent usage rate is the ratio of the number of active jobs to the maximum number of concurrently executable jobs assigned to the agent host.

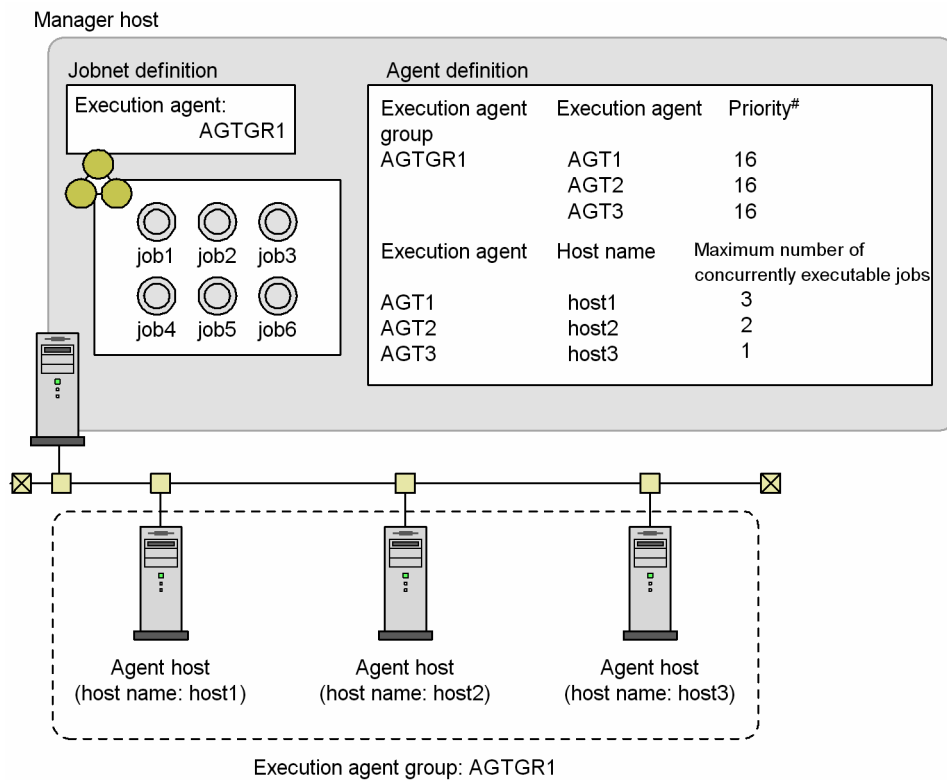
For details about how to distribute jobs to multiple execution agents in an execution agent group and the agent usage rate, see *5.4.9 Distributing jobs among execution hosts (agent hosts)* in the manual *JP1/Automatic Job Management System 3 Overview*.

## (2) Distribution among agents assigned different job execution limits

To distribute loads by using different job execution limits at each execution agent, set the maximum number of concurrently executable jobs to a different value, but set the same priority for each execution agent in the group.

The following figure shows an example where the execution agents are assigned a different number of concurrently executable jobs.

Figure 2–63: Example of distribution between execution agents with different job execution limits



#: Specify the execution agent priority in the range 1 to 16, where 16 is the highest priority. The default is 16 (highest priority).

In this example, the job execution order is determined as follows, where the maximum number of concurrently executable jobs of the three execution agents (AGT1, AGT2, and AGT3) in execution agent group AGTGR1 are 3, 2, and 1.

1. Determine the execution agent for job1.

Ratio: *number of active jobs / maximum number of concurrently executable jobs*:

- Execution agent AGT1: 0 / 3
- Execution agent AGT2: 0 / 2
- Execution agent AGT3: 0 / 1

All execution agents have the same priority. Therefore, the execution agent that will have the lowest agent usage rate<sup>#</sup> among those defined in the execution agent group AGTGR1 when it executes the next job becomes the execution agent for job1. In this example, AGT1 becomes the execution agent for job1.

2. Determine the execution agent for job2.

Because job1 has been assigned to execution agent AGT1, the ratio (*active jobs / maximum concurrent jobs*) is now:

- Execution agent AGT1: 1 / 3
- Execution agent AGT2: 0 / 2
- Execution agent AGT3: 0 / 1

All execution agents have the same priority. Therefore, the execution agent that will have the lowest agent usage rate<sup>#</sup> among those defined in the execution agent group AGTGR1 when it executes the next job becomes the execution agent for job2. In this example, AGT2 becomes the execution agent for job2.

3. Determine the execution agent for job3.

Because job2 has been assigned to execution agent AGT2, the ratio (*active jobs / maximum concurrent jobs*) is now:

- Execution agent AGT1: 1 / 3
- Execution agent AGT2: 1 / 2
- Execution agent AGT3: 0 / 1

Execution agent AGT1 now has the lowest agent usage rate<sup>#</sup>. Therefore, job3 is assigned to execution agent AGT1.

#### 4. Determine the execution agent for job4.

Because job3 has been assigned to execution agent AGT1, the ratio (active jobs / maximum concurrent jobs) is now as follows:

- Execution agent AGT1: 2 / 3
- Execution agent AGT2: 1 / 2
- Execution agent AGT3: 0 / 1

All execution agents have the same priority and every execution agent will also have the same agent usage rate<sup>#</sup> when it executes the next job. Therefore, one of AGT1 to AGT3, which are defined in the execution agent group AGTGR1, becomes the execution agent for job4. In this example, AGT1 becomes the execution agent for job4.

#### 5. Determine the execution agent for job5.

Because job4 has been assigned to execution agent AGT1, the ratio (*active jobs / maximum concurrent jobs*) is now:

- Execution agent AGT1: 3 / 3
- Execution agent AGT2: 1 / 2
- Execution agent AGT3: 0 / 1

Execution agents AGT2 and AGT3 will have the same agent usage rate<sup>#</sup> when they execute the next job. Because AGT2 and AGT3 also have the same priority, AGT2 or AGT3, defined in the execution agent group AGTGR1, becomes the execution agent for job5. In this example, AGT2 becomes the execution agent for job5.

#### 6. Determine the execution agent for job6.

Because job5 has been assigned to execution agent AGT2, the ratio (*active jobs / maximum concurrent jobs*) is now:

- Execution agent AGT1: 3 / 3
- Execution agent AGT2: 2 / 2
- Execution agent AGT3: 0 / 1

Because both execution agents AGT1 and AGT2 have reached the job execution limits assigned to them, AGT3 becomes the execution agent for job6.

#

The agent usage rate is the ratio of the number of active jobs to the maximum number of concurrently executable jobs assigned to the agent host.

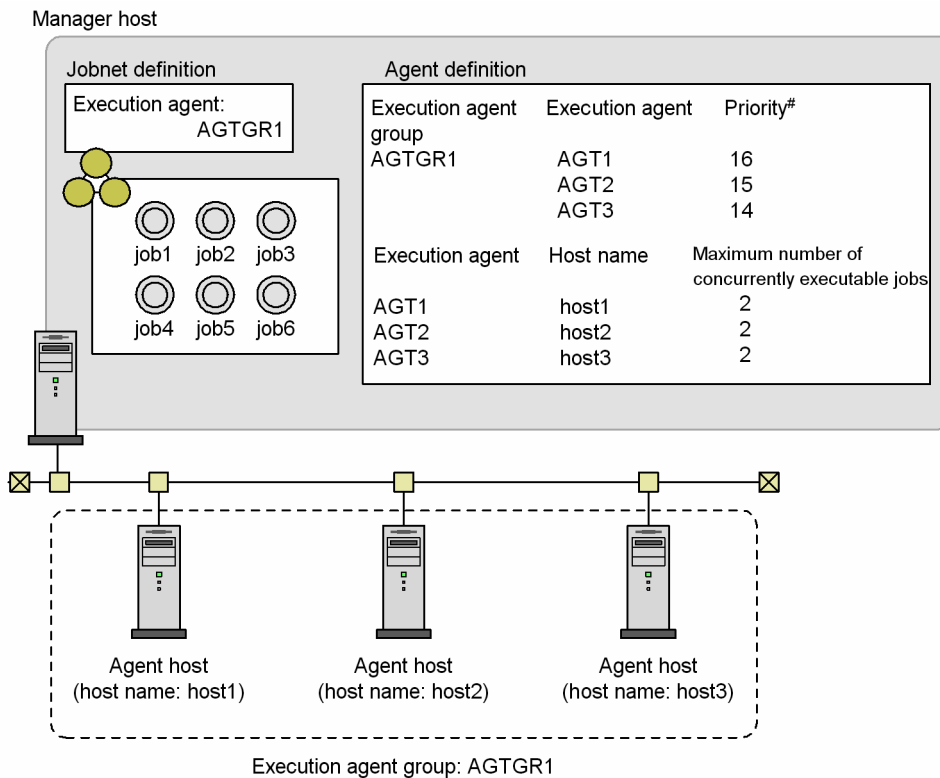
For details about how to distribute jobs to multiple execution agents in an execution agent group and the agent usage rate, see *5.4.9 Distributing jobs among execution hosts (agent hosts)* in the manual *JP1/Automatic Job Management System 3 Overview*.

### **(3) Job execution distributed to other execution agents when the limit is reached**

To have JP1/AJS3 distribute jobs to another execution agent in the group when the number of concurrently executed jobs reaches a limit at an agent, you must specify the priority of each agent in the group. You can also assign different job execution limits to each execution agent as required.

The following figure shows an example where jobs are distributed to another execution agent when the number of concurrently executed jobs has reached the limit.

Figure 2–64: Example of distribution to another execution agent when the number of jobs executed reaches the limit



#: Specify the execution agent priority in the range 1 to 16, where 16 is the highest priority. The default is 16 (highest priority).

In this example, the job execution order is determined as follows, where the priorities of the three execution agents (AGT1, AGT2, and AGT3) in execution agent group AGTGR1 are 16 (highest), 15 (second highest), and 14 (third highest), and a maximum of two jobs can be executed concurrently by each agent.

1. job 1 is executed.

Ratio: *number of active jobs / maximum number of concurrently executable jobs*:

- Execution agent AGT1: 0 / 2
- Execution agent AGT2: 0 / 2
- Execution agent AGT3: 0 / 2

Because AGT1 is the execution agent with the highest priority, job1 is assigned to AGT1.

2. job 2 is executed.

Now that job1 has been assigned to AGT1, the ratio (*active jobs / maximum concurrent jobs*) is now:

- Execution agent AGT1: 1 / 2
- Execution agent AGT2: 0 / 2
- Execution agent AGT3: 0 / 2

Although execution agents AGT2 and AGT3 now have a lower agent usage rate, also job2 is assigned to AGT1 because AGT1 has the highest priority.

3. job 3 is executed.

Now that job2 has been assigned to AGT1, the ratio (*active jobs / maximum concurrent jobs*) is now:

- Execution agent AGT1: 2 / 2
- Execution agent AGT2: 0 / 2
- Execution agent AGT3: 0 / 2

Although AGT1 has the highest priority, it has now reached its maximum number of concurrently executable jobs. Therefore, job3 is assigned to AGT2, the execution agent with the next highest priority.

4. job 4 is executed.

Now that job3 has been assigned to AGT2, the ratio (*active jobs / maximum concurrent jobs*) is now:

- Execution agent AGT1: 2 / 2
- Execution agent AGT2: 1 / 2
- Execution agent AGT3: 0 / 2

Like job3, job4 is assigned to AGT2.

5. job 5 is executed.

Now that job4 has been assigned to AGT2, the ratio (*active jobs / maximum concurrent jobs*) is now:

- Execution agent AGT1: 2 / 2
- Execution agent AGT2: 2 / 2
- Execution agent AGT3: 0 / 2

As AGT1 and AGT2 have both reached their maximum number of concurrently executable jobs, job5 is assigned to execution agent AGT3.

6. job 6 is executed.

Now that job5 has been assigned to AGT3, the ratio (*active jobs / maximum concurrent jobs*) is now:

- Execution agent AGT1: 2 / 2
- Execution agent AGT2: 2 / 2
- Execution agent AGT3: 1 / 2

As AGT1 and AGT2 have both reached their maximum number of concurrently executable jobs, job 6 is assigned to execution agent AGT3.

#

The agent usage rate is the ratio of the number of active jobs to the maximum number of concurrently executable jobs assigned to the agent host.

For details about how to distribute jobs to multiple execution agents in an execution agent group and the agent usage rate, see *5.4.9 Distributing jobs among execution hosts (agent hosts)* in the manual *JP1/Automatic Job Management System 3 Overview*.

## 2.3.2 JP1/AJS3 functions useful for work task automation

This section describes the JP1/AJS3 functionality that you can use to automate work tasks.

The following table lists tasks that you might want to perform with JP1/AJS3 and the functionality to use in each case.

Table 2–19: JP1/AJS3 functions for realizing work task automation

| No. | Task you want to perform in JP1/AJS3   | Relevant JP1/AJS3 function  | Relevant section  |
|-----|--|---|---|
| 1   | Prepare processing that changes dynamically depending on the results of the preceding job.   | Judgment job  | <i>2.4.3 Dynamically changing a process depending on the result of a preceding job (example of defining a jobnet that uses a judgment job)</i>            |
| 2   | Receive JP1 events from a host of your choice.   | JP1 event reception monitoring job  | <i>2.4.4 Executing an event-driven process (example of defining a jobnet that uses an event job and a custom event job)</i>                               |
| 3   | Detect when a file is created, deleted, or updated.  | File monitoring job   |   |
| 4   | Monitor specific data output to user log files or syslog.  | Log file monitoring job, or a combination of the JP1/Base log file trap function and a JP1 event reception monitoring job                                 |   |
| 5   | Monitor messages output to the Windows event log.  | Monitoring Event Log job, or a combination of the JP1/Base event log trap function and a Receive JP1 Event job  |   |
| 6   | Execute jobs with a regular time interval between them, or execute jobnets at a regular interval.                                  | Interval Control job  |   |
| 7   | Execute a jobnet at regular intervals.   | Interval Control job with a start condition   |   |
| 8   | Use the information of the preceding event job and custom event job to execute subsequent processing (job or jobnet).              | Inheriting of event job and custom event job reception information  |   |
| 9   | Execute processing (of a job or jobnet) by using a received file with an unspecified filename transferred into a specified folder. | <ul style="list-style-type: none"> <li>• Start condition</li> <li>• Monitoring Files job</li> <li>• Inheriting event job reception information</li> </ul> |   |
| 10  | Send JP1 events to a host of your choice.  | Send JP1 Event job  | <i>2.4.5 Sending a JP1 event at completion of the preceding job or when an event occurs (example of defining a jobnet that uses a Send JP1 event job)</i> |
| 11  | Execute a recovery job or jobnet when an error occurs during job execution.  | <ul style="list-style-type: none"> <li>• Recovery job</li> <li>• Recovery jobnet</li> </ul>   | <i>2.4.6 Executing a specific process when a job ends abnormally (example of defining a jobnet that uses a recovery unit)</i>                             |
| 12  | Control the execution order of different root jobnets.   | Jobnet connector  | <i>2.4.7 Controlling the execution order of a root jobnet (example of defining a jobnet that uses a jobnet connector)</i>                                 |
| 13  | Control the execution order of units in different jobnets.   | Wait condition  | <i>2.4.8 Controlling the order of unit execution in different jobnets (example of defining a jobnet that uses a wait condition)</i>                       |
| 14  | Extract necessary information from the standard output file of a preceding job to execute subsequent processing (a job or jobnet). | Passing information setting job   | <i>2.4.9 Passing information that changes dynamically to a succeeding unit (example of defining a jobnet that uses a passing information setting job)</i> |

| No. | Task you want to perform in JP1/AJS3   | Relevant JP1/AJS3 function   | Relevant section  |
|-----|--|--|---|
| 15  | Automatically retry a job and continue the task if an executable file defined for the job ends abnormally.                   | Automatic retry on abnormal end  | <i>2.4.13 Automatic retry for abnormally ending jobs</i>  |
| 16  | Put subsequent executions of a jobnet on hold when the jobnet ends abnormally  | Set a hold attribute for jobnets <ul style="list-style-type: none"> <li>• Hold if prev. = 'abend'</li> <li>• Hold if prev. = 'warning' or 'abend'</li> </ul> | <i>12.3.8 Define Details - [Jobnet] dialog box (for a root jobnet) in the JP1/Automatic Job Management System 3 Operator's Guide</i>                    |
| 17  | Change the definition of a jobnet during operation.  | Jobnet release   | <i>7.3 Switching a jobnet definition while the jobnet is registered for execution in the JP1/Automatic Job Management System 3 Administration Guide</i> |
| 18  | Group together several root jobnets with different schedules so that their tasks can be performed as a single jobnet.        | Planning group   | <i>11.1 Using a planning group to change the plans for root jobnets in the manual JP1/Automatic Job Management System 3 Overview</i>                    |
| 19  | Execute processing (job or jobnet) in response to mail sent from a particular person or mail with a particular subject line. | <ul style="list-style-type: none"> <li>• Linkage with a mail system</li> <li>• Receive Mail job</li> </ul>   | <i>2. Linking Mail Systems in the JP1/Automatic Job Management System 3 Linkage Guide</i>   |
| 20  | Receive email notification when a job or jobnet has completed, or when there is an error in the system.                      | <ul style="list-style-type: none"> <li>• Linkage with a mail system</li> <li>• Send Mail job</li> </ul>  |   |
| 21  | Monitor the operating status of JP1/AJS3 and job execution statuses by using HP NNM.   | <ul style="list-style-type: none"> <li>• Linkage with HP NNM</li> <li>• OpenView Status Report Job</li> </ul>  | <i>A. Monitoring Jobnets Using HP NNM in the JP1/Automatic Job Management System 3 Linkage Guide</i>  |
| 22  | Automate a work task whose parameters change with each execution, without having to change the jobnet definition.            | Macro variable and passing information   | <i>3.1.3 Applications that use macro variables in the manual JP1/Automatic Job Management System 3 Overview</i>   |

## 2.4 Jobnet definition examples

This section provides some examples of jobnet definition, based on the characteristics of each units, and keeping in mind the specific requirements of the application and its processes.

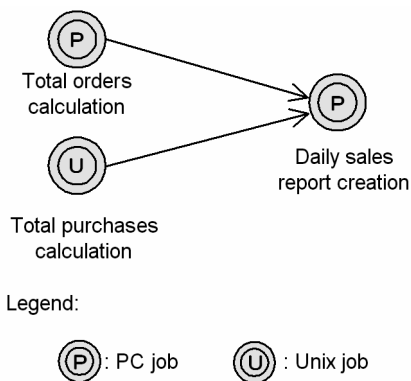
The descriptions in this section assume that the user is defining jobnets through a GUI. For details on the parameters you can specify in a unit definition, and how to use each type of unit, see 5. *Defining Jobnets* in the *JPI/Automatic Job Management System 3 Operator's Guide*.

### 2.4.1 Executing a process in a specified file (example of defining a jobnet consisting of standard jobs)

Use standard jobs when defining a jobnet in which processes are performed by file execution.

In the example below, standard jobs are used to define a jobnet in which the total orders are calculated and output to a file by the program `juchu.exe`, the total purchases are calculated and output to a file by the program `shiire.exe`, and a daily sales report is prepared and printed, using sales figures calculated from these results, by the program `nippou.exe`. Host A (Windows) is used for calculating the total orders and preparing the daily sales report. Host B (UNIX) is used for calculating the total purchases.

Figure 2–65: Example of defining a jobnet that uses standard jobs



The *total orders calculation* and *daily sales report creation* jobs are executed on a Windows host. Therefore, use PC jobs for these processes. Use a Unix job for the *total purchases calculation* since this job is executed on a UNIX host.

For the *total orders calculation* job, specify `hostA` in **Exec-agent**, and `juchu.exe` in **File name**.

For the *total purchases calculation* job, specify `hostB` in **Exec-agent**, and `shiire.exe` in **File name**.

For the *daily sales report creation* job, specify `hostA` in **Exec-agent**, and `nippou.exe` in **File name**.

The *total orders calculation* and *total purchases calculation* jobs do not need to run in any particular order, but the sales data must be calculated from the results of these two jobs, giving the job flow shown above.

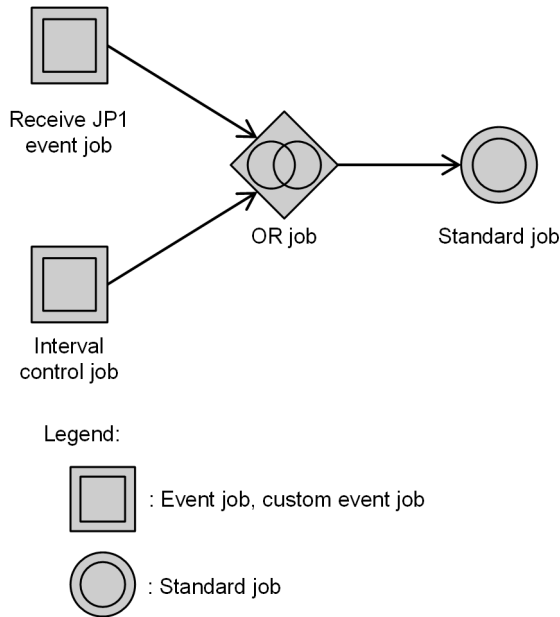
### 2.4.2 Executing a process when one of multiple conditions is satisfied (example of defining a jobnet that uses an OR job)

Use an OR job when defining a jobnet in which a process is performed when one of multiple conditions is satisfied.



In the example below, an OR job is used to define a jobnet that monitors the system for 10 minutes for receipt of a JP1 event. If the JP1 event is received, the succeeding job is executed immediately. If the JP1 event is still not received after 10 minutes, the succeeding job is executed at that point.

Figure 2–66: Example of defining a jobnet that uses an OR job



Only event jobs and custom event jobs can be defined as the preceding jobs of an OR job. In this example, we want to watch for a JP1 event to be received, so we define a Receive JP1 event job. We use an Interval control job to monitor for elapse of the 10-minute time period, and define the wait time as 10 minutes.

If any event monitored by the event job and the custom event job defined as the preceding job of an OR job occurs, the succeeding job is executed, and other event jobs and custom event jobs stop monitoring events and enter the *Bypassed* status. Thus, in this example, once the event being monitored by the Receive JP1 event job is received, the Interval control job stops monitoring for the interval to elapse. At completion of the succeeding job that was triggered by event reception, the jobnet also ends.

When the succeeding job performs an end judgment on the return value of the OR job, the OR job's return value is the same as the return value of the event job and the custom event job executed as the preceding job.

#### Cautionary notes

- If execution of an event job or a custom event job is stopped, the event job or the custom event job enters the *Not sched. to exe.* status. Thereafter, when the preceding job of the event job or the custom event job terminates, the status of the event job or the custom event job changes from *Not sched. to exe.* to *Bypassed*, and the succeeding OR job is executed.

If the status of an event job or a custom event job has changed to *Killed* or *Ended abnormally* because of killing a jobnet or a timeout, other event jobs and custom event jobs also stop monitoring events and enter the *Bypassed* status. In this case, however, the OR job is not executed because the preceding job of the OR job includes the abnormally terminated event job or custom event job.

- When you restart execution from the event job and the custom event job defined as the first or second preceding job of an OR job, check whether the preceding jobs include an event job or a custom event job that has been terminated and placed in the *Bypassed* status. If there is such an event job or a custom event job, the OR job is executed immediately after the restart. In this situation, if, for example, you want the event job or the custom event job to monitor events, place all event jobs and custom event jobs that directly precede the OR job in the *Ended abnormally* status. Then, re-execute the root jobnet with **From abnormally ended job** selected in **Rerun Method**.

- If there is an event job or a custom event job that cannot be executed in accordance with the relation line, such as when the preceding unit is running, the succeeding unit of an OR job will not be executed even if another event job or custom event job ends. When the preceding unit ends, the event job or the custom event job enters the Bypassed status and the succeeding unit of the OR job is executed.

## 2.4.3 Dynamically changing a process depending on the result of a preceding job (example of defining a jobnet that uses a judgment job)

Use a judgment job for a jobnet in which processing is to change dynamically depending on the results of the preceding job, on whether a file exists, or on the information inherited from the preceding job.

### (1) Example of using a judgment job

JP1/AJS3 provides judgment jobs in which the next process is selected according to the end result of the preceding process. Judgment takes place according to the following two patterns:

- Judgment based on return code
- Judgment based on whether a file exists

The following provides examples of using each type of judgment job.

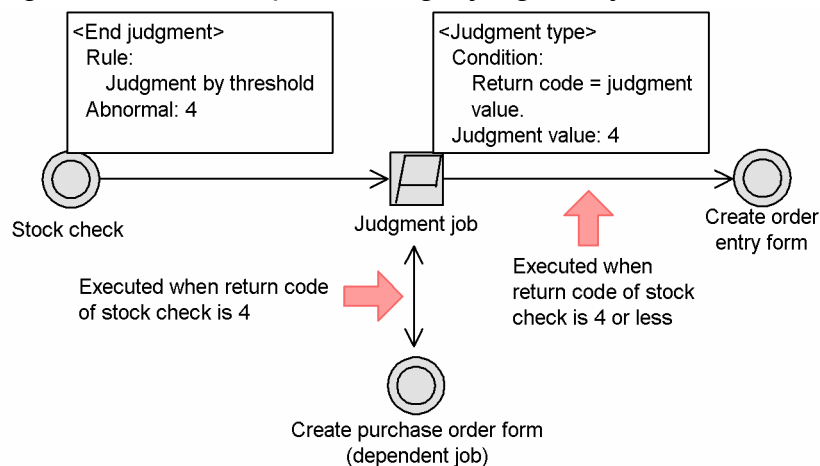
We also explain how to rerun a dependent unit if it ends abnormally.

#### (a) Judgment based on return code

The following figure shows an example of using a judgment job that works with return codes.

The jobnet created in this example creates a purchase order form if the stock level is low, or an order entry form if there is adequate stock, depending on the execution result of a job that checks the stock level.

Figure 2–67: Example of using a judgment job that works with return codes



The stock check job returns 3 or a lower value if there is adequate stock, 4 if the stock level is low, and 5 or higher if there is a stock shortage.

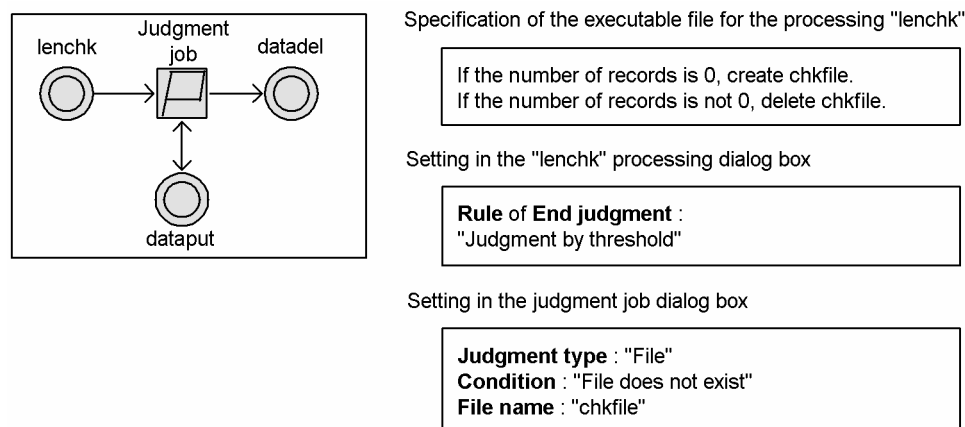
By creating a processing flow like that in the figure above, you can execute the following processing:

- When there is adequate stock  
The return code is 3 or lower. Therefore, the *create purchase order form* job is skipped and the *create order entry form* job is executed.
- When the stock level is low  
The return code is 4. Therefore, the *create purchase order form* job is executed first, followed by the *create order entry form* job.

## (b) Judgment based on file presence or absence

The following figure shows an example of using a judgment job based on the presence or absence of a file.

Figure 2–68: Example of using a judgment job based on the presence or absence of a file



- If the number of records is 0 (if there is a chkfile), "dataput" is skipped and "datadel" is executed.
- If the number of records is not 0 (if there is no chkfile), "dataput" is executed and then "datadel" is executed.

By creating a processing flow like that shown in the figure above, you can also detect abnormalities in other processing (commands) executed in the executable file *lenchk*, and execute the dependent job *dataput*.

Note that judgments based on the presence or absence of files are used to check files at the host where the work task manager (JP1/AJS3 - Manager) is installed. Depending on the environment settings, you might be able to check the presence or absence of files on other hosts. However, because communication problems or other issues can make the result of the judgment potentially unreliable, we recommend that judgments based on the presence or absence of files specify files on the manager host. For details, see *3.1.1(1)(c) Judgment job* in the manual *JP1/Automatic Job Management System 3 Overview*.

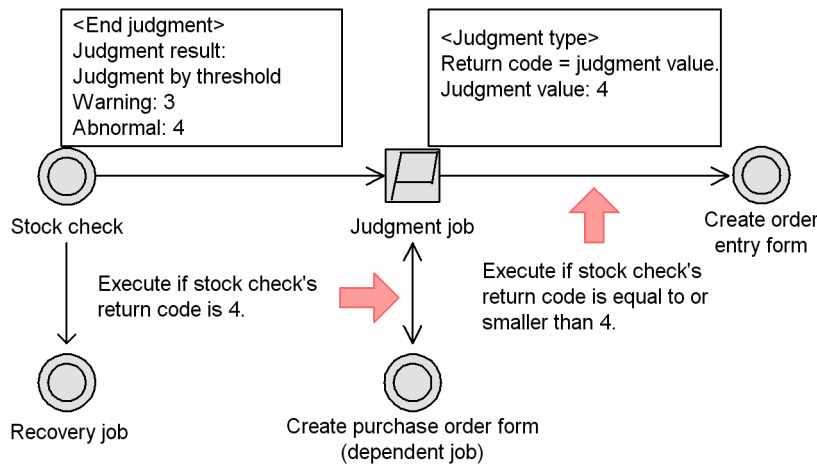
## (c) Rerunning dependent units

Dependent units are rerun differently than normal units. For precautions on rerunning dependent units, see *4.5.11 Rerunning a job or jobnet* in the manual *JP1/Automatic Job Management System 3 Overview*.

## (2) Example of defining a jobnet that uses a judgment job

In the example below, a judgment job is used to define a jobnet that creates a purchase order form if the stock level is slightly down, or an order entry form if there is adequate stock, depending on the execution result of a job that checks the stock level. If there is a definite stock shortage, the stock check job ends abnormally and a recovery job is executed. For details on recovery jobs, see *2.4.6 Executing a specific process when a job ends abnormally (example of defining a jobnet that uses a recovery unit)*.

Figure 2–69: Example of defining a jobnet that uses a judgment job



In this example, the judgment depends on the return value of the preceding job. The *stock check* job returns 3 or a lower value if there is adequate stock, 4 if the level is slightly down, and 5 or higher if there is a stock shortage.

First, set **Judgment by threshold** as the end judgment of the *stock check* job. Set the warning threshold to 3 and the abnormal threshold to 4. With these settings, the *stock check* job is judged by its return value, and its status changes as follows:

- Return value is 3 or lower: *Ended normally*
- Return value is 4: *Ended with warning*
- Return value is 5 or higher: *Ended abnormally*

Set the judgment job to execute a dependent job if the *stock check* job's return code equals the judgment value of 4.

Accordingly, the succeeding job of the *stock check* job is as follows:

- When the return value of the *stock check* job is 3 or lower:  
The return value does not match the judgment value of 4 set in the judgment job. Therefore, the dependent job *create purchase order form* is skipped and the *create order entry form* job is executed.
- When the return value of the *stock check* job is 4:  
The return value matches the judgment value set in the judgment job, so the dependent job *create purchase order form* is executed first. At normal termination of the dependent job, the *create order entry form* job is executed.
- When the return value of the *stock check* job is 5 or higher:  
A recovery job is executed. Because no succeeding job is executed in the event of an abnormal termination, the *create order entry form* job is not executed after the recovery job.

You can define two or more judgment jobs in succession. This allows you to define a number of processing paths equivalent to the number of defined judgment jobs, each depending on the end result of the preceding job.

The following explains how to use multiple judgment jobs.

### (3) Example of using a succession of judgment jobs

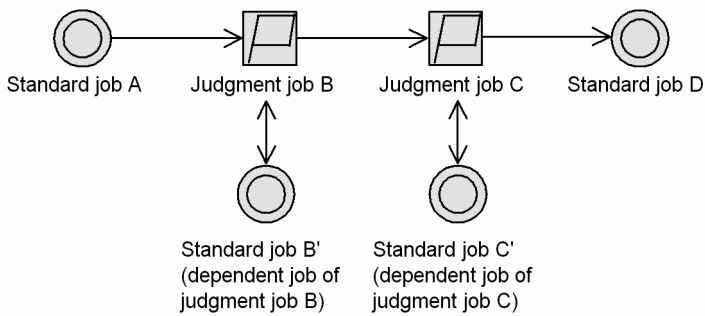
You can define two or more judgment jobs in succession.

When you define multiple judgment jobs based on a return code, the second and subsequent judgment jobs evaluate the return code set in the preceding job of the first judgment job, not the return code of the dependent unit.

Conversely, when you define multiple judgment jobs based on the presence or absence of a file, the second and subsequent judgment jobs evaluate the file information current at the time they make their evaluation, not the file information current when the first judgment job performed its evaluation.

An example of defining a succession of judgment jobs is shown below.

Figure 2–70: Example of using a succession of judgment jobs



Suppose that each unit is defined as follows:

- Standard job A: **End judgment** is set to **Judgment by threshold**, and 5 is entered in the **Warning** field
- Judgment job B: The judgment type is **Return code = judgment value.**, and 0 is specified for **Judgment value.**
- Judgment job C: The judgment type is **Return code = judgment value.**, and 4 is specified for **Judgment value.**

When you execute this jobnet, the return value of standard job A determines the succeeding job, as follows:

- When standard job A returns 0
 

The return code matches the condition set for judgment job B. Therefore, standard job B' (the dependent job of judgment job B) is executed. Next, judgment job C evaluates the return code. Because the return code does not match the condition set for judgment job C, C's dependent job (standard job C') is not executed. Instead, standard job D is executed.
- When standard job A returns 4
 

Because this return code does not match the condition set for judgment job B, B's dependent job (standard job B') is not executed. Judgment job C now evaluates the return code. Because it matches the condition set for C, C's dependent job (standard job C') is executed, followed by standard job D.
- When standard job A returns a value other than 0 or 4
 

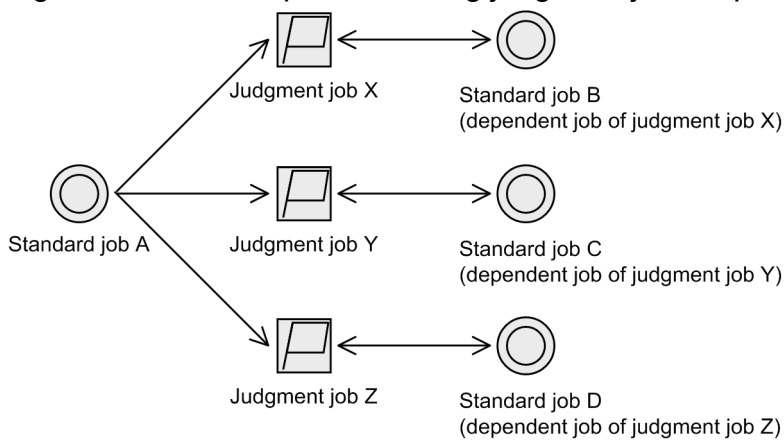
Because the return code matches neither of the conditions for judgment jobs B and C, neither of their dependent jobs (standard jobs B' and C') is executed. Only standard job D is executed.

#### (4) Example of defining judgment jobs in parallel

You can define judgment jobs in parallel.

The following figure shows an example of defining judgment jobs in parallel.

Figure 2–71: Example of defining judgment jobs in parallel



The units are defined as follows.

Standard job A

**End judgment: Judgment by threshold**

**Warning:** 30

Judgment job X

**Judgment type** is **Return code** and **Condition** is **Return code within judgment value range.**

Greater than or equal to: 1

Less than or equal to: 9

Judgment job Y

**Judgment type** is **Return code** and **Condition** is **Return code within judgment value range.**

Greater than or equal to: 10

Less than or equal to: 19

Judgment job Z

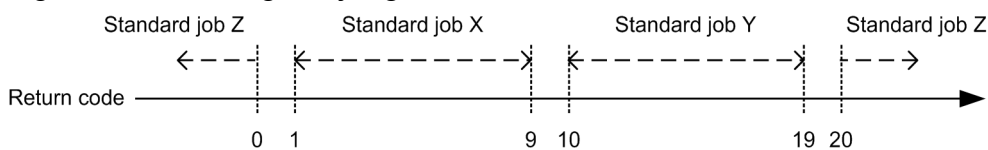
**Judgment type** is **Return code** and **Condition** is **Return code not within judgment value range.**

Greater than or equal to: 1

Less than or equal to: 19

The following figure shows the range of judgment values for each judgment job.

Figure 2–72: Range of judgment values



When you execute this jobnet, the return code of standard job A determines the succeeding job as follows:

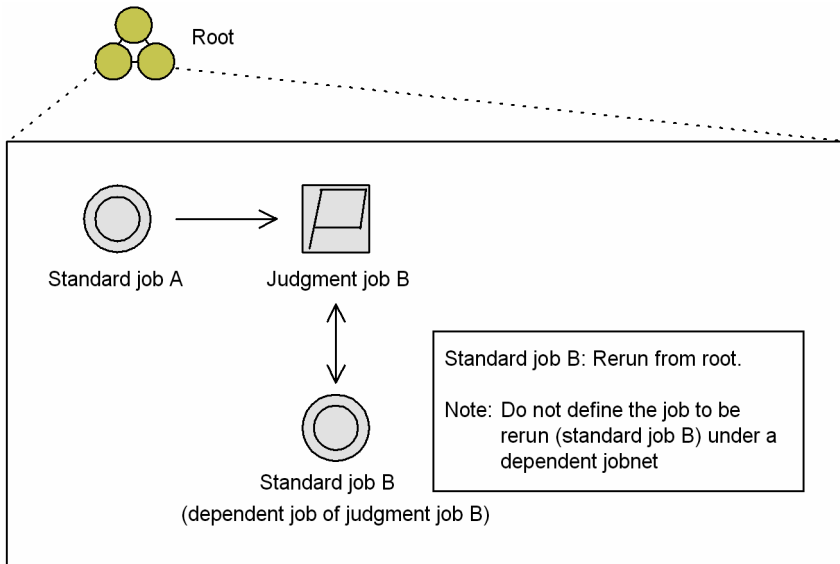
- When standard job A returns a code in the range from 1 to 9  
The return code matches the condition set for judgment job X. Therefore, standard job B (the dependent job of judgment job X) is executed.
- When standard job A returns a code in the range from 10 to 19  
The return code matches the condition set for judgment job Y. Therefore, standard job C (the dependent job of judgment job Y) is executed.

- When standard job A returns a code smaller than 1 or greater than 19  
The return code matches the condition set for judgment job Z. Therefore, standard job D (the dependent job of judgment job Z) is executed.

## (5) Cautionary notes

If you want to rerun a root jobnet or the preceding unit based on the result of a judgment job, define the job that you want to rerun as a dependent job as shown in the following figure:

Figure 2–73: Example of rerunning a root jobnet from the result of a judgment job

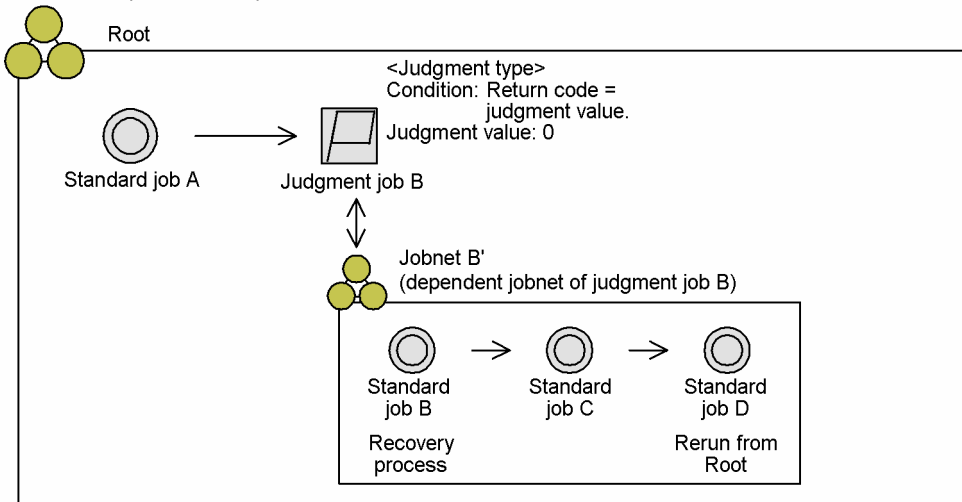


If you define the job that you want to rerun under a dependent jobnet, the dependent jobnet might execute again depending on the termination timing of the rerun job.

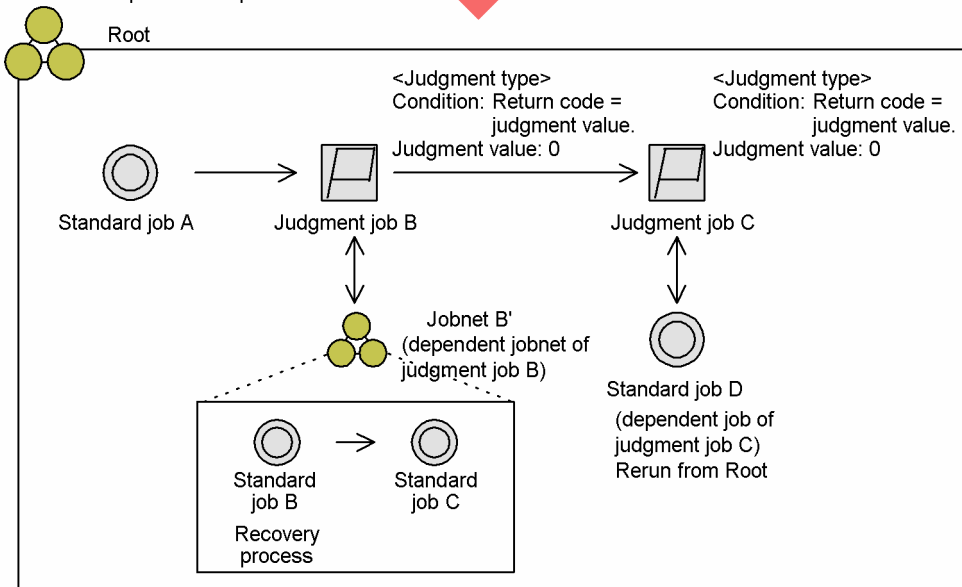
In the first of the two definition examples in the figure below, recovery and rerun processing are defined in a single dependent jobnet. If you want to implement a recovery process before rerunning a job, you must define a dependent jobnet that performs the recovery process and a dependent job that reruns the root jobnet, as shown by the second improved example.

Figure 2–74: Definition examples incorporating a recovery process before re-execution

■ Definition example: Before improvement



■ Definition example: After improvement



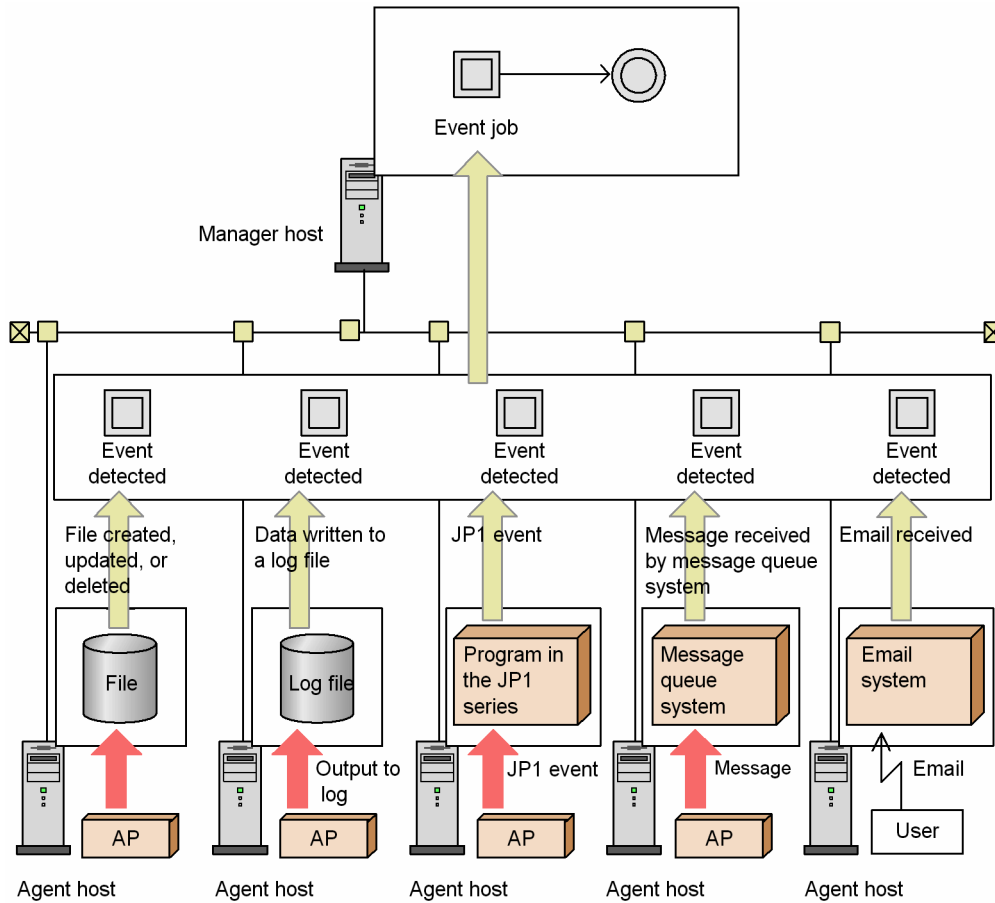
## 2.4.4 Executing an event-driven process (example of defining a jobnet that uses an event job and a custom event job)

Use an event job when defining a jobnet in which a process is triggered by an occurrence such as event receipt or file update. If you want to use an event as a trigger to execute a process and want a linked program to monitor occurrence of the event, define a jobnet by using a custom event job.

An event job can be used to monitor events on any manager host or agent host on the network. The following figure shows the types of events that can be monitored.



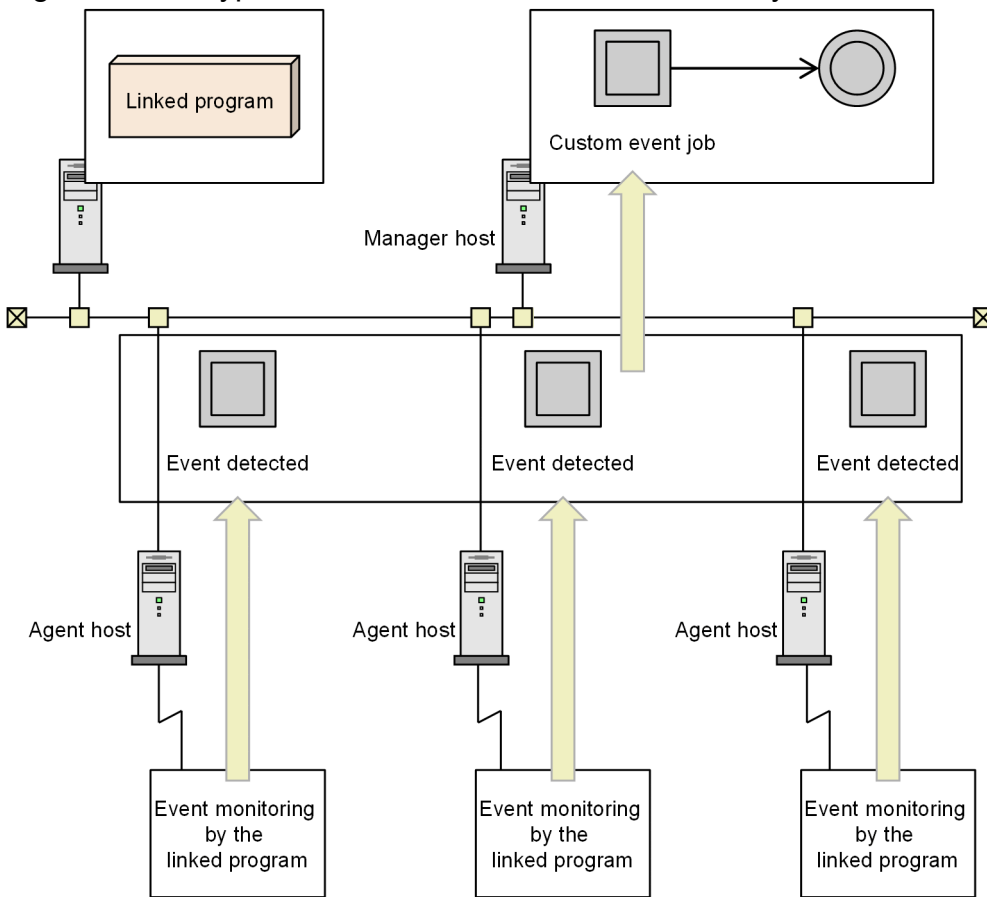
Figure 2–75: Types of events that can be monitored by an event job



Legend:  
 AP :Application program

A custom event job can link with a non-JP1/AJS3 program to monitor various events that the linked program monitors. The following figure shows an overview of event monitoring by using a custom event job.

Figure 2–76: Types of events that can be monitored by a custom event job



#### Timing of event detection by an event job and a custom event job

There might sometimes be a time lag between the execution time of an event job or a custom event job and the time that it is ready to monitor events. Events occurring during this time lag will not be detected. You should bear this in mind when defining a jobnet that uses an event job or a custom event job.

For Receive JP1 event jobs, you can use the find events prior to execution setting to solve this problem (see *(1) Executing a process on receipt of a JP1 event (Receive JP1 event job)* below.)

#### Using randomly occurring events to trigger a jobnet

When an event occurs more than once at irregular times, we recommend that you set a start condition for the jobnet triggered by that event. For details on setting a start condition, see *3.4 Defining a start condition* in the manual *JP1/Automatic Job Management System 3 Overview*.

If you know that the event you are monitoring occurs in a predictable manner, you can define an event job and a custom event job at the start of the jobnet, rather than defining a start condition.

#### Setting a timeout period for an event job and a custom event job

When a timeout period is set for an event job and a custom event job, the period is counted at the host on which the job is executed. If a power failure or other problem occurs at the target host and event monitoring is resumed after the host is restarted, the timeout period is counted again, beginning from the restart time. To suspend monitoring by the event job and the custom event job at an absolute time, counted from the execution start time regardless of the host status, define an event job and a custom event job as the jobnet start condition and set the valid range of start condition to absolute time. For details on start conditions, see *3.4 Defining a start condition* in the manual *JP1/Automatic Job Management System 3 Overview*.

You must also specify what status the event job or the custom event job is to be placed in after the timeout period specified for the job has elapsed. Select *Killed*, *Ended normally*, *Ended with warning*, or *Ended abnormally* status (the default is *Killed* status).

By setting an event job or a custom event job in this way, you can choose to cancel or continue the jobnet after the event job's timeout period elapses.

### Passing event information

You can define the event information received by an event job and a custom event job as a variable (macro variable) that is passed to the succeeding unit. For details on passing event information, see (6) *Passing information received by an event job and a custom event job* below.

An example of an event job and a custom event job defined in a jobnet is given below. For additional information on defining an event job and a custom event job in a jobnet, see 7.6 *Notes on using event jobs* and 7.10 *Notes on using custom event jobs*.

JP1/AJS3 must be linked with the appropriate program to use a Receive mail job. For details, see the *JP1/Automatic Job Management System 3 Linkage Guide*.

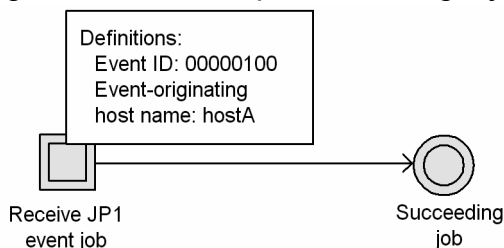
## (1) Executing a process on receipt of a JP1 event (Receive JP1 event job)

A *JP1 event* is an event managed by JP1/Base, issued whenever some event occurs in a JP1 program. Because JP1 events contain information such as a severity level (error, warning, or notice) or message, you can execute a succeeding job or jobnet on receipt of an error or warning, or only when a specific message is received. Using the JP1/Base event converter, you can also execute a succeeding job or jobnet at termination of an external application program.

To execute a Receive JP1 event job, you must first start the JP1/Base event service. (In the API settings file, set `keep-alive` for the JP1/Base event service.) If this service is inactive, the Receive JP1 event job is placed in the *Now running* status until the service starts.

The following figure shows an example of defining a jobnet that executes a certain job when the Receive JP1 event job receives a JP1 event from hostA, which differs from the host where the jobnet was defined.

Figure 2–77: Example of defining a jobnet that uses a Receive JP1 event job



A Receive JP1 event job can link with a jobnet defined on another host.

On hostA, define a Send JP1 event job that sends an event with the event ID of 100. To monitor any JP1 events issued by hostA, set hostA in the Receive JP1 event job as the event-originating host name. In addition, to execute the succeeding job when an expected JP1 event is received, set 100 as the event ID.

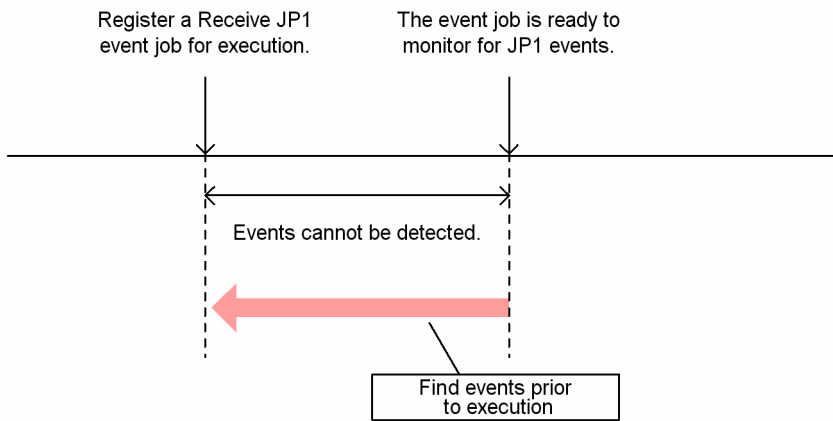
After these settings have been made, if the Send JP1 event job in the jobnet is executed on hostA, the jobnet stops monitoring JP1 events and executes the succeeding job.

Note that the system will not detect any JP1 event occurring during the time lag after execution of the Receive JP1 event job until it is actually able to monitor for receipt of JP1 events. One solution to this problem is the *find events prior to execution* setting.

- Find events prior to execution

This feature searches for events occurring before an executed Receive JP1 event job is in ready state. Any matching event is identified as having occurred.

Figure 2–78: Find events prior to execution



## (2) Executing a process at file update (Monitoring files job)

Use a Monitoring files job when defining a jobnet in which file update or file creation triggers a job.

The monitoring options that you can specify for a Monitoring files job are described below.

### Monitoring options

Specify the file status required as the condition to end monitoring and initiate the job. You can set any of the following four monitoring options:

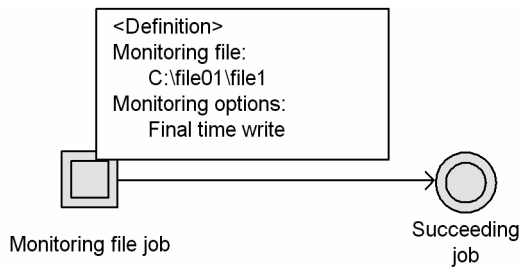
- Creation of a file of the specified file name (**Create**)
- Deletion of a file of the specified file name (**Delete**)
- Change in size of a file of the specified file name (**Change size**)
- Change in last update time of a file of the specified file name (**Final time write**)

### Supplementary notes

- When creation of a file with the specified name is being monitored, that file might already exist at the time the Monitoring files job starts. You can specify whether the job should assume that the monitoring condition has been satisfied if a file with the specified name already exists.
- You can specify more than one monitoring condition. For example, if you want to execute a succeeding job when a file is deleted or updated, you can specify the **Delete** and **Final time write** options. Note, however, that the **Change size** and **Final time write** options are mutually exclusive.
- For details on when a monitoring condition is or is not satisfied, see [7.6.2\(1\) Events monitored by the Monitoring Files job](#).
- If any of the **Create**, **Change size**, and **Final time write** monitoring conditions are satisfied, the system performs a *close check* to make sure the monitored file is not being accessed by any process other than the Monitoring files job. If another process is accessing the file, the condition is judged to be unsatisfied, and another close check is performed when the next monitoring interval occurs. If the file is not being accessed by a process other than the Monitoring files job, the condition is judged to have been satisfied.  
The close check prevents the job from assuming that the condition is satisfied before the transmission (for example, copying) of a monitored file has been completed.
- Files can be specified as monitoring targets over a network.

In the example below, a Monitoring files job is used to define a jobnet that monitors the write time to a particular file (file name: File1) and executes a succeeding job when the file is updated.

Figure 2–79: Example of defining a jobnet that uses a Monitoring files job



Specify the file to monitor and the monitoring option in the definition of the Monitoring files job. In the **Monitoring file** field, enter the path for File1, and in the **Monitoring option** area, select **Final time write**.

In this example, the Monitoring files job ends and the triggering condition is satisfied when the monitored file closes (no further applications are accessing the file) and the last update time changes.

To monitor files via a network, set `Y` for the `NetworkFilewatch` environment setting parameter. For details about the settings for monitoring files via a network, see [6.3.19 Settings for monitoring a network file by using a file monitoring job](#) in the *JP1/Automatic Job Management System 3 Configuration Guide* (in Windows) or [15.3.18 Settings for monitoring a network file by using a file monitoring job](#) in the *JP1/Automatic Job Management System 3 Configuration Guide* (in UNIX).

In addition, specify the access permission settings for monitoring-target files as follows:

#### In Windows

Prepare a user account with permission to change the monitoring-target files, and then set the user account as the user account for starting the JP1/AJS3 service. For details about changing the user account for starting the JP1/AJS3 service, see [4.2.3 Changing the JP1/AJS3 service settings \(Windows only\)](#) in the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*.

#### In UNIX

Settings must be made to ensure that view permissions can be used to access the network file specified as a monitoring target from the client.

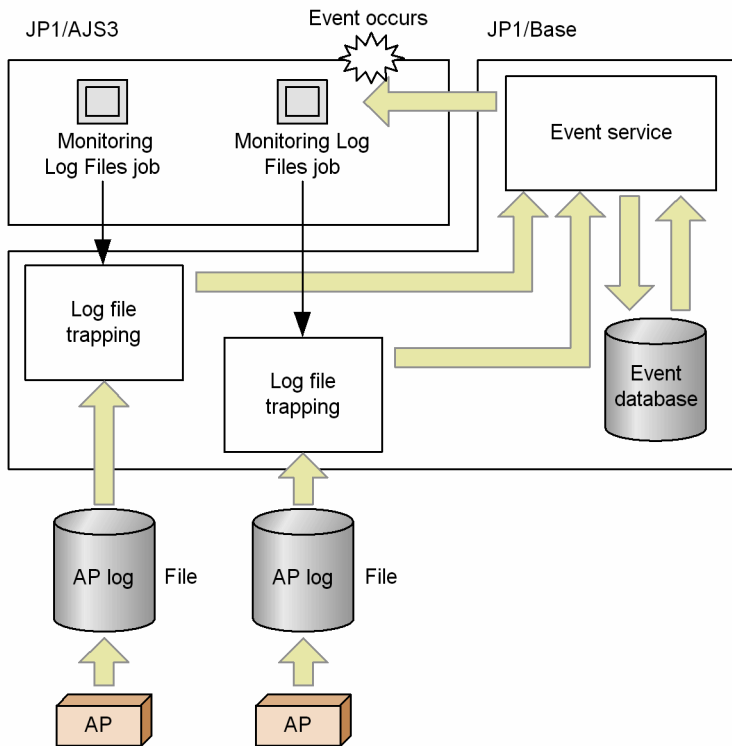
If you monitor files shared by using protocol other than SMB (for example, NFS) in a Windows environment or monitor files via a network in a UNIX environment, and if there is a possibility that a process on another host (a host other than the agent host running the applicable job) might open the files, we recommend that you set the system to monitor a file that is created to indicate processing completion when the monitoring-target file updating ends. Do not use a file monitoring job to directly monitor files to be updated. For notes on monitoring files shared by using protocol other than SMB in a Windows environment or monitoring files via a network in a UNIX environment, see [7.6.2 Notes on the Monitoring Files job](#).

### (3) Executing a process at log file update (Monitoring log files job)

A Monitoring log files job utilizes the JP1/Base log file trapping facility. JP1/Base log file trapping converts the log file records output by an application into JP1 events, and registers them in an event database. By defining a Monitoring log files job, you can execute a job or jobnet when a log file is updated. For details on JP1/Base log file trapping, see the *JP1/Base User's Guide*. For notes about Monitoring log files jobs, see [7.6.4 Notes on the Monitoring Log Files job](#).

The following figure shows how a Monitoring log files job operates.

Figure 2–80: Operation of a Monitoring log files job



Legend:

→ : Execute job

→ : Data flow

AP : Application program

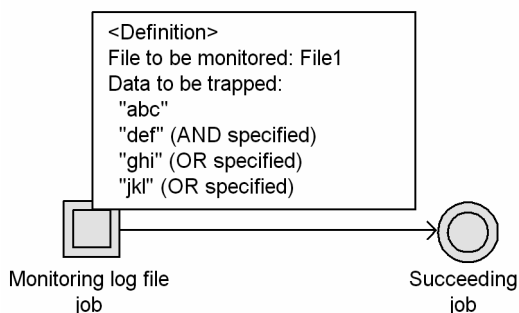
A Monitoring log files job monitors for output of specific data to a log file on the host specified as the monitoring target. In this job, you specify the log file and the character string to be monitored. Regular expressions can be used to specify the character string (see the *JP1/Base User's Guide* for regular expressions in Windows, or your UNIX documentation for regular expressions in UNIX).

As the monitoring interval, you can specify a value from 1 to 86,400 (seconds). Only log files in text format can be monitored. You can monitor up to eight log files.

An example of setting a Monitoring log files job as the triggering condition is shown below.

In this example, the Monitoring log files job is used to define a jobnet that executes a succeeding job when log data containing a specific character string is written to the log file (file name: File1).

Figure 2–81: Example of defining a jobnet that uses a Monitoring log files job



Specify the log file name and the character strings to be monitored in the definition of the Monitoring log files job. In the **File to be monitored** field, enter the path for File1. In the **Data to be trapped** area, enter the desired character strings as shown in the figure.

In this example, the Monitoring log files job ends when data matching any of three conditions (a string containing "abc" and "def", a string containing "ghi", or a string containing "jkl") is written to File1 and the log data is retrieved from the log file. This satisfies the triggering condition and the succeeding job is executed.

You can also configure a Monitoring log files job to monitor multiple log files and execute the succeeding job when log data containing a specific character string is written to any of the monitored files.

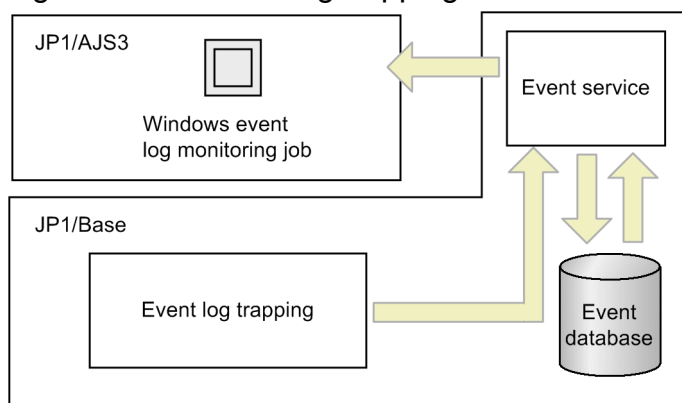
#### (4) Executing a process on receipt of a Windows event log record (Monitoring event log job)

A Monitoring event log job utilizes the JP1/Base event log trapping facility, which converts Windows event log records into JP1 events and registers them in an event database. By defining a Monitoring event log job, you can execute a job or jobnet when a Windows event is logged.


For details on JP1/Base event log trapping and the action definition file, see the *JP1/Base User's Guide*. For notes about Monitoring event log jobs, see [7.6.5 Notes on the Monitoring Event Log job](#).

The following figure shows how a Monitoring event log job operates.

Figure 2–82: Event log trapping



Legend:

 : Data flow

A Monitoring event log job works as follows.

Log type

The types of logs that you can specify are listed below:

- **Application**
- **Security**
- **System**
- **DNS Server**
- **Directory Service**
- **File Replication Service**

- **Any log type**<sup>#</sup>

#

For details about how to check the log types that can be specified for **Any log type**, see the *JP1/Base User's Guide*.

Event types

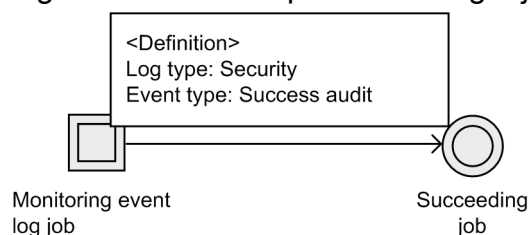
You can specify the following types of events:

- **Verbose**
- **Information**
- **Warning**
- **Error**
- **Critical**
- **Success audit**
- **Failure audit**

An example of defining a Monitoring event log job is shown below.

In this example, the Monitoring event log job is used in a jobnet that executes a succeeding job when a Windows event reporting successful authentication on the security system is logged to the Windows event log.

Figure 2–83: Example of defining a jobnet that uses a Monitoring event log job



Before executing a Monitoring event log job, you must set the event log trapping behavior in the JP1/Base action definition file. In this example, the log type is `Security`, and the attribute name (with `Type` set) is `Audit_success`.

You must also set the log type and event type to be monitored. Set **Security** in **Log type**, and **Success audit** in **Event type**.

With these settings, the Monitoring event log job ends when the specified Windows event is output and the event log data is retrieved. This satisfies the triggering condition and the succeeding job is executed.

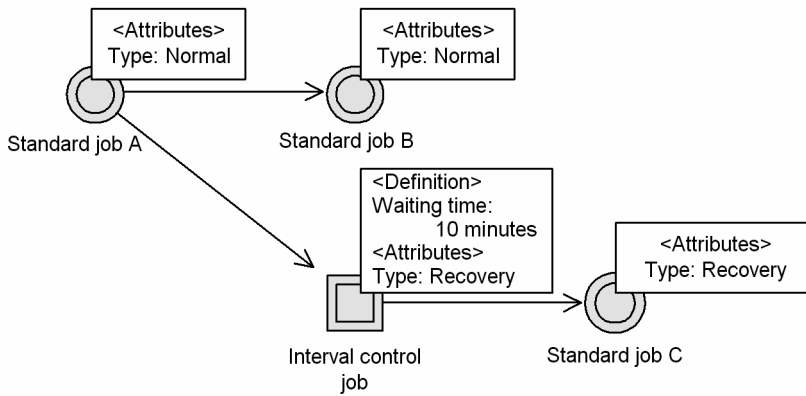
## (5) Executing a process after a specified wait time (Interval control job)

Use an Interval control job when defining a jobnet in which a job is executed after a specified wait time.

In the example below, an Interval control job is used to define a jobnet that executes a recovery job 10 minutes later if the succeeding job ends abnormally.



Figure 2–84: Example of defining a jobnet that uses an Interval control job



If the preceding job ends abnormally, the next process is executed 10 minutes later. Therefore, use an Interval control job to monitor the time period. Set 10 minutes in **Waiting time**. Because the Interval control job will be executed at abnormal termination, set **Recovery** in the job **Type**. Likewise, set **Recovery** in **Type** for standard job C (the Interval control job's succeeding job).

With these settings, if standard job A ends abnormally, the Interval control job (recovery job) monitors the time until 10 minutes has passed, then standard job C (the succeeding job) is executed. If standard job A ends normally, standard job B is executed.

The **Waiting time** specified in an Interval control job refers to the wait time of the interval control process, not the time required to execute the Interval control job. Depending on the communication status or other factors, there might be a difference between the specified wait time and the actual interval.

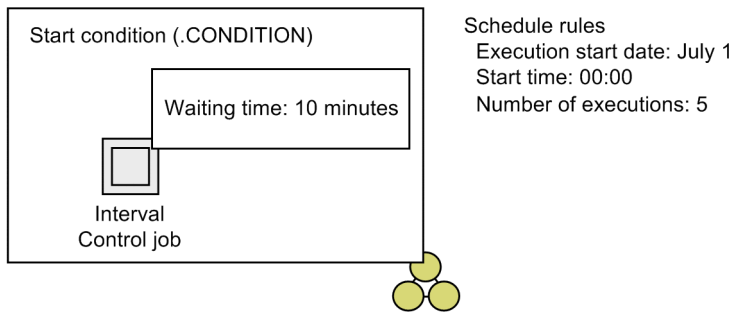
### (a) Defining the Interval Control job as a start condition

When you define the Interval Control job as a start condition, you can choose whether to satisfy the start condition as soon as the Interval Control job starts executing.

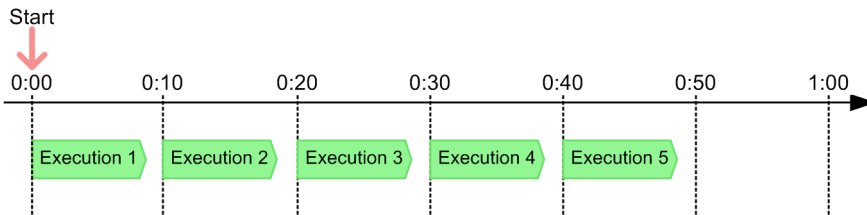
If you want the start condition to be satisfied immediately after the job has started executing, in the Detailed Definition - [Interval Control] dialog box, in the **Expire right after starting** section, click **Yes**. If you do not want the start condition to be satisfied immediately after the job has started executing and instead want to wait for the length of time specified in the **Waiting time** section for execution to start, in the **Expire right after starting** section, click **No**.

The following figure shows the behavior of two cases when five executions from 0:00 on July 1 have been specified in the Schedule Rule dialog box.

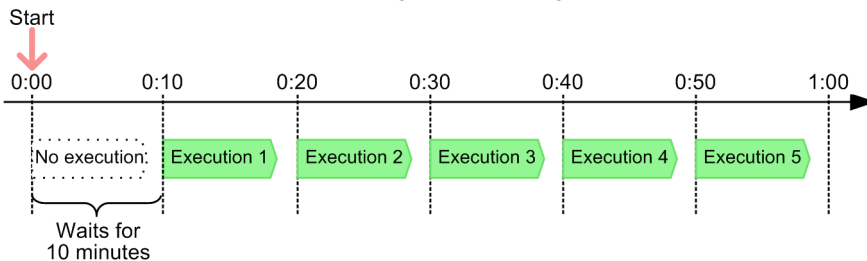
Figure 2–85: Example of defining the Interval Control job as a start condition



Example 1: When **Yes** is chosen for **Expire right after starting**



Example 2: When **No** is chosen for **Expire right after starting**



In example 1, **Yes** is chosen for **Expire right after starting**. In this case, the monitoring for the start condition starts at 0:00 on July 1. The start condition is quickly satisfied and the first execution of the jobnet begins. After that, the jobnet is executed every 10 minutes as specified in the **Waiting time** section.

In example 2, **No** is chosen for **Expire right after starting**. In this case, the monitoring for the start condition starts at 0:00 on July 1. After the 10 minutes specified in the **Waiting time** section have elapsed (at 0:10), the first execution of the jobnet begins. Thereafter, the jobnet is executed every 10 minutes.

When you click **No** for **Expire right after starting**, for the start time, you need to specify a period of time equal to the waiting time that is earlier than the time that the first execution of the jobnet starts.

#### Cautionary notes

- **Expire right after starting** is available when the version of JP1/AJS3 - Manager or JP1/AJS3 - Agent on the execution target host of the Interval Control job is 10-00 or later, and the version of JP1/AJS3 - View is 10-00 or later. If the version of JP1/AJS3 - Manager is 10-00 or later and the version of JP1/AJS3 - Agent on the execution target host is earlier than 10-00, specify the default execution agent (@SYSTEM) running on JP1/AJS3 - Manager as the execution agent. If the version of JP1/AJS3 - Manager or JP1/AJS3 - Agent on the execution target host is earlier than 10-00, the Interval Control job changes to the *Ended abnormally* status.
- If you kill the JP1/AJS3 service or the scheduler service when **Yes** is chosen for **Expire right after starting**, the jobnet starts executing after the time specified in the **Waiting time** section has elapsed the next time the jobnet starts.

## (6) Passing information received by an event job and a custom event job

The event information received by an event job and a custom event job can be defined as a variable (macro variable) that is inherited by the succeeding unit. This inherited information is called *passing information*.

To pass event information to a succeeding unit, you must define a macro variable in the event job or the custom event job and specify the same macro variable name in the succeeding unit.

In the event job or the custom event job, specify the macro variable in the following form:

Macro variable definition

*macro-variable-name* : *passing-information-name*

For *macro-variable-name*, specify a character string of no more than 64 characters in the format ?AJS2xxxx?. In *xxxxx*, you can use uppercase alphabetic characters (A to Z), numerals (0 to 9), and periods (.). In *passing-information-name*, you can specify only the passing information supported for the particular type of event job and custom event job. For details, see *B. Information Passed by Event Jobs and Custom Event Jobs*.

At execution of a macro variable, the passed information is expanded in the command line on the target host that executes the succeeding unit of the event job and the custom event job. You should define a macro variable only if you are sure that the information to be passed is of a form that can be handled as a command argument when the succeeding job is executed.

For details on macro variables, see *2.2.6 Considerations regarding the use of macro variables*.

The following explains how the information received by event jobs and custom event jobs is passed to the various types of succeeding units.

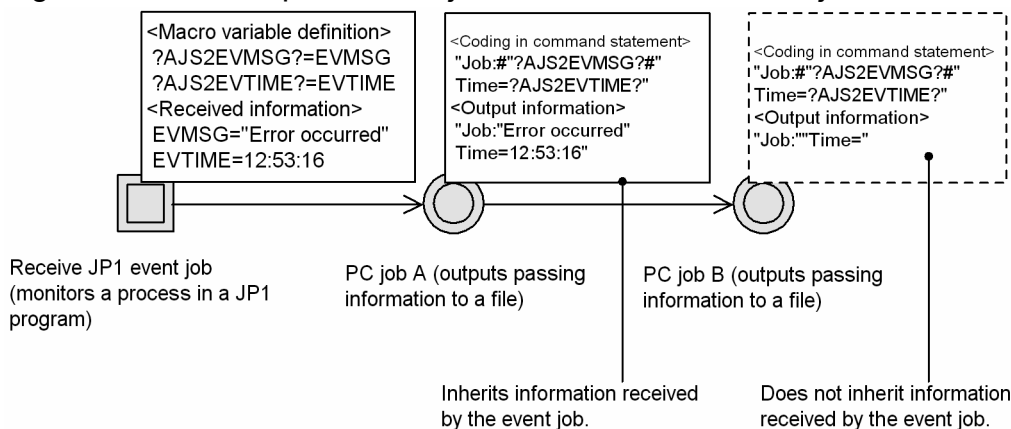
To make event job and custom event job information available to an entire jobnet, use a start condition. (For details on start conditions, see *3.4 Defining a start condition* in the manual *JP1/Automatic Job Management System 3 Overview*.)

### (a) Standard job, HTTP connection job or action job defined after an event job

When you define a standard job (PC job, Unix job, or flexible job), HTTP connection job, or action job as a succeeding job of an event job or a custom event job, you can use macro variables in the event job or the custom event job and succeeding job. Macro variables can represent non-constant string items, such as command text, script file names, parameter names, and environment variables. By using macro variables, the event job or the custom event job can pass received event information to the succeeding job.

The following figure shows an example of a PC job defined after an event job.

Figure 2–86: Example of a PC job defined after an event job

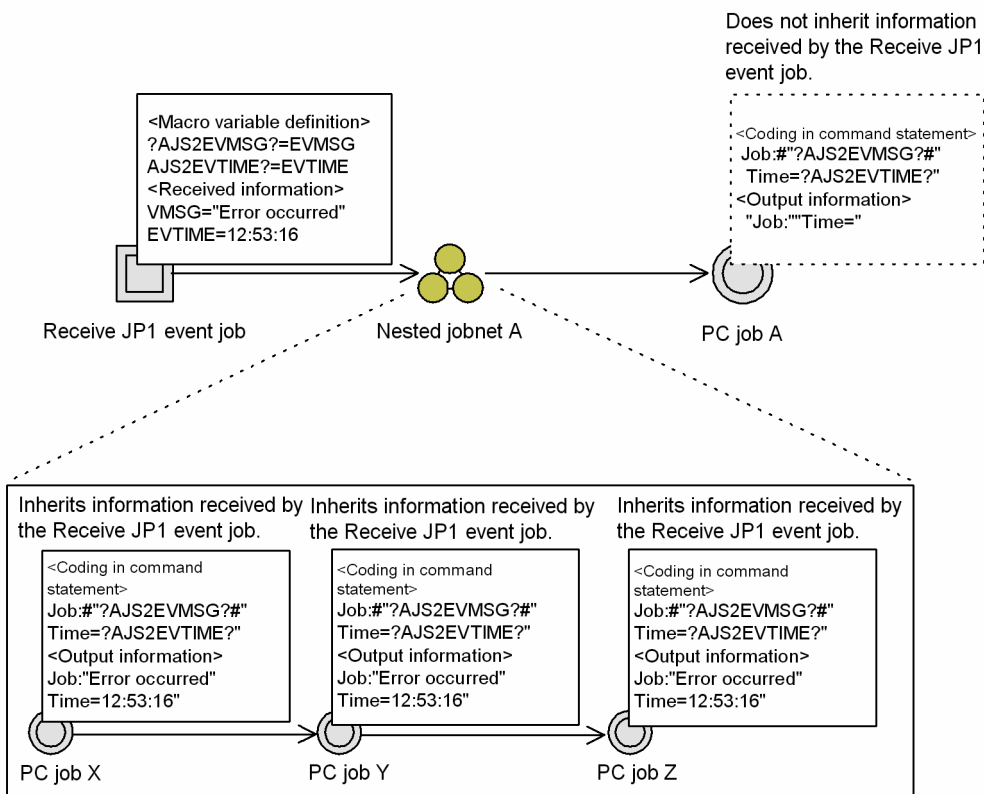


In the above example, the event information received by the JP1 event reception monitoring job is passed to PC job A. The event information is not passed from PC job A to PC job B, which is the succeeding job of PC job A. The information is not passed to PC job B, even if PC job A ends abnormally.

### (b) Nested jobnet defined after an event job

By specifying a macro variable name in a nested jobnet defined as the succeeding unit of an event job or a custom event job, you can pass the received event information to the standard jobs (PC jobs, Unix jobs, or flexible jobs), HTTP connection jobs and action jobs defined in the nested jobnet. The following figure shows an example of a nested jobnet defined after an event job.

Figure 2–87: Example of a nested jobnet defined after an event job

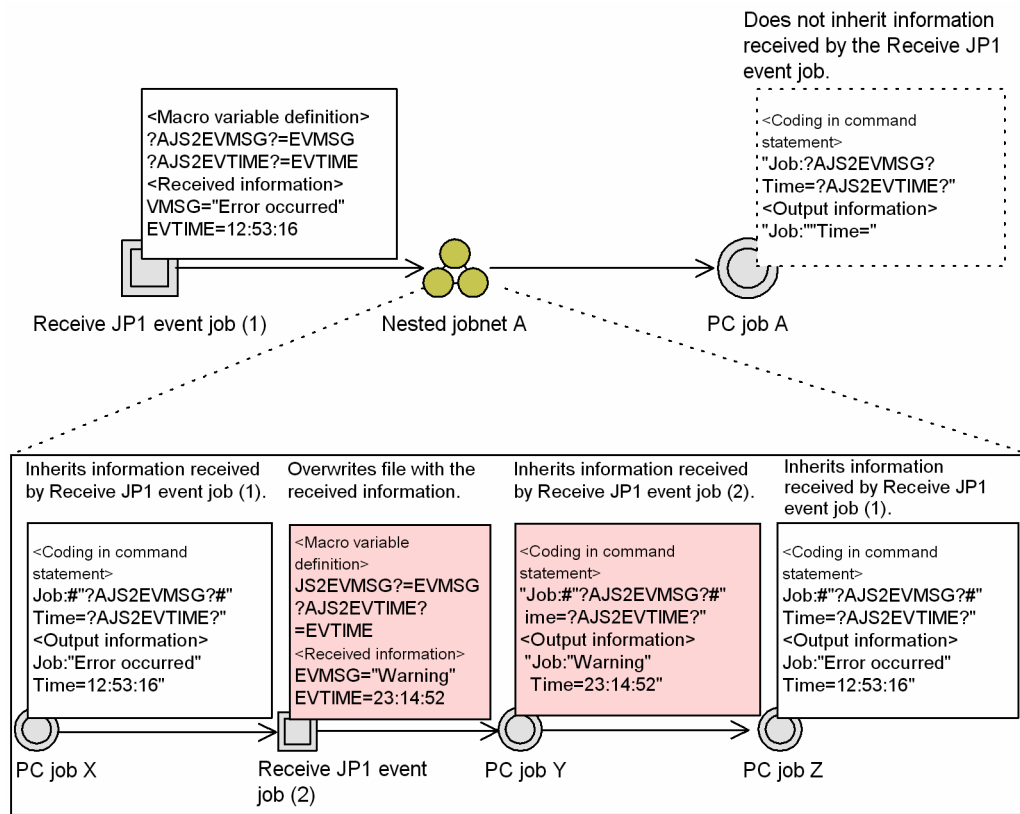


Note: Assume that each standard job inherits the macro variable and outputs data to file.

In the above example, the event information received by the Receive JP1 event job is passed to PC jobs X, Y, and Z defined in nested jobnet A.

If the same event job and custom event job with the same macro variable is also defined within the nested jobnet, the information received by the nested event job or custom event job takes precedence. The following example shows the same event job and macro variable defined as a preceding job and also within the nested jobnet.

Figure 2–88: Example of the same event job defined in a nested jobnet



Note: Assume that each PC job inherits the macro variable and outputs data to file.

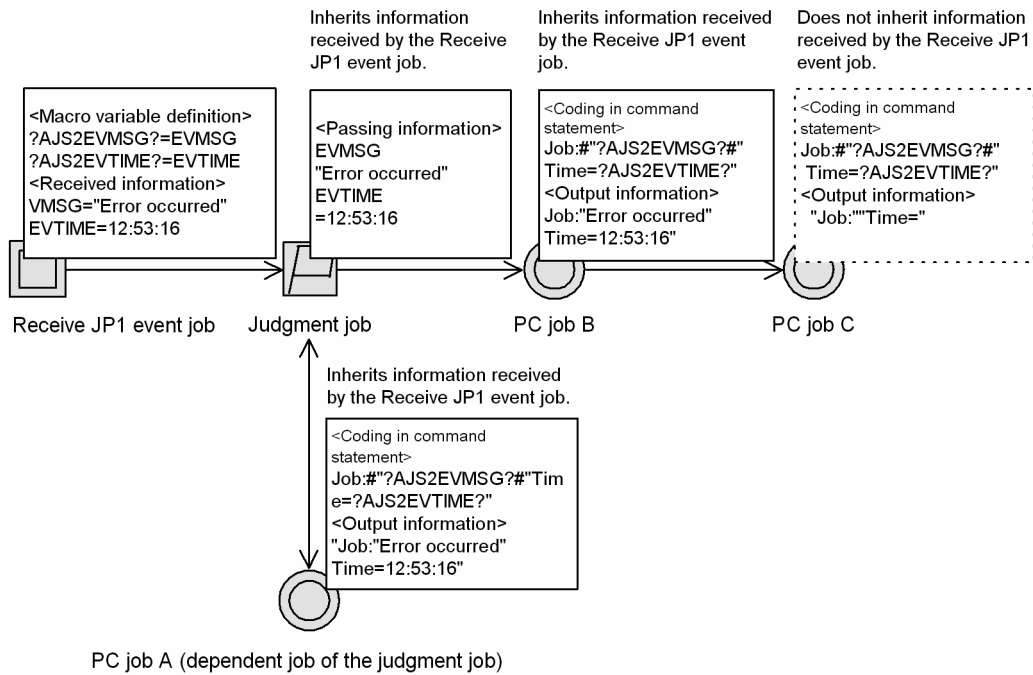
In this example, PC job X inherits event information from Receive JP1 event job (1), whereas PC job Y inherits the information received by Receive JP1 event job (2). The macro variable defined in Receive JP1 event job (1) is specified in PC job Z, so the information received by Receive JP1 event job (1) is passed to PC job Z.

### (c) Judgment job defined after an event job

By specifying a macro variable name in the dependent job of a judgment job that follows an event job or a custom event job, you can pass the information received by the event job or the custom event job to both the dependent job and the succeeding job of the judgment job.

The following figure shows an example of a judgment job defined after an event job.

Figure 2–89: Example of a judgment job defined after an event job



Note: Assume that each PC job inherits the macro variable and outputs data to file.

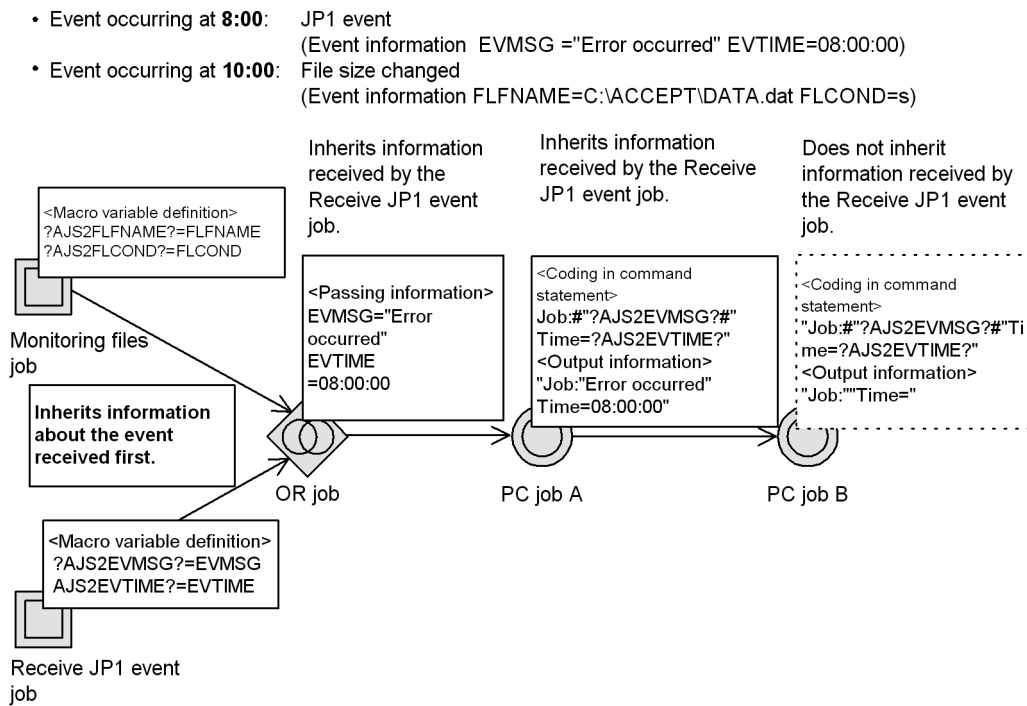
When a judgment job is defined as the succeeding job of an event job or a custom event job, the received information is passed to both the dependent job and the succeeding job of the judgment job. In the above example, the event information received by the Receive JP1 event job is passed to PC job A (the dependent job) and to PC job B (the succeeding job).

#### (d) OR job defined after an event job

By specifying a macro variable name in the succeeding job of an OR job that follows an event job or a custom event job, you can pass the information received by the event job or the custom event job to the succeeding job of the OR job.

The following figure shows an example of an OR job defined after an event job.

Figure 2–90: Example of an OR job defined after an event job



Note: Assume that each PC job inherits the macro variable and outputs data to file.

When an OR job is defined as the succeeding job of an event job or a custom event job, the received information is passed to the succeeding job of the OR job. In the above example, events are being monitored by two types of event jobs: a Monitoring files job and a Receive JP1 event job. A JP1 event occurs at 8:00. The information received by the Receive JP1 event job on detecting this event is passed to the OR job's succeeding job, and the Monitoring files job is placed in *Bypassed* status.

#### Cautionary notes

- Passing information can be passed even if the event job or the custom event job and the succeeding job are executed on different agent hosts. If the hosts use different character sets, the character codes in the passing information are converted. Note, however, that if the character set for the succeeding job does not contain character codes corresponding to the character codes being converted, character conversion will fail.
- When a macro variable is specified in the command line of a succeeding job, the information will not be passed correctly if it contains a space or single quotation mark (').
- Do not pass data containing an escape order to the command line. Enclose the macro variable with double quotation marks ("). This prevents unpredictable behavior should a space be included in the passed information.
- When passing information is used in the command line of a job, any double quotation marks (") included in the information are ignored. For example, AB"C is passed to the succeeding job in the form ABC. This might cause the job to execute incorrectly, depending on the command line constraints of the particular operating system. If the information contains a special character, use an environment variable to pass the information, rather than expanding the macro variable in the command line. Note, however, that by utilizing the feature for enabling double quotation marks, information containing double quotation marks (") can be passed as is. For details on this feature, see 4.3.1(5) *Passing event data containing double quotation marks* in the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*. See also 6.3.4 *Passing event data containing double quotation marks* in the *JP1/Automatic Job Management System 3 Configuration Guide* and 15.3.4 *Passing event data containing double quotation marks* in the *JP1/Automatic Job Management System 3 Configuration Guide*. This feature is not available in JP1/AJS2 - Manager 06-71 or earlier versions.

## Supplementary notes

- When a single succeeding job follows multiple event jobs and custom event jobs, it inherits the information received by all the event jobs and the custom event jobs. However, if you define the same macro variable name for all the event jobs and the custom event jobs, the received event job and custom event job information will be overwritten each time. The succeeding job will be able to reference only the most recently received information.
- When two or more macro variables with the same name are defined in the passing information of an event job or a custom event job, the information defined first is passed to the succeeding job. For example, if the first macro variable is `?AJS2111?:EVID` (which assigns an event ID to `?AJS2111?`), and the second macro variable is `?AJS2111?:EVMSG` (which assigns message information to `?AJS2111?`), the information assigned to this macro variable will be an event ID (EVID).
- If there is no information to be passed from the event job or the custom event job, or if the event job or the custom event job did not execute, the character string representing the macro variable name is passed to the macro variable defined in the succeeding job. For example, if the macro variable name is `?AJS2111?`, the string `?AJS2111?` is passed.
- In Receive JP1 Event jobs, the parentheses characters "(" and ")" can be used in extended regular expressions specified as monitoring conditions. By enclosing a regular expression within parentheses, you can extract the character string corresponding to the regular expression, and pass the character string to the subsequent job as passing information `EVENVn` (n: 0 to 9). For example, suppose that a running Receive JP1 Event job with the following definitions receives a JP1 event that includes the message "Event Test". If the monitoring conditions for the Receive JP1 Event job are satisfied, the character string "Event" is passed if `?AJS2EVENV1?` is specified for the subsequent job, whereas the character string "Test" is passed if `?AJS2EVENV2?` is specified for the subsequent job.

- Message : (.\*)(.\*)

- Passing Information : ?AJS2EVENV1?: EVENV1, ?AJS2EVENV2?: EVENV2

Note that to use extended regular expressions in a Windows environment, you need to specify extended regular expressions as regular expressions to be used in JP1/Base. For details, see the *JP1/Base User's Guide*.

### 2.4.5 Sending a JP1 event at completion of the preceding job or when an event occurs (example of defining a jobnet that uses a Send JP1 event job)

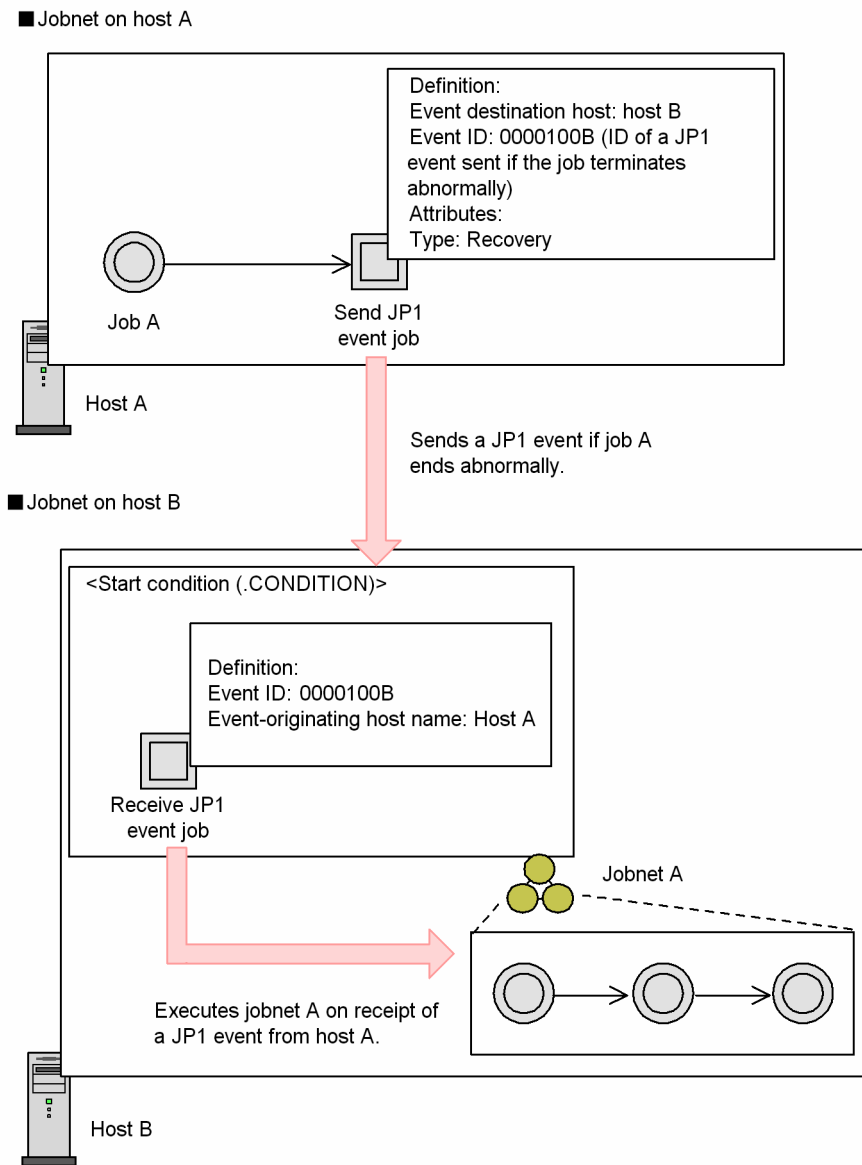
Use a Send JP1 event job when defining a jobnet in which a JP1 event is sent when the preceding job completes or when an event occurs.

To execute a Send JP1 event job, you must first start the JP1/Base event service on both the sending host and the receiving host.

In the example below, a Send JP1 event job is used to define a jobnet in which a JP1 event is sent to host B if a job (job A) ends abnormally on host A. On receipt of the JP1 event, a jobnet to run after abnormal end (jobnet A) is executed on host B.



Figure 2–91: Example of defining a jobnet that uses a Send JP1 event job



On host A, specify a Send JP1 event job as the succeeding job of job A. The Send JP1 event job is executed if job A ends abnormally. Therefore, set **Recovery** in the job **Type**. For details, see [2.4.6 Executing a specific process when a job ends abnormally \(example of defining a jobnet that uses a recovery unit\)](#). If job A ends abnormally, a JP1 event will be sent to host B. Therefore, specify host B in **Event destination host**. The JP1 event ID sent to host B is 0000100B. The JP1 event ID can be set to any user-specified value.

On host B, as the start condition to execute jobnet A, define a Receive JP1 event job for receiving the JP1 event from host A. For details on start conditions, see [3.4 Defining a start condition](#) in the manual *JP1/Automatic Job Management System 3 Overview*. In the Receive JP1 event job, set host A in **Host name** and specify the event ID sent from host A as 0000100B.

#### Event arrival confirmation

You can check whether a JP1 event was actually sent to a specified destination host by a Send JP1 event job. To perform this check, you can set a check interval (seconds) or a check count.

## 2.4.6 Executing a specific process when a job ends abnormally (example of defining a jobnet that uses a recovery unit)

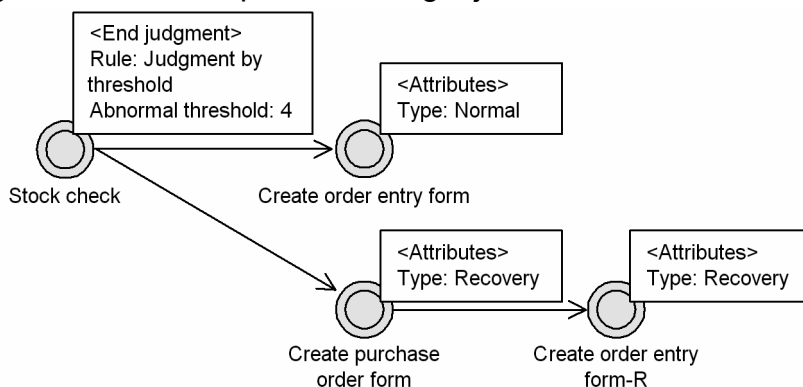
Use a recovery unit when defining a jobnet in which a process is executed only if a job ends abnormally.

You can set a recovery unit by entering **Recovery** in **Type** in the **Attributes** definition of the job or jobnet. Any job or jobnet can be set the recovery attributes.

### (1) Example of defining a jobnet that uses a recovery job

In the example below, a recovery job is used in defining a jobnet that performs a stock check, and then creates an order entry form if there is adequate stock. If there is insufficient stock, the *stock check* job ends abnormally, a purchase order form is created, and then the order entry form is created.

Figure 2–92: Example of defining a jobnet that uses a recovery job



In this example, the *stock check* job returns 5 or a higher value if the execution result indicates a stock shortage.

First, set **Judgment by threshold** as the end judgment of the *stock check* job and set the abnormal threshold to 4. (That is, the *stock check* job ends normally if the return code is 4 or lower, or ends abnormally if the return code is 5 or higher.) If the *stock check* job ends normally, the *create order entry form* job will be executed next. Therefore, leave **Normal** (default) as the *create order entry form* job's **Type** setting. If the *stock check* job ends abnormally, the *create purchase order form* job will be executed next. Therefore, set **Recovery** in **Type** in the **Attributes** definition of the *create purchase order form* job. When this job ends, the *create order entry form-R* job will be executed. Set **Recovery** in **Type** for this job too (because only a recovery unit can be defined as the succeeding unit of a recovery unit).

Accordingly, the succeeding job of the *stock check* job is as follows:

- When the return code is 4 or lower  
The check result shows adequate stock and the *stock check* job ends normally. The succeeding job (*create order entry form*) will be executed next.
- When the return code is 5 or higher  
The check result shows that stock is too low and the *stock check* job ends abnormally. The recovery job (*create purchase order form*) is executed. When this job ends normally, the succeeding job of the recovery job (*create order entry form-R*) is executed.

Note that when a recovery unit is executed (because the preceding job ended abnormally), even if that recovery unit ends normally, the entire jobnet in which those units are defined is assumed to have ended abnormally. Also, if no preceding job is defined, the recovery unit is not be executed and is set to *Not executed + Ended* status.

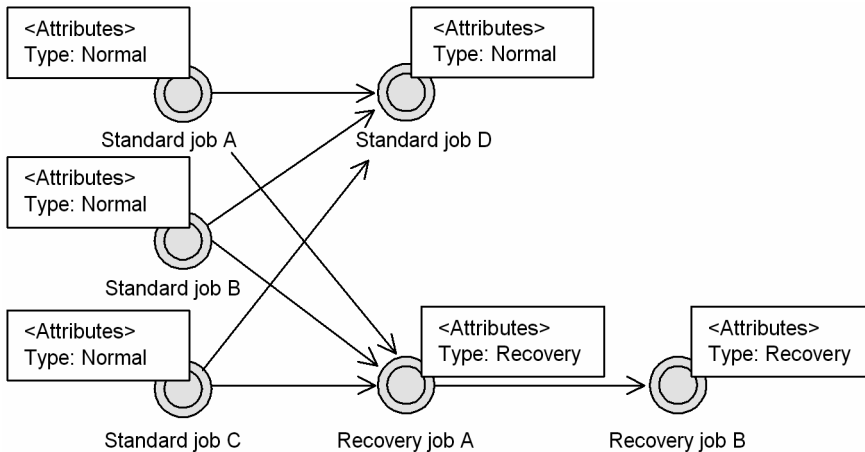
## Supplementary note

When you define an OR job as a recovery job, you must define its preceding event jobs and custom event jobs, and its succeeding job as recovery jobs.

### (2) Example of defining multiple preceding jobs for a recovery unit

You can define multiple preceding jobs for a recovery unit. In this case, the recovery unit will be executed only if all the preceding jobs end abnormally. An example of defining multiple preceding jobs for a recovery job is shown below.

Figure 2–93: Example of defining multiple preceding jobs

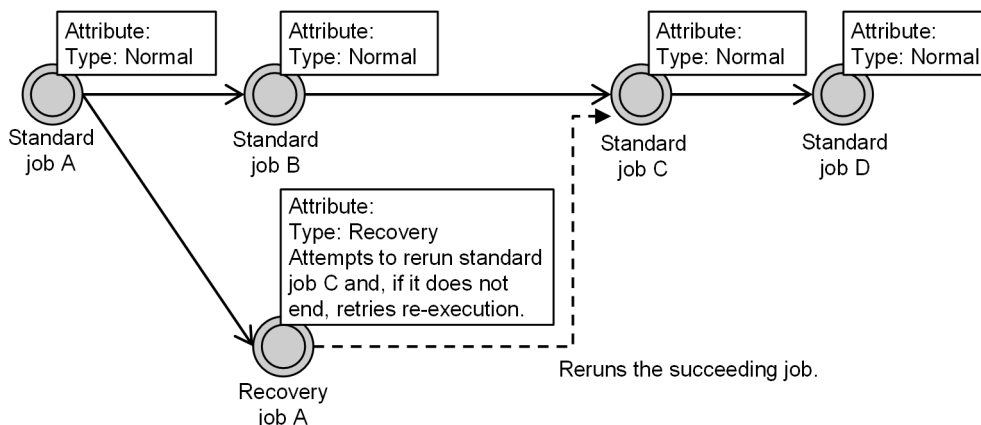


In this example, recovery job A is executed only if standard jobs A, B, and C all end abnormally. Recovery job B is executed following normal termination of recovery job A.

### (3) Example of defining a jobnet that reruns the succeeding unit of an abnormally ended unit

When the succeeding unit of an abnormally ended unit is rerun, re-execution might fail depending on the timing. You can prevent such a failure by specifying the `-rr`, `-rn`, and `-ri` options for the `ajsrrerun` command. If the command is run from a recovery unit, re-execution can be retried while the target unit is not ended. The following figure shows an example of defining a jobnet that reruns the succeeding unit of an abnormally ended unit.

Figure 2–94: Example of defining a jobnet that reruns the succeeding unit of an abnormally ended unit



In this example, when standard job A ends abnormally, recovery job A uses the `ajsrrerun` command to rerun standard job C. Recovery job A retries re-execution while standard job C has not ended.

For details about re-execution from a recovery unit, see *4.5.11 Rerunning a job or jobnet* in the manual *JP1/Automatic Job Management System 3 Overview*.

For details on the `ajsrerun` command, see *ajsrerun* in *3. Commands Used for Normal Operations* in the manual *JP1/Automatic Job Management System 3 Command Reference*.

#### Cautionary note

When the succeeding unit of an abnormally ended unit is rerun, if many retries are executed at short intervals, job execution might be adversely affected. To avoid this, note the following points when defining re-execution in a recovery unit:

- We recommend that you use the defaults for the retry count and interval for re-execution. If you change the retry count and interval, before doing so, measure and verify the processing time in the environment in which the system is actually used and set values that are not likely to affect the job execution. Note that a jobnet ends after all its subordinate units end. You must take this into account in cases such as where you measure the time required when a unit abnormally ends, and its succeeding units are placed in *Not executed + Ended* status. In this case, you must measure the length of time until the upper-level jobnet abnormally ends after all the succeeding units are placed in *Not executed + Ended* status. After measuring the time in this way, determine the proper retry count and interval for re-execution according to the system operation.
- If re-execution of multiple units needs to be retried sequentially or simultaneously, set a longer retry interval.
- For the execution agents of recovery jobs that run the `ajsrerun` command with the `-rr`, `-rn`, and `-ri` options, if necessary, set the maximum number of concurrently executable jobs to prevent concurrent execution of many instances of that command.

## 2.4.7 Controlling the execution order of a root jobnet (example of defining a jobnet that uses a jobnet connector)

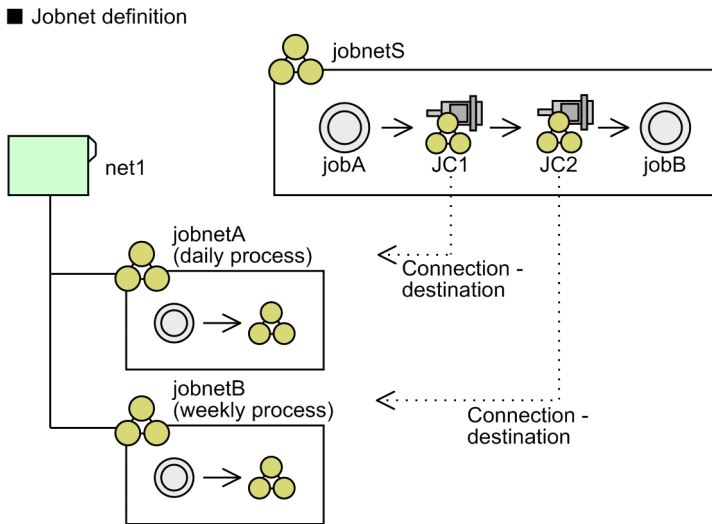
Use jobnet connectors to define a process flow with characteristics like the following:

- A root jobnet *jobnetA* with a daily schedule is executed after a daily job *jobA*.
- A job *jobB* is executed after the root jobnet *jobnetA*.
- A root jobnet *jobnetB* with a weekly schedule is executed only on Sundays after *jobnetA*.

This example assumes that the root jobnets *jobnetA* and *jobnetB* have already been defined to execute on a daily and weekly schedule, respectively.

The following figure shows an example of defining jobnet connectors.

Figure 2–95: Example of defining jobnet connectors



■ Schedule definition

|         | Mon                      | Tue                      | Wed                      | Thu                      | Fri                      | Sat                      | Sun                      |
|---------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| jobnetS | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| jobnetA | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| jobnetB |                          |                          |                          |                          |                          |                          | <input type="checkbox"/> |

First, create the root jobnet *jobnetS*, which incorporates the jobnet connectors. Then, create a job flow in which *jobA*, *jobnetA*, *jobnetB*, and *jobB* are executed sequentially. Associate the root jobnets *jobnetA* and *jobnetB* with the jobnet connectors *JC1* and *JC2*, respectively, and define a job flow in which *jobA*, *JC1*, *JC2*, and *jobB* are executed sequentially. Then, specify *jobnetA* as the connection-destination jobnet for *JC1*, and *jobnetB* as the connection-destination jobnet for *JC2*. Assign a schedule for daily execution to the root jobnet *jobnetS*.

Next, in the definitions of the connection-destination jobnets (*jobnetA* and *jobnetB*), specify the appropriate settings including the corresponding jobnet connector name and the method of execution order control. Use **Yes** as the **Exec. order control** setting for both root jobnets. As the jobnet connector name, specify *JC1* for *jobnetA* and *JC2* for *jobnetB*. When setting the execution order control method for each jobnet, specify whether the jobnet is to be executed in synchronization with the jobnet connector.

When you have finished defining the jobnets with jobnet connectors and their connection-destination jobnets, register them for execution. At this point, connections are established between generations that share an execution date according to the connection rules, and processing is executed in the defined order.

## 2.4.8 Controlling the order of unit execution in different jobnets (example of defining a jobnet that uses a wait condition)

### (1) Linking units in different jobnets

To control the execution order of units in different jobnets, use a wait condition when defining the units.

Use wait conditions when defining a process flow with characteristics like the following:

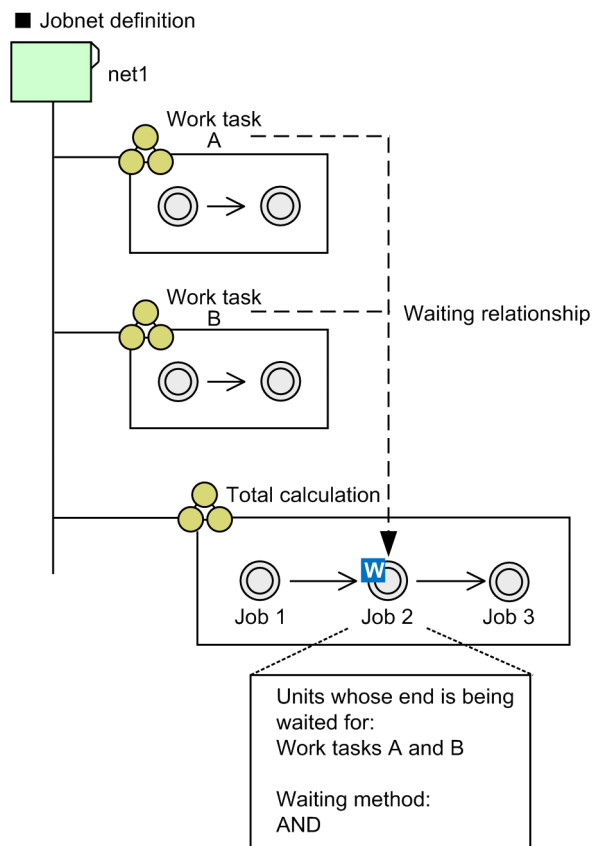
- In the *Total calculation* root jobnet, Job 1, Job 2, and Job 3 are executed sequentially.

- Job 2 uses the data output by the root jobnets *work task A* and *work task B*. For this reason, Job 2 must run after *work task A* and *work task B* have finished executing.

This example assumes that *work task A* and *work task B* have already been defined to execute once per day on weekdays.

The following figure shows an example of defining such a jobnet using wait conditions.

Figure 2–96: Example of defining a unit that uses wait conditions



■ Schedule definition

|                   | Mon | Tue | Wed | Thr | Fri | Sat | Sun |
|-------------------|-----|-----|-----|-----|-----|-----|-----|
| Work task A       | ■   | ■   | ■   | ■   | ■   |     |     |
| Work task B       | ■   | ■   | ■   | ■   | ■   |     |     |
| Total calculation | ■   | ■   | ■   | ■   | ■   |     |     |

First, create the *Total calculation* root jobnet, and define a job flow in which Job 1, Job 2, and Job 3 are executed sequentially. Because Job 2 needs to run after *work task A* and *work task B* have finished, define a wait condition for Job 2. Specify *work task A* and *work task B* as the units whose ends are being waited for, and AND as the wait method. The behavior of the unit when there are no applicable generations can be set to suit how you use the system.

Next, define a schedule for *Total calculation*. Because *Total calculation* needs to run on the same day as *work task A* and *work task B*, define its schedule to match that of *work task A* and *work task B*. In this example, because *work task A* and *work task B* are scheduled to execute on weekdays, define the same Monday to Friday schedule for *Total calculation*.

After defining the units, assigning the wait condition, and defining the schedules, register *work task A*, *work task B*, and *Total calculation* for execution. Job 2 waits for *work task A* and *work task B* to finish executing, and starts executing when both have finished.

For details on the operating methods applicable to units with wait conditions and units whose ends are being waited for after they start running, see 8.3 *Operating methods related to wait conditions* in the *JP1/Automatic Job Management System 3 Administration Guide*.

## (2) Incorporating repeated processing in a job flow (definition example using both a start condition and wait conditions)

To incorporate a unit that performs repeated processing in a job flow, define the unit using both a start condition and wait conditions.

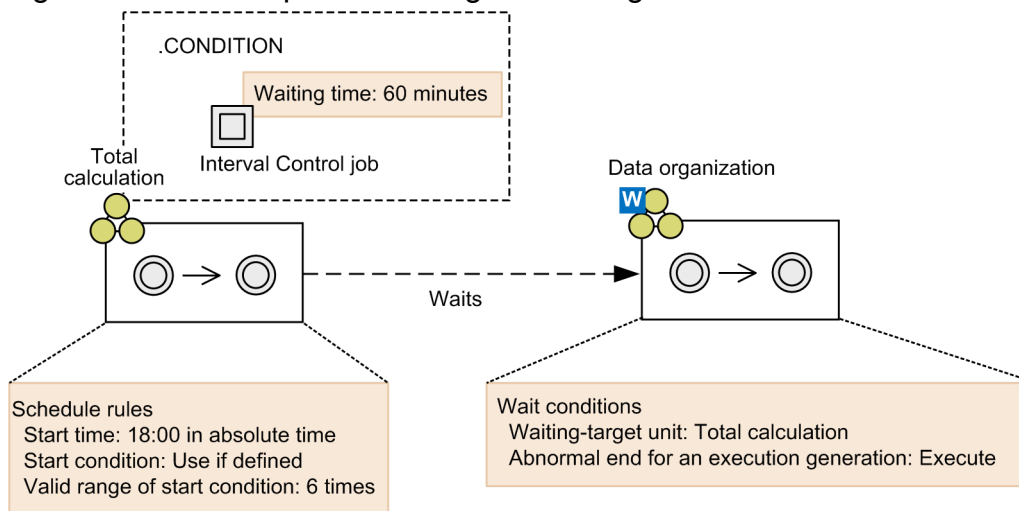
For example, you can define the following work tasks by using wait conditions:

- Define the *Total calculation* root jobnet to calculate the total amount of data. Execute the *Total calculation* root jobnet six times, once every hour starting at 18:00.
- Define the *Data organization* root jobnet to organize the totaled data. Execute the *Data organization* root jobnet regardless of whether all six executions of the *Total calculation* root jobnet have ended successfully.

The example assumes that a schedule has already been defined for each root jobnet.

The following figure shows how to use a start condition and wait conditions to define the work tasks.

Figure 2–97: Example of defining units using a start condition and wait conditions



First, define the *Total calculation* root jobnet. Define the job to be executed for *Total calculation* and then specify regular execution for the root jobnet. The detailed procedure for definition is as follows:

1. Create a start condition and define the Interval Control job.
2. In the Detailed Definition - [Interval Control] dialog box, in the **Waiting time** section, enter 60 (minutes).
3. In the Schedule Rule dialog box, define schedule rules as follows:
  - **Start time:** Select **Absolute time** and enter 18:00.
  - **Start condition:** Click **Use if defined**.
  - **Valid range of start condition:** In the **Times** box, enter 6.

Next, define the *Data organization* root jobnet. Define the job to be executed for *Data organization* and then define the wait conditions for waiting for *Total calculation* to end. In the Waiting Condition Settings dialog box, specify the wait conditions as follows:

- **Waiting-target unit:** Enter `Total` calculation.
- **Abnormal end for an execution generation:** Click **Execute**.

When you finish defining the units, schedules, and wait conditions, register *Total calculation* and *Data organization* for execution. After the root jobnets are registered for execution, *Total calculation* is executed six times on the hour starting at 18:00. When all the execution generations of *Total calculation* have finished executing, *Data organization* starts executing regardless of whether the execution generations of *Total calculation* ended abnormally.

For details about how to operate the units with wait conditions and the units whose ends are being waited for after execution has started, see 8.3 *Operating methods related to wait conditions* in the *JP1/Automatic Job Management System 3 Administration Guide*.

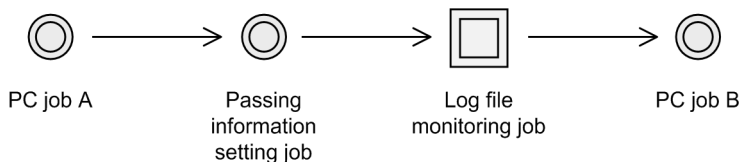
## 2.4.9 Passing information that changes dynamically to a succeeding unit (example of defining a jobnet that uses a passing information setting job)

Use a passing information setting job when defining a jobnet that executes processing based on information that changes dynamically.

A passing information setting job extracts the necessary information from a standard output file that is output by a preceding job, and assigns it to a *global macro variable* for use by a succeeding unit. A global macro variable is a type of variable set by the passing information setting job. For details about macro variables, see 2.2.6 *Considerations regarding the use of macro variables*.

The following figure shows an example of information passed using a passing information setting job.

Figure 2–98: Example of information passed using passing information setting jobs



This example assumes that the units are defined as follows:

- PC job A
  - Executable file: A batch file to output the date to the standard output
  - Standard output file: *file-name*
- Passing information setting job
  - Regular expression: `DATE= (.*)`
  - Output macro variable: `?AJS2FILEDATE?`
- Log file monitoring job
  - Log file name: `?AJS2FILEDATE?.txt`
  - Trapped data: `ERROR`
- PC job B
  - Executable file: An error-handling batch file



You must create the batch files specified as the executable files for PC job A and PC job B before you execute the jobs.

When you execute the jobnet, the individual units are executed as follows:

1. PC job A outputs the date to a standard output file.

This information appears as `DATE=20XX1010` in the standard output file.

2. The passing information setting job extracts the date portion from the standard output file output by PC job A.

A value of `20XX1010` is assigned to the macro variable `?AJS2FILEDATE?`.

3. The log file monitoring job monitors the file.

The conditions for the log file monitoring job are established if the character string `Error` is written to the file `20XX1010.txt`.

4. PC job B runs error-handling processing.

## (1) Methods for defining the job

The following describes how to define the preceding jobs, passing information setting job, and succeeding units for a job flow that uses a passing information setting job.

### (a) Preceding jobs of passing information setting job

The preceding jobs of a passing information setting job must satisfy the following criteria:

- The passing information setting job has at least one preceding job
- Only one preceding job meets all the following criteria:
  - The job is a PC job<sup>#1</sup>, Unix job<sup>#1</sup>, flexible job<sup>#2</sup>, HTTP connection job or custom job
  - The job specifies a file name or `$JP1AJS2_JPQSTDOUTTEMP$` as the name of the standard output file<sup>#3, #4</sup>
  - The job outputs its results to the standard output file

#1

Excluding queueless jobs.

#2

When the contents of the standard output file are passed from a flexible job to a passing information setting job, the maximum size of data that can be passed is 4,096 bytes. If the size of the output data exceeds 4,096 bytes, only the first 4,096 bytes are available (data beyond this is ignored). The 4,096-byte limitation is applied on both the relay agent and the destination agent. Therefore, if the relay agent and the destination agent use different character encodings, the output data must not exceed 4,096 bytes in either character encoding. Note that if a multibyte character is split at the 4,096th byte, the multibyte character is replaced by a question mark (?).

#3

Specify a file name if you want the standard output file to remain on the agent host. The specified file remains on the agent host even if transferred to the manager host.

If there is no need for the standard output file to remain on the agent host, specify

`$JP1AJS2_JPQSTDOUTTEMP$` as the file name. In this case, the standard output file is output to the agent host as a temporary file and transferred to the manager host. You cannot append data to such a file. You can check the name of the file transferred to the manager host by executing the `ajs show` command with the `-i` option and specifying the 2-byte format indicator `%s0`.

#4

For flexible jobs, you do not need to specify `$JP1AJS2_JPQSTDOUTTEMP$` as the file name.

## Cautionary notes

- If you define a passing information setting job as a dependent job of a judgment job, the preceding job of the judgment job is considered to be a preceding job of the passing information setting job. In this scenario, a standard output file name must be specified in the definition of the preceding job of the judgment job.
- If restrictions placed on file reception interrupt the transfer of the standard output file, the process will create an incomplete file. Make sure that the passing information setting job is still able to extract the necessary information from the incomplete file. Note that a newly installed JP1/AJS3 is initially set to discard the rest of the data in a file after receiving 5 MB of data. If this setting causes extraction of information to fail, adjust the value of the `LimitReceiveFileSize` environment setting parameter.
- If restrictions placed on file transmission interrupt the transfer of the standard output file, the process will create an incomplete file. Make sure that the passing information setting job is still able to extract the necessary information from the incomplete file. Note that a newly installed JP1/AJS3 is initially set to discard the rest of the data in a file after sending 3 MB of data. If this setting causes extraction of information to fail, adjust the value of the `LimitSendFileSize` environment setting parameter.

## (b) Passing information setting job

Specify the following attributes for the passing information setting job:

### Regular expressions

Specify the regular expressions used to extract information from the standard output file which is output by the preceding job.

### Output macro variable

Specify the name of the macro variable to which the job assigns the information extracted according to the regular expression.

If no lines in the standard output file that is output by the preceding job match the regular expressions, a NULL character string is assigned to the output macro variable.

The following table describes the values a passing information setting job returns when it finishes executing.

Table 2–20: Return values from passing information setting jobs

| Return value of the preceding job | Return values from passing information setting jobs | Description   |
|-----------------------------------|---|---|
| 0                                 | 0   | All regular expressions were matched  |
|                                   | 1   | At least one regular expression was not matched   |
|                                   | 20 and higher                                       | Other errors  |
| Other than 0                      | Return value of the preceding job                   | If the value of the <code>PassingInfoUsePreRc</code> environment setting parameter is 1                                       |
|                                   | 0   | If environment setting parameter <code>PassingInfoUsePreRc</code> is set to 0, and all regular expressions match              |
|                                   | 1   | If environment setting parameter <code>PassingInfoUsePreRc</code> is set to 0, At least one regular expression was no matched |
|                                   | 20 and higher                                       | If environment setting parameter <code>PassingInfoUsePreRc</code> is set to 0, Other errors                                   |

If you want to continue execution of the succeeding unit even if a regular expression does not match, perform the following procedure.

- If the value of the `PassingInfoUsePreRc` environment setting parameter is 0, set 1 as the abnormal threshold value for the passing information setting job.
- If the value of the `PassingInfoUsePreRc` environment setting parameter is 1, make settings so that the return value of the preceding job is 0, and set the abnormal threshold of the passing information setting job to 1.

If you rerun a passing information setting job that has finished executing, the information assigned to the output macro variable is set again based on the contents of the standard output file and the regular expressions that were current at the time the job was rerun.

Regular expressions specified in passing information setting jobs are treated as extended regular expressions regardless of whether the feature is enabled in JP1/Base. For details on extended regular expressions, see the *JP1/Base User's Guide*.

Take the following actions to specify an extended regular expression:

- Enclose the section to be extracted in parentheses.
- Use a period (.) to represent any character except a line break.
- A dollar sign (\$) means the string must end with this pattern to match.

#### Cautionary notes

- If you specify more than one pair of parentheses, the character string matching the leftmost pair is extracted.
- If you do not specify parentheses, a character string that matches the entire extended regular expression is extracted.
- Information is extracted from each line contained in the standard output file using the extended regular expressions. The job cannot extract information spanning multiple lines.
- If there is an error in the syntax of an extended regular expression (for example, there is no closing parenthesis to match an opening one), the passing information setting job terminates abnormally.
- A line in the standard output file cannot exceed 1,024 bytes in length. If a passing information setting job attempts to extract information from a line that is longer than 1,024 bytes, the job terminates abnormally.
- If multiple character strings match the extended regular expression, the first match encountered from the beginning of the file is extracted.

The following table describes examples of the information extracted by extended regular expressions.

Table 2–21: Examples of extended regular expressions

| Target character string | Pattern                       | Extended regular expression | Extracted information |
|-------------------------|-------------------------------|-----------------------------|-----------------------|
| abcdefghi               | One pair of parentheses       | abc (...) ghi               | def                   |
|                         | Multiple pairs of parentheses | abc (.) (.) (.) ghi         | d                     |
|                         | No parentheses                | ...\$                       | ghi                   |

### (c) Succeeding jobs of passing information setting units

Specify the name of the variable specified as the output macro variable for the passing information setting job.

## (2) Valid range of passing information

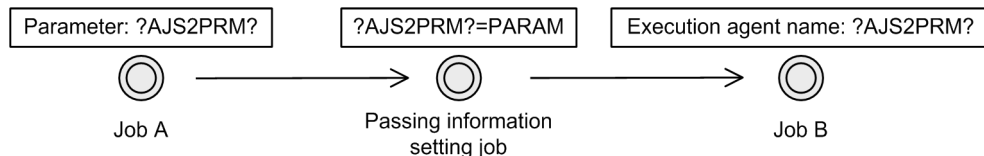
The passing information is valid within the scope of the root jobnet. However, the passing information extracted by the passing information setting job is assigned to the macro variable only after the passing information setting job has

executed. You cannot reference the value assigned to the macro variable before the job executes. If you rerun a passing information setting job, the value set in the macro variable by the job is not cleared.

The figure below shows the value of the macro variable after each job is executed, assuming the following conditions:

- The macro variable `?AJS2PRM?` is defined in Job A and Job B
- The passing information setting job assigns `PARAM` to `?AJS2PRM?`

Figure 2–99: Example of passing information setting job



The following table describes the executed job and the value of the macro variable at execution, for the example in the figure above.

Table 2–22: Executed job and value of macro variable

| Executed job                    | Value of macro variable |
|---------------------------------|-------------------------|
| Job A                           | <code>?AJS2PRM?</code>  |
| Passing information setting job | --                      |
| Job B                           | <code>PARAM</code>      |
| Job A (rerun)                   | <code>PARAM</code>      |

### (3) Referencing passed information

Whether you can reference the result of the passing information setting job depends on the definition of its upper-level jobnet.

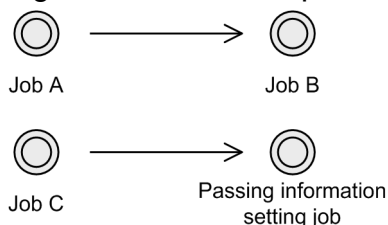
For details on checking what information was passed to a macro variable, see [2.2.6\(2\) Checking macro variables](#).

The following describes the context in which passed information can be referenced.

#### (a) No relation lines

For a job that has no relation line drawn to the passing information setting job, you can check the passed information provided that the passing information setting job was executed first.

Figure 2–100: Example of passed information for jobs with no relation line



The following table describes whether passed information can be referenced for the example in this figure.

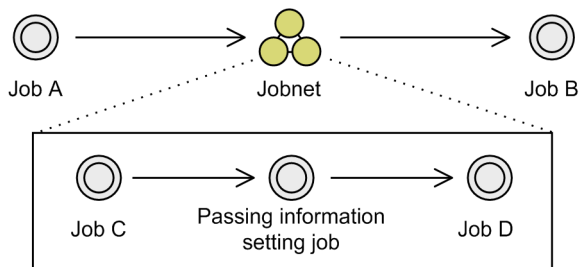
Table 2–23: Whether passed information can be referenced for jobs with no relation lines

| Job   | Whether passed information can be referenced  |
|-------|---|
| Job A | Can be referenced if the passing information setting job was executed before Job A. The same applies if the job is rerun.<br>Passed information cannot be referenced if the passing information setting job was executed after Job A. |
| Job B | Can be referenced if the passing information setting job was executed before Job B. The same applies if the job is rerun.<br>Passed information cannot be referenced if the passing information setting job was executed after Job B. |
| Job C | Cannot be referenced at execution of the root jobnet.<br>Can be referenced if Job C is rerun.   |

### (b) Nested jobnet

The jobs in a root jobnet can reference the information passed by a passing information setting job in a nested jobnet.

Figure 2–101: Example of passed information for a nested jobnet



The following table describes whether passed information can be referenced for the example in this figure.

Table 2–24: Whether passed information can be referenced for nested jobnets

| Job   | Whether passed information can be referenced  |
|-------|---|
| Job A | Cannot be referenced at execution of the root jobnet.<br>Can be referenced if Job A is rerun. |
| Job B | Can be referenced.  |
| Job C | Cannot be referenced at execution of the root jobnet.<br>Can be referenced if Job C is rerun. |
| Job D | Can be referenced.  |

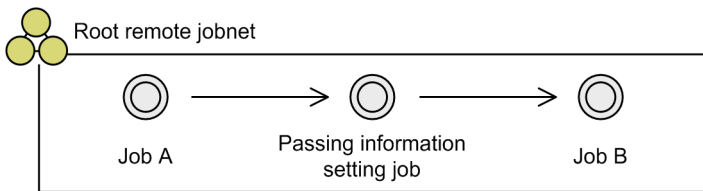
### (c) Remote jobnet

The following describes whether passed information can be referenced when a passing information setting job is defined in a jobnet that includes a remote jobnet.

#### ■ Root remote jobnet

When a passing information setting job is defined in a root remote jobnet, the passed information can be referenced by units in the root remote jobnet.

Figure 2–102: Example of passed information for a root remote jobnet



The following table describes whether passed information can be referenced for the example in this figure.

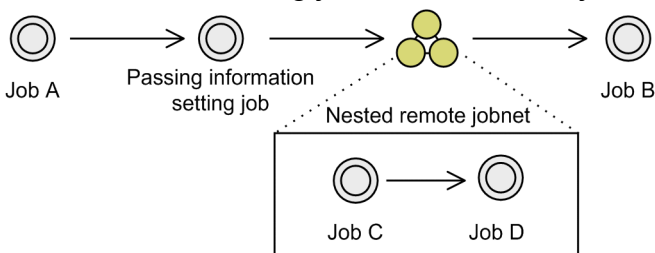
Table 2–25: Whether passed information can be referenced for root remote jobnets

| Job   | Whether passed information can be referenced  |
|-------|---|
| Job A | Cannot be referenced at execution of the root jobnet.<br>Can be referenced if Job A is rerun. |
| Job B | Can be referenced.  |

■ **Nested remote jobnet (with passing information setting job defined in root jobnet)**

When a passing information setting job is defined in a root jobnet, only the jobs in that root jobnet can reference the passed information. Jobs in the nested remote jobnet cannot reference the information.

Figure 2–103: Example of passing information for a nested remote jobnet (with passing information setting job defined in root jobnet)



The following table describes whether passed information can be referenced for the example in this figure.

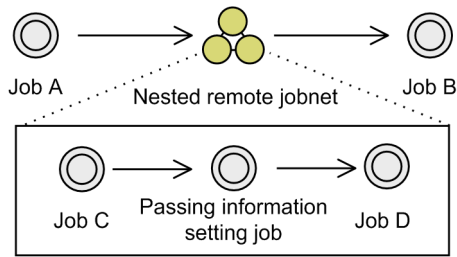
Table 2–26: Whether passed information can be referenced for nested remote jobnets (with passing information setting job defined in root jobnet)

| Job   | Whether passed information can be referenced  |
|-------|---|
| Job A | Cannot be referenced at execution of the root jobnet.<br>Can be referenced if Job A is rerun. |
| Job B | Can be referenced.  |
| Job C | Cannot be referenced.   |
| Job D | Cannot be referenced.   |

■ **Nested remote jobnet (with passing information setting job defined in nested remote jobnet)**

When a passing information setting job is defined in a nested remote jobnet, only the jobs in that nested remote jobnet can reference the passed information. Jobs in the root jobnet cannot reference the information.

Figure 2–104: Example of passing information for a nested remote jobnet (with passing information setting job defined in nested remote jobnet)



The following table describes whether passed information can be referenced for the example in this figure.

Table 2–27: Whether passed information can be referenced for nested remote jobnets (with passing information setting job defined in nested remote jobnet)

| Job   | Whether passed information can be referenced  |
|-------|---|
| Job A | Cannot be referenced.   |
| Job B | Cannot be referenced.   |
| Job C | Cannot be referenced at execution of the root jobnet.<br>Can be referenced if Job C is rerun. If the nested remote jobnet is rerun, passed information cannot be referenced afterwards. |
| Job D | Can be referenced.  |

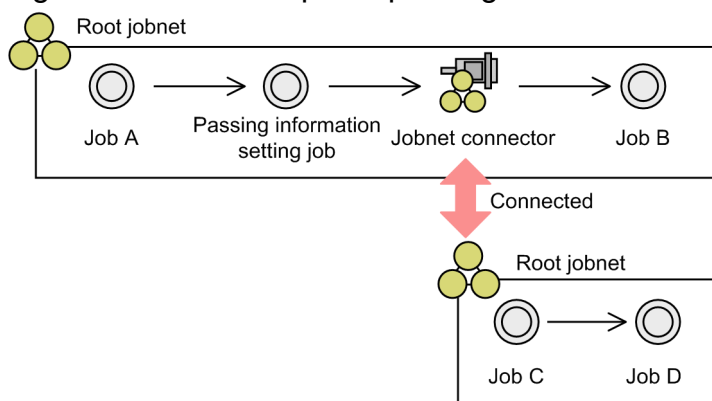
#### (d) Jobnet connector

The information passed by a passing information setting job defined in a jobnet with a jobnet connector or its connection-destination jobnet can only be referenced from the jobnet that contains the passing information setting job.

##### ■ Passing information setting job in jobnet with jobnet connector

Only the jobnet where the jobnet connector is defined can reference the passed information.

Figure 2–105: Example of passing information setting job defined in jobnet with jobnet connector



The following table describes whether passed information can be referenced for the example in this figure.

Table 2–28: Whether passed information can be referenced for jobnet with jobnet connector defined

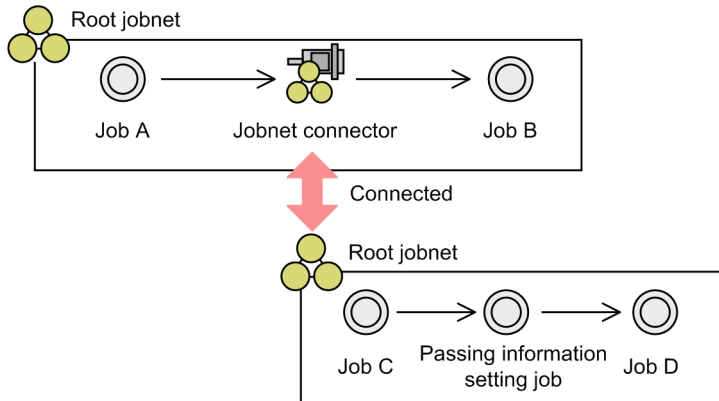
| Job   | Whether passed information can be referenced          |
|-------|---|
| Job A | Cannot be referenced at execution of the root jobnet. |

| Job   | Whether passed information can be referenced |
|-------|--|
| Job A | Can be referenced if Job A is rerun.         |
| Job B | Can be referenced.                           |
| Job C | Cannot be referenced.                        |
| Job D | Cannot be referenced.                        |

### ■ Passing information setting job in connection-destination jobnet

Only the connection-destination jobnet can reference the passed information.

Figure 2–106: Example of passing information setting job in connection-destination jobnet of jobnet connector



The following table describes whether passed information can be referenced for the example in this figure.

Table 2–29: Whether passed information can be referenced for passing information setting job in connection-destination jobnet of jobnet connector

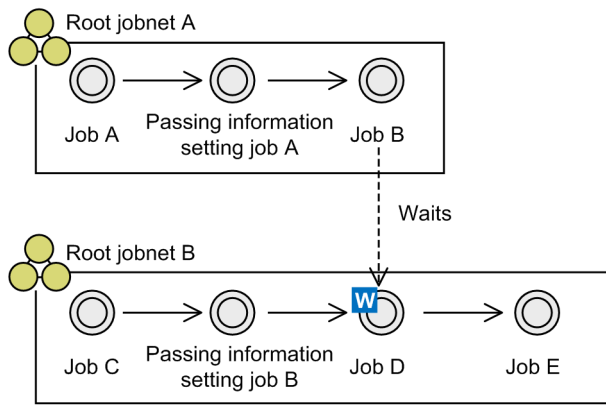
| Job   | Whether passed information can be referenced  |
|-------|---|
| Job A | Cannot be referenced.   |
| Job B | Cannot be referenced.   |
| Job C | Cannot be referenced at execution of the root jobnet.<br>Can be referenced if Job C is rerun. |
| Job D | Can be referenced.  |

### (e) Wait conditions

When a unit with wait conditions and its units whose ends are being waited for are in different root jobnets, passed information can only be referenced within the scope of the respective root jobnets.



Figure 2–107: Example of passed information using wait conditions



Legend:

----> : Flow of wait processing

The following tables describe whether passed information can be referenced for the example in this figure.

Table 2–30: Whether passed information can be referenced (passing information setting Job A)

| Job   | Whether passed information can be referenced  |
|-------|---|
| Job A | Cannot be referenced at execution of Root Jobnet A.<br>Can be referenced if Job A is rerun. |
| Job B | Can be referenced.  |
| Job C | Cannot be referenced.   |
| Job D | Cannot be referenced.   |
| Job E | Cannot be referenced.   |

Table 2–31: Whether passed information can be referenced (passing information setting Job B)

| Job   | Whether passed information can be referenced  |
|-------|---|
| Job A | Cannot be referenced.   |
| Job B | Cannot be referenced.   |
| Job C | Cannot be referenced at execution of Root Jobnet B.<br>Can be referenced if Job C is rerun. |
| Job D | Can be referenced.  |
| Job E | Can be referenced.  |

## (4) Cautionary notes

Note the following when using passing information setting jobs:

- If you copy a passing information setting job to an environment running a version of JP1/AJS3 - Manager earlier than 09-50, execution of the job will result in an error.
- You cannot specify an execution agent for a passing information setting job. The job is executed by the default execution agent @SYSTEM of the manager host where the job is registered for execution. A passing information setting job in a remote jobnet is executed by the default execution agent on the transfer destination host.

- If more than one passing information setting job defined under a given root jobnet uses the same macro variable name, the valid passing information is that of the passing information setting job executed last. For cautionary notes regarding the use of macro variables, see [2.2.6\(5\) Cautionary notes](#).
- To use passing information setting jobs, enable the setting for ensuring that data generated by jobs is output to result files. For details, see [6.2.26 Setting for ensuring that data generated by jobs is output to result files](#) in the *JP1/Automatic Job Management System 3 Configuration Guide*.

## 2.4.10 Executing jobs in a cloud environment (example of defining a jobnet that uses flexible jobs)

If you define a jobnet to execute jobs in a cloud environment in which host names and IP addresses are not fixed, use flexible jobs. If you use flexible jobs, processing can be executed by an agent host that is not directly managed by the manager host. Therefore, flexible jobs are useful when jobs need to be executed in an auto-scaling environment in situations such as the following:

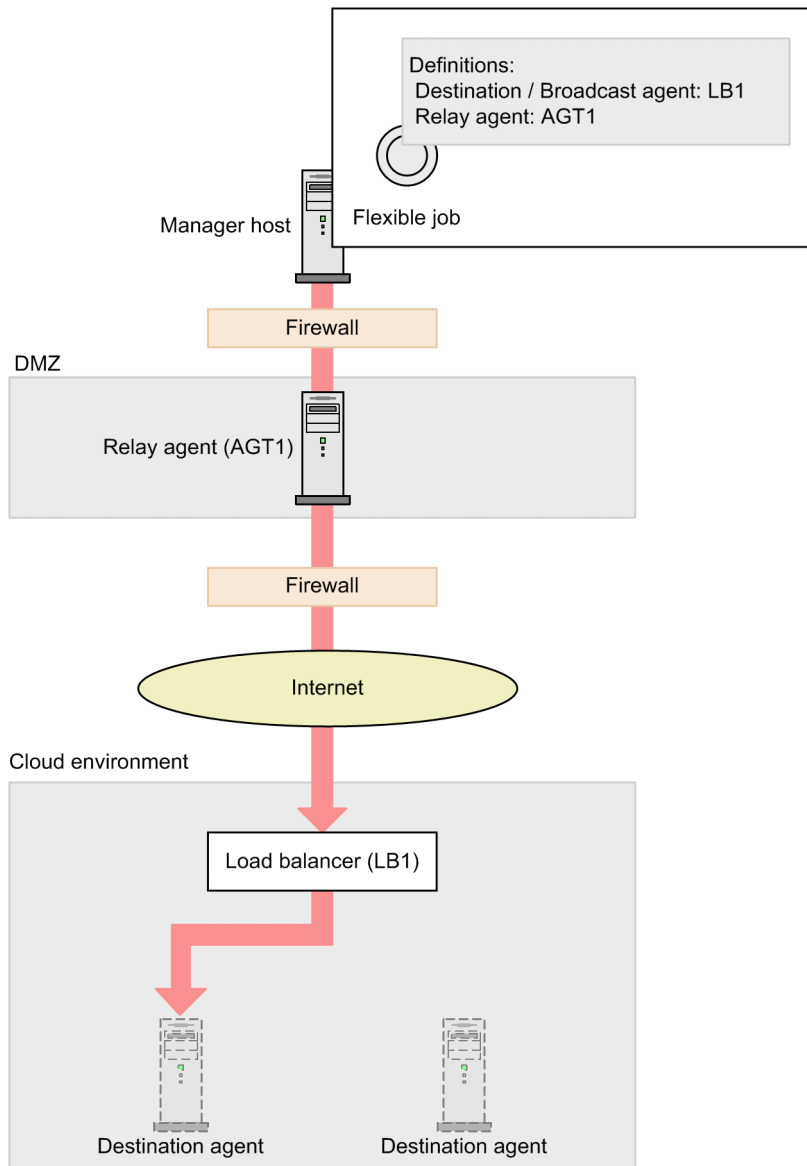
- A load balancer is used to distribute a job to execution agents.
- A job is executed on all execution agents that exist in an auto-scaling environment.

We also recommend that you use a flexible job if you execute any of the following:

- A job to be executed in an environment in which host names and IP addresses are not fixed  
If host names and IP addresses are fixed, we recommend that you use PC jobs and Unix jobs rather than flexible jobs.
- A job that does not require analysis, and that requires only re-execution if execution of the job does not succeed  
Scale-in and scale-out actions in an auto-scaling environment in a cloud are managed on the side of the cloud. Thus, the time before startup of the destination agent by scaling-out in an auto-scaling environment finishes and the deletion of destination agents by scaling-in cannot be detected by JP1/AJS3. At that timing, if a flexible job is executed, it might fail, and it is difficult to analyze the cause of the error. Therefore, in an auto-scaling environment, we recommend that you execute a jobnet that you have only to re-execute without analyzing the cause if the jobnet fails.

An example of defining a flexible job that uses a load balancer is shown in the following figure. For details about the system components and other considerations required to execute flexible jobs, see [2.9 Executing jobs in a cloud environment](#) in the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*. For an example of defining a jobnet that executes a flexible job on multiple execution agents simultaneously, see [2.4.11 Executing a job on multiple execution agents simultaneously \(example of defining a jobnet that uses flexible jobs\)](#).

Figure 2–108: Example definition of a flexible job that uses a load balancer



When the jobnet defined in the above figure is executed, the following operations take place:

- A job execution request is transferred via relay agent (AGT1).
- The requested job is distributed by load balancer (LB1) to destination agents where the job is executed.

For notes on using flexible jobs, see [7.8 Notes on using flexible jobs](#).

For details about the settings necessary to execute flexible jobs, see [21.3 Setup to execute flexible jobs](#) in the *JP1/Automatic Job Management System 3 Configuration Guide*.

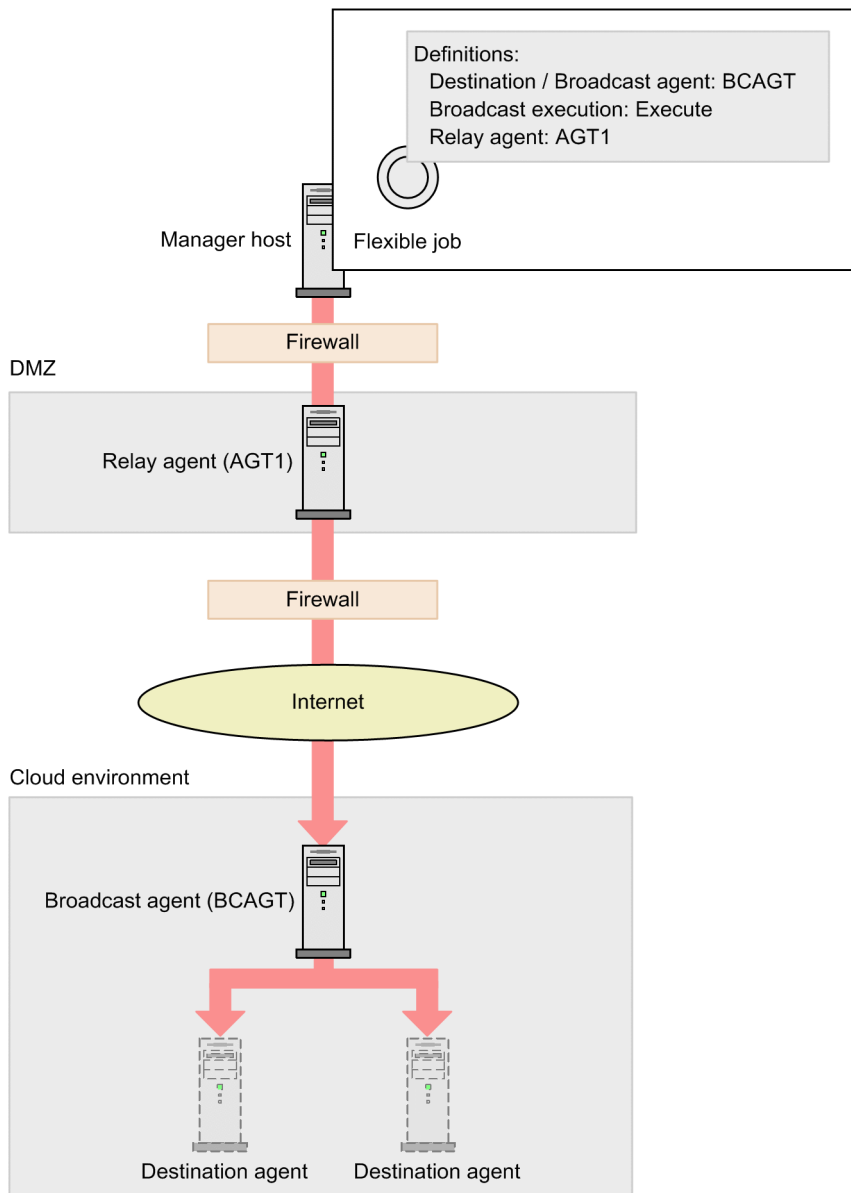
## 2.4.11 Executing a job on multiple execution agents simultaneously (example of defining a jobnet that uses flexible jobs)

In JP1, the act of simultaneously executing a single job on multiple execution agents is called *broadcast execution*. To use broadcast execution, define a flexible job in a jobnet.

Broadcast execution of a flexible job allows the user to simultaneously execute a job on multiple execution agents by defining only one flexible job. If the number of execution agents is increased or decreased, the agent that simultaneously distributes a job (the *broadcast agent*) automatically expands or shrinks the distribution scope. Therefore, you do not need to add or delete jobs.

The following figure shows an example of defining a jobnet that uses broadcast execution of a flexible job. For details about the system components and other considerations required to execute flexible jobs, see 2.9 *Executing jobs in a cloud environment* in the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*, and see 2.10 *Considerations for executing a job by broadcast execution* in the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*.

Figure 2–109: Example definition of a jobnet that uses broadcast execution of a flexible job



When the jobnet defined in the above figure is executed, the following operations take place:

- A job execution request is transferred via relay agent (AGT1).
- The broadcast agent (BCAGT) distributes the job.
- The job is executed on multiple destination agents in a cloud environment.

When you use broadcast execution of a flexible job, you do not need to add or delete jobs even if the number of destination agents increases or decreases as a result of auto-scaling. In addition, you can set up a single destination agent and then set up auto-scaling to replicate the other destination agents based on a machine image of the destination agent you set up. If you use auto-scaling in this way, you do not need to set up each destination agent individually.

#### Sync mode and async mode

Broadcast execution of a flexible job can be executed in sync mode or async mode. Each successive job will be executed at a different time depending on the mode. The following describes sync mode and async mode.

##### Sync mode

After all destination agents notify the broadcast agent that execution of a job finished, the next job starts.

##### Async mode

When the broadcast agent finishes sending a job execution request to all destination agents, the next job starts.

For notes about using broadcast execution, see [7.8 Notes on using flexible jobs](#).

For details about the settings necessary to perform broadcast execution, see [21.3.2 Setup procedure to execute a job by broadcast execution](#) in the *JP1/Automatic Job Management System 3 Configuration Guide*.

## 2.4.12 Linking with a business system on the web (example of defining a jobnet that uses HTTP connection jobs)

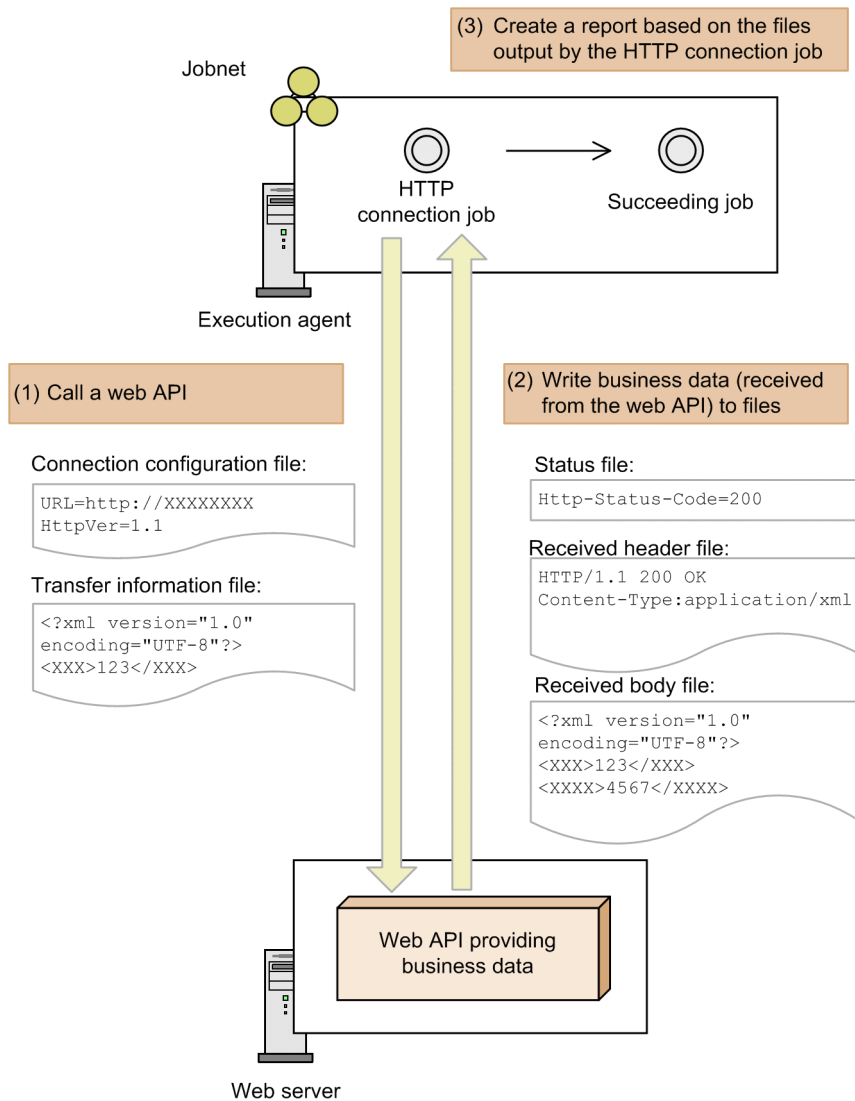
To define a jobnet that links a JP1/AJS3-based business system with a business system on the web, use HTTP connection jobs.

HTTP connection jobs can be used to send requests and receive responses via the HTTP protocol. HTTP connection jobs also allow JP1/AJS3 to call a web API (such as the REST API) that is provided in a cloud environment or on a web server.

### (1) Example of defining a jobnet that uses an HTTP connection job

The following figure shows an example of using an HTTP connection job to define a jobnet that obtains business data through a web API provided on the web, and creates a report.

Figure 2–110: Example of defining a jobnet that uses HTTP connection jobs



## (2) Definition of an HTTP connection job

This subsection describes the definition items specific to HTTP connection jobs.

To send HTTP requests by using HTTP connection jobs, define the files that specify the request type and request information. If extended mode is used when the versions of JP1/AJS3 - Manager and JP1/AJS3 - Agent are 12-50 or later, you can send both the URL parameter and message body simultaneously when sending an HTTP request.

The following table describes the definition items that are used to send HTTP requests.

Table 2–32: Definition items used to send HTTP requests

| No. | Definition item | Description  |
|-----|-----------------|--|
| 1   | Request type    | Specify the type of request to be sent to the web API. You can specify one of the following request types: <ul style="list-style-type: none"> <li>• GET</li> <li>• POST (add)</li> <li>• PUT (update)</li> <li>• DELETE</li> </ul> |

| No. | Definition item  | Description  |
|-----|--|--|
| 2   | Connection configuration file name                               | Specify the name of the connection configuration file that contains the HTTP connection information for calling a web API.                     |
| 3   | Transmission information file name <sup>#1</sup>                 | Specify the name of the transmission information file that contains the HTTP request information for calling a web API.                        |
| 4   | Transmission information file name (URL parameter) <sup>#2</sup> | Specify the name of the transmission information file that contains the URL parameter of the HTTP request information for calling the Web API. |
| 5   | Transmission information file name (Message body) <sup>#2</sup>  | Specify the name of the transmission information file that contains the message body of the HTTP request information for calling the Web API.  |

#1

This is a file to be specified if the versions of JP1/AJS3 - Manager and JP1/AJS3 - Agent are 12-10 or earlier or if extended mode is not used.

#2

This is a file to be specified if the versions of JP1/AJS3 - Manager and JP1/AJS3 - Agent are 12-50 or later when extended mode is used.

To receive HTTP responses by using HTTP connection jobs, define the name of the file to which received data will be written, and how that received data will be stored.

The following table describes the definition items that are used to receive HTTP responses.

**Table 2–33: Definition items used to receive HTTP responses**

| No. | Definition item            | Description   |
|-----|----------------------------|---|
| 1   | Status file name           | Specify the name of the status file to which the HTTP status code of receive data obtained from the web API will be written.  |
| 2   | How to store received data | Specify whether the header and body of the receive data obtained from the web API are to be output to the same file or different files.<br>If you choose to output the header and body to the same file, both the header and body of receive data are stored in the received header file. |
| 3   | Received header file name  | Specify the name of the received header file to which the header of the receive data obtained from the web API will be written.   |
| 4   | Received body file name    | Specify the name of the received body file to which the body of the receive data obtained from the web API will be written.   |

When you receive data in chunk format by using an HTTP response, you can store decoded data to a file. To decode data in chunk format, specify `y` for the `TRANSFER_DECODING` environment setting parameter. For details about the `TRANSFER_DECODING` environment setting parameter, see *20.14.2(1) TRANSFER\_DECODING* in the *JP1/Automatic Job Management System 3 Configuration Guide*.

For details about the files that are used for HTTP connection jobs, see *C. Files Used for HTTP Connection Jobs*.

Also, you can specify job return code 0 to 9 for each received HTTP status code value. If you do not specify a return code, the default return code is applied. You can specify multiple HTTP status codes for one job return code. However, you cannot specify one HTTP status code for multiple job return codes. For details about the return code of a job, see *7.11.3 Checking the return code of a job*.

### (3) HTTP headers set by HTTP connection jobs

All HTTP connection jobs automatically set HTTP headers according to the request type. The following table describes the HTTP headers that can be set automatically.

Table 2–34: HTTP headers set by HTTP connection jobs

| No. | Request type  | HTTP header    | Description  |
|-----|---|----------------|--|
| 1   | GET   | User-Agent     | Fixed to JP1/AJS3 11-00 HTTP Connection Job  |
| 2   |   | Host           | The host name and port number of the connection destination are set according to the information specified in the connection configuration file.   |
| 3   |   | Accept         | Fixed to */*   |
| 4   | <ul style="list-style-type: none"> <li>• POST</li> <li>• PUT</li> <li>• DELETE</li> </ul> | User-Agent     | Fixed to JP1/AJS3 11-00 HTTP Connection Job  |
| 5   |   | Host           | The host name and port number of the connection destination are set according to the information specified in the connection configuration file.   |
| 6   |   | Accept         | Fixed to */*   |
| 7   |   | Content-Length | The data size of the HTTP body is calculated and set. If the transmission information file is not specified or is empty, this header is not set.   |
| 8   |   | Expect         | 100-continue is set if both of the following conditions exist: <ul style="list-style-type: none"> <li>• The HTTP version specified in the connection configuration file is 1.1.</li> <li>• A transmission information file is specified and request information is written in the file.</li> </ul> |

In addition to the HTTP headers that are automatically set by HTTP connection jobs, you can set additional headers by specifying `Header=additional-header` entries in the connection configuration file. For details, see [C.1 Connection configuration file](#).

### (4) Security measures for HTTP connection jobs

This subsection describes HTTPS communication and server authentication used for HTTP connection jobs.

#### (a) HTTPS communication

HTTP connection jobs support HTTPS communication. The following table describes the details of the HTTPS support of HTTP connection jobs.

Table 2–35: HTTPS support of HTTP connection jobs

| No. | Definition             | Description   |
|-----|------------------------|---|
| 1   | Communication protocol | TLS v1.0, TLS v1.1, TLS v1.2, and TLS v1.3 are supported.   |
| 2   | Server authentication  | Supported.  |
| 3   | Client authentication  | Not supported.  |
| 4   | Host name verification | Supported.<br>Whether the following names match is checked: Host name specified for the URL entry in the connection configuration file, and the CN (Common Name) or SAN (subject Alt Name) included in the server certificate returned by the server. |



| No. | Definition                               | Description   |
|-----|--|---|
| 5   | Checking the certificate revocation list | Supported.<br>Note that to enable this checking, the file name of the certificate revocation list must be specified in the connection configuration file. |

For details about the connection configuration file, see *C.1 Connection configuration file*.

## (b) Server authentication

Basic Authentication can be used as a server authentication method.

Other authentication methods can also be used by setting additional headers in the connection configuration file.

For details about the connection configuration file, see *C.1 Connection configuration file*.

## (5) Information output by HTTP connection jobs

When an HTTP connection job terminates, it outputs the following information to the standard output file:

```
Http-Status-Code=HTTP-status-value#1
Http-Status-File=Status file name
Http-Response-Header=Received header file name
Http-Response-Body=Received body file name#2
```

#1

The type of information output here differs depending on the HTTP status code. If the HTTP status code is 1 or a larger value, the HTTP status is output here. If the HTTP status code is 0 or a smaller value, the maintenance information set by JP1/AJS3 is output.

#2

If you specify that the header and body of the receive data are to be output to the same file as HTTP connection jobs' definition item *How to store received data*, only the entry name `Http-Response-Body=` is output. The value (*received-body-file-name*) is not output.

### Supplementary notes

- Any information items that have been output to the standard output file can be passed to a succeeding job by using a passing information setting job.
- Each HTTP connection job always outputs information to the standard output file in four-line format regardless of the return code. If an HTTP connection job terminates abnormally, the values of entries (values on the right side of the equal sign (=)) might not be output.

## (6) Cautionary notes

- Only Windows or Linux can be used as the OSs of the manager and agent hosts that can be specified as execution hosts in HTTP connection jobs.
- If an HTTP connection job is abnormally terminated (due to a timeout), or forcibly terminated, the job itself terminates but the processes executed via a web API are not terminated. If you also cancel execution of processes that are executed via a web API, do so according to the specifications of that API.
- If you include security information (such as a password) in the following files, which are used for HTTP connection jobs, you do so on your own responsibility:
  - Connection configuration file

- Transfer information file
- Received header file
- Received body file

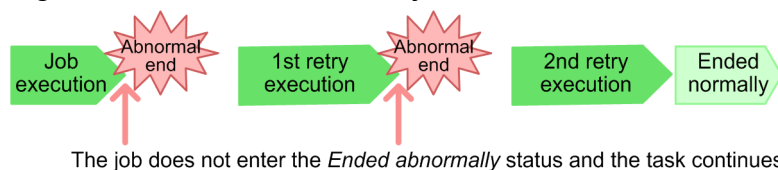
### 2.4.13 Automatic retry for abnormally ending jobs

If an executable file defined for a job ends abnormally, retrying the job might correct temporary errors. You can define automatic retry for those jobs capable of being retried in the event they end abnormally. In this way, you can continue tasks even if a temporary error has occurred in an executable file.

*Automatic retry* means automatically retrying a job if an executable file specified for a job ends abnormally. The execution of a job by an automatic retry is called a *retry execution*.

The following figure shows the behavior of a job when an automatic retry is performed.

Figure 2–111: Behavior of a job when an automatic retry is performed



When an automatic retry is performed, the job does not enter the *Ended abnormally* status even when an executable file defined for the job ends abnormally. Instead, after a preset interval, the job is automatically retried.

## (1) Overview of automatic retries

The following provides an overview of automatic retries.

### (a) Conditions triggering automatic retries

If an error occurs in an executable file of a job that satisfies the following conditions, the job is automatically retried without entering the *Ended abnormally* status.

- In the **End judgment** section, in the **Rule** box, **Judgment by threshold** is selected.
- In the **Retry on abnormal end** section, **Yes** is chosen.

**Retry on abnormal end** is available for the following jobs:

- Unix jobs
- PC jobs
- QUEUE jobs
- Flexible jobs
- HTTP connection jobs
- Standard custom jobs
- Custom PC jobs
- Custom Unix jobs

### Cautionary note

Automatic retries can be configured when the version of JP1/AJS3 - View and JP1/AJS3 - Manager is 10-00 or later. However, even when the version of JP1/AJS3 - Manager is 10-00 or later, automatic retries are not available if the database uses a compatible ISAM configuration.

## (b) Settings related to how an automatic retry is executed

The settings related to executing an automatic retry are called *retry settings*.

The following table describes the retry settings.

Table 2–36: Settings related to executing an automatic retry

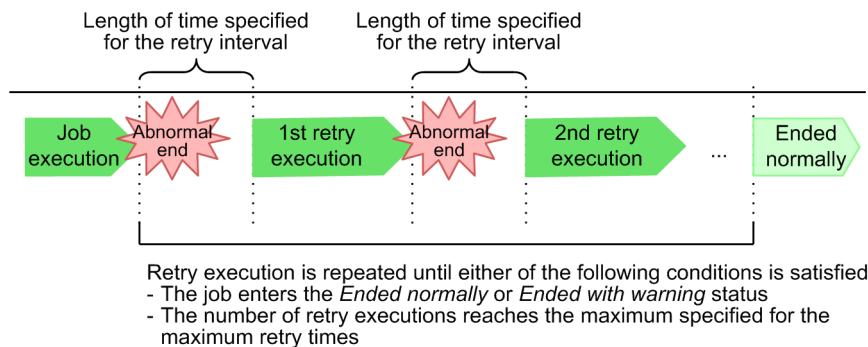
| No. | Item                         | Description  |
|-----|------------------------------|--|
| 1   | <b>Retry on abnormal end</b> | Indicates whether to perform an automatic retry if executable files specified for jobs cause an error.                   |
| 2   | <b>Return code</b>           | Indicates a range of return codes for which retry execution is to be performed.  |
| 3   | <b>Maximum retry times</b>   | Indicates the maximum number of times a retry is to be executed.   |
| 4   | <b>Retry interval</b>        | Indicates an interval between the time an executable file for a job causes an error and the time retry execution begins. |

### Cautionary note

For **Return code**, specify the minimum range that is necessary. If you set an unnecessarily wide range of return codes for automatic retries, retries are executed for return codes that are impossible to correct by performing retry execution. As a result, the number of job executions increases and job execution performance is likely to be affected.

The following figure shows the behavior of a job that has ended abnormally when retry settings are specified.

Figure 2–112: Behavior of a job ending abnormally when retry settings are specified



If an executable file specified for a job with retry settings ends abnormally, a retry is executed after the length of time specified in **Retry interval** elapses. Retry execution is repeated until the job ends normally or with a warning, or the retry execution count has reached the maximum specified in **Maximum retry times**.

You can check the retry settings in JP1/AJS3 - View windows or by using a command. The following table describes the JP1/AJS3 - View windows and command you can use and the retry settings you are able to check.

Table 2–37: JP1/AJS3 - View windows and command you can use to check the retry settings

| No. | JP1/AJS3 - View window and command for checking retry settings | Retry settings that can be checked  |
|-----|--|---|
| 1   | Jobnet Editor window   | <b>Retry on abnormal end</b>  |
| 2   | Search window  | <ul style="list-style-type: none"> <li>• <b>Maximum retry times</b></li> <li>• <b>Retry interval</b></li> </ul> |
| 3   | <code>ajsprint</code> command                                  | All retry settings  |

### (c) Information related to the execution status of an automatic retry

The following information related to the execution status of an automatic retry is called *retry information*:

- **Retry status**
- **Retry execution times**
- **Retry registration time**
- **Retry start time**

Retry information details are described below.

#### Retry status

The retry status indicates the progress of an automatic retry processing when it is being executed.

The following table describes the types of retry statuses.

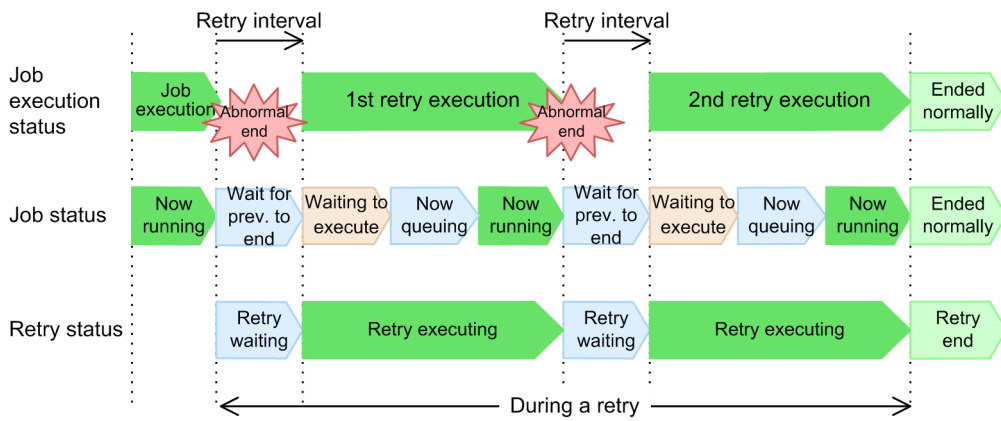
Table 2–38: Types of retry statuses

| No. | Type of retry status | Description  | Corresponding job status  |
|-----|----------------------|--|---|
| 1   | Retry waiting        | Indicates that an executable file for a job resulted in an error and the job is waiting for the length of time specified in <b>Retry interval</b> to elapse. | <ul style="list-style-type: none"> <li>• Wait for prev. to end</li> <li>• Being held</li> </ul>   |
| 2   | Retry executing      | The job is in the <i>Waiting to execute</i> , <i>Now queuing</i> , or <i>Now running</i> status due to an automatic retry processing.                        | <ul style="list-style-type: none"> <li>• Waiting to execute</li> <li>• Now queuing</li> <li>• Now running</li> </ul>  |
| 3   | Retry end            | Automatic retry processing has ended.  | <ul style="list-style-type: none"> <li>• Ended normally</li> <li>• Ended with warning</li> <li>• Ended abnormally</li> <li>• Failed to start</li> <li>• Unknown end status</li> <li>• Bypassed</li> </ul> <p>End status including the above</p> |

The *Retry waiting* and *Retry executing* statuses are generically referred to as *during a retry*.

The following figure shows an example of the status transitions when an automatic retry is performed.

Figure 2–113: Status transitions when an automatic retry is performed



If an executable file specified for a job with retry settings ends abnormally, the retry status is *Retry waiting* for the length of time specified in **Retry interval**. The job at this time is in the *Wait for prev. to end* status, not the *Ended abnormally* status. When the length of time specified in **Retry interval** elapses, the retry status transitions to *Retry executing*.

If the executable file specified for the job ends normally or ends with a warning before the number of retry executions can reach the number specified in **Maximum retry times**, the job enters the *Ended normally* or *Ended with warning* status.

If the executable file specified for the job does not end normally or ends with a warning when the number specified in **Maximum retry times** has been reached, the job enters the *Ended abnormally* status.

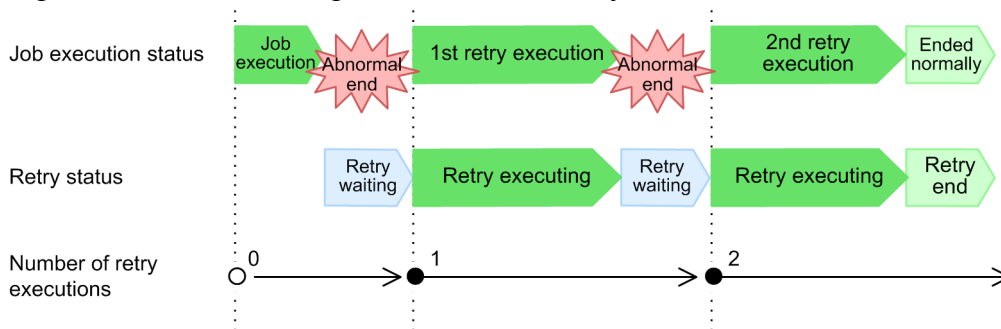
The retry status transitions to *Retry end* when the job enters the *Ended normally*, *Ended with warning*, or *Ended abnormally* status.

#### Number of retry executions

The number of retry executions indicates the number of retry executions.

The following figure shows how the number of retry executions is counted.

Figure 2–114: Counting the number of retry executions



Legend:

○ : Point at which the process of counting the number of retry executions starts

● : Point at which the number of retry executions is counted

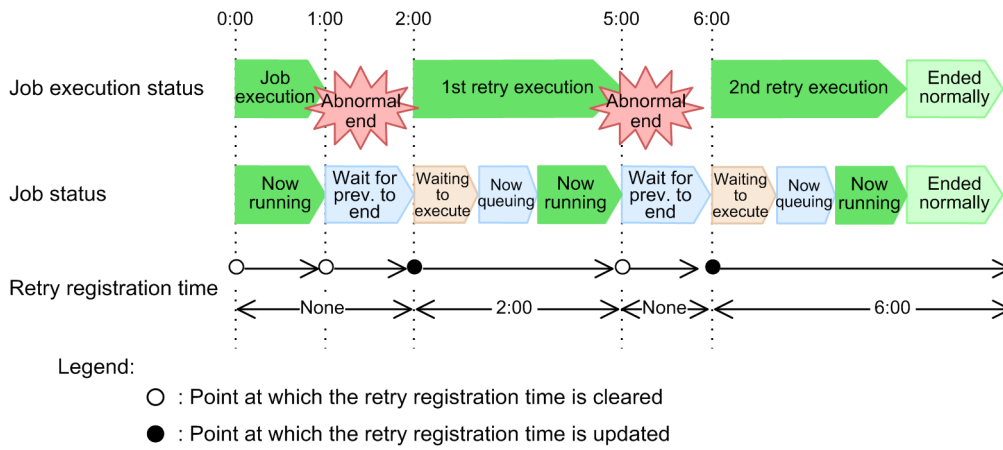
#### Retry registration time

In a retry execution, the retry registration time is the time the job enters the *Waiting to execute* status.

The specified retry registration time is cleared when an executable file specified for a job ends abnormally and the job enters the *Wait for prev. to end* status. When the job enters the *Waiting to execute* status, the retry registration time is updated to the time that the job enters that status. If retry is executed multiple times, the retry registration time is updated to the time at which the job enters the *Waiting to execute* status for the last time.

The following figure shows updating of the retry registration time.

Figure 2–115: Updating the retry registration time



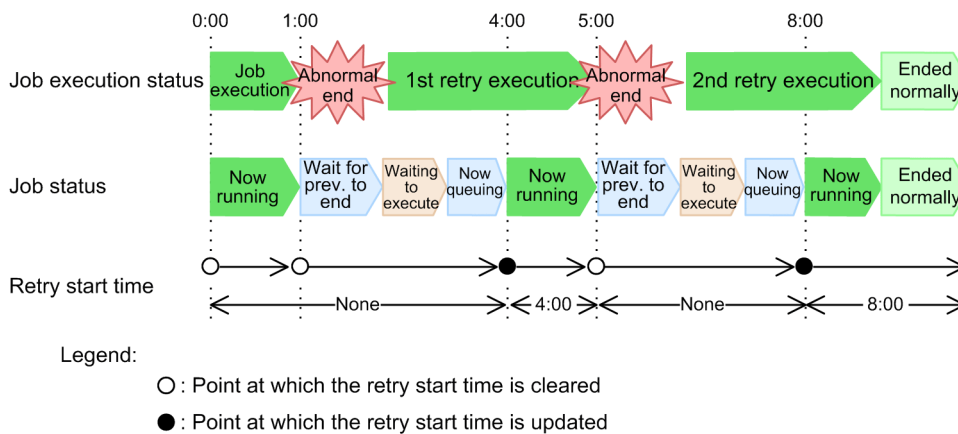
### Retry start time

In a retry execution, the retry start time is the time that the job enters the *Now running* status.

The specified retry start time is cleared when an executable file specified for a job ends abnormally and the job enters the *Wait for prev. to end* status. When the job enters the *Now running* status, the retry start time is updated to the time that the job enters that status. If retry execution is performed multiple times, the retry start time is updated to the time at which the job enters the *Now running* status for the last time.

The following figure show the updating of the retry start time.

Figure 2–116: Updating the retry start time



You can check the retry information in JP1/AJS3 - View windows or by using a command. The following table describes the JP1/AJS3 - View windows and command you can use and the retry information you are able to check.

Table 2–39: JP1/AJS3 - View windows and command that you can use to check the retry information

| No. | JP1/AJS3 - View window and command for checking retry information  | Retry information you can check   |
|-----|--|---|
| 1   | <ul style="list-style-type: none"> <li>• Monitor Details dialog box</li> <li>• Detailed Schedule dialog box</li> <li>• <code>ajsshow</code> command</li> </ul> | All retry information   |
| 2   | <ul style="list-style-type: none"> <li>• Daily Schedule window</li> <li>• Monthly Schedule window</li> <li>• Jobnet Monitor window</li> </ul>                  | <ul style="list-style-type: none"> <li>• <b>Retry status</b></li> <li>• <b>Retry execution times</b></li> </ul> |
| 3   | Search window  | <b>Retry execution times</b>  |

If retry execution is performed multiple times, you can use JP1/AJS3 - View windows and the `aj sshow` command to check the result of the last retry execution. If you want to check the result of each retry execution, use JP1 events, scheduler logs, or the Execution Result Details dialog box.

## (2) Monitoring jobs with retry settings

The following describes the timeout period and monitoring for a end delay for jobs using retry settings.

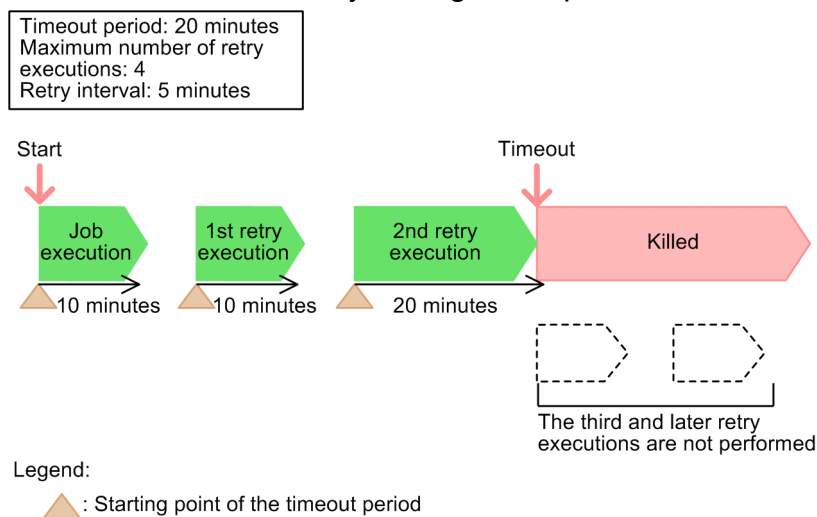
### (a) Timeout period for jobs with retry settings

When you specify both a timeout period and retry settings for a job, the elapsed time for the job is reset at the beginning of each retry execution. The elapsed time for a job is monitored to determine when the timeout period expires.

If you want to monitor the entire elapsed time for a job that includes multiple retry executions, monitor a end delay for the jobnet containing the job for the length of time required for executing the jobnet. To do so, in the Define Details dialog box for the jobnet containing the job, select the **Time-required-for-execution** check box for **Monitor jobnet**, and then specify the monitoring time.

The following figure shows how the elapsed time for a job is monitored to determine the expiration of the timeout period when retry settings are specified.

Figure 2–117: Monitoring the elapsed time for a job to determine the expiration of the timeout period when retry settings are specified



In this example, 20 minutes have passed since the start of the job during the first retry execution. However, the elapsed time for the job (monitored to determine when the timeout period expires) is reset at the beginning of each retry execution. For this reason, the job does not enter the *Killed* status. When 20 minutes have passed since the start of the second retry execution, the job enters the *Killed* status.

Note that if the timeout period has expired and the job enters the *Killed* status when the number specified in **Maximum retry times** has not been reached, no more retry executions will be performed.

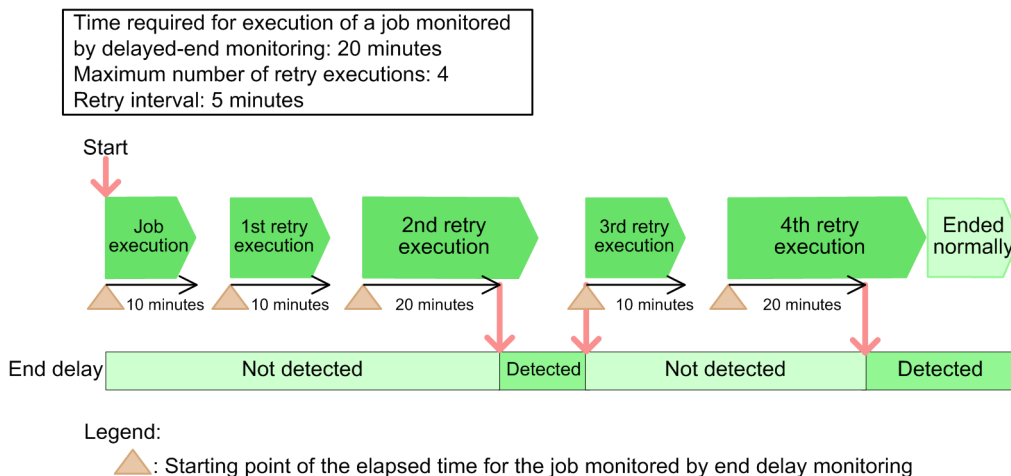
### (b) End delay monitoring for jobs with retry settings

When you specify both delayed monitoring and retry settings for a job, the elapsed time for the job is reset at the beginning of each retry execution, and the elapsed time for a job is monitored to determine if a end delay has occurred.

If you want to monitor the entire elapsed time for a job that includes multiple retry executions, use end delay monitoring, which monitors the jobnet containing the job for the length of time required for the jobnet.

The following figure shows how the elapsed time for a job is monitored to determine if a end delay has occurred when retry settings are specified.

Figure 2–118: Monitoring the elapsed time for a job to determine if a end delay has occurred when retry settings are specified



In this example, 20 minutes have passed since the start of the job during the first retry execution. However, the elapsed time for the job (monitored to determine if a end delay has occurred) is reset at the beginning of each retry execution. For this reason, a end delay is not detected. When 20 minutes have passed since the start of the second retry execution, a end delay is detected. When retry execution is performed after a end delay is detected, end delays are no longer detected.

If a end delay is detected as a result of the last retry execution, the delay information continues to be displayed even after the job has ended.

**Cautionary note**

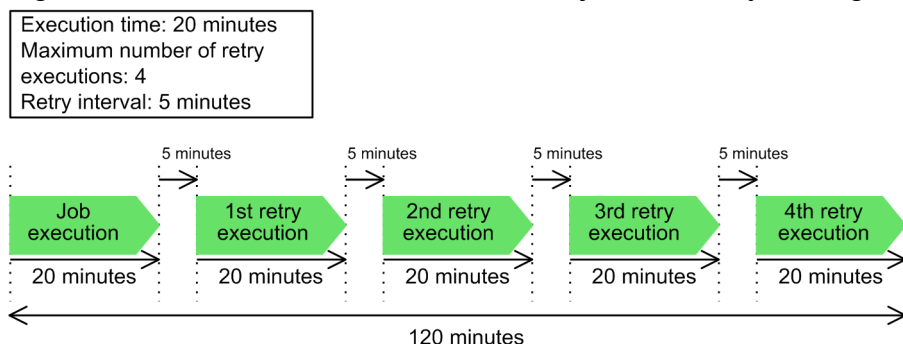
If a end delay is detected for a job after which a end delay has not been detected as a result of the next retry execution, the upper-level jobnet enters the *Nested jobnet delayed end* status.

### (3) Execution simulation of jobs with retry settings

When you simulate the execution of a job with retry settings, the simulation includes the time required for waiting for an automatic retry and the length of time for retry execution.

The following figure shows an example of execution simulation of a job with retry settings.

Figure 2–119: Execution simulation of a job with retry settings



In this example, the execution time for the job is 20 minutes. When the job was defined, 4 was specified for **Maximum retry times** and 5 minutes was specified for **Retry interval**. As a result, the simulation takes a total of 120 minutes (the



time for the original job execution, the time for four retry executions, and the retry intervals between the original and retry executions).

## (4) Behavior of units with retry settings

The following describes the behavior of jobs with retry settings, the behavior of preceding units of jobs with retry settings, and the behavior of upper-level units of jobs with retry settings.

### (a) Re-executing jobs with retry settings and preceding units

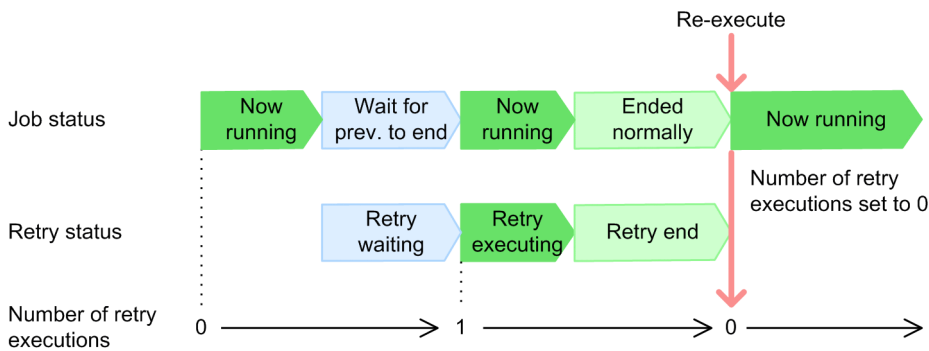
The following describes the behavior of units when you re-execute jobs after automatic retry has ended, re-execute processing from preceding jobs during a retry, and re-execute only preceding units during a retry.

#### ■ Re-executing jobs after automatic retries have ended

When you re-execute a job after automatic retries have ended, the number of retry executions is also reset.

The following figure shows how a job behaves when you re-execute a job after automatic retries have ended.

Figure 2–120: Re-executing a job after automatic retries have ended



When you re-execute a job, the number of retry executions is set to 0.

#### ■ Re-executing processing from preceding units during a retry

When you re-execute processing from a preceding unit during a retry, the number of retry executions is reset. The job being retried transitions to the *Wait for prev. to end* status and waits for the preceding unit to end.

The time at which the job being retried transitions to the *Wait for prev. to end* status to wait for the preceding unit to end depends on the status of the job being retried at the time the preceding unit is re-executed.

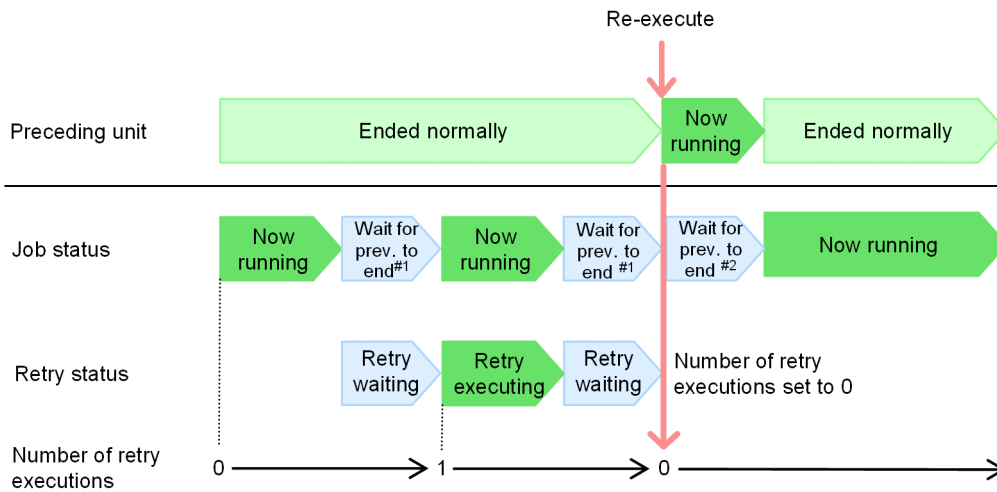
The following describes how the job behaves when you re-execute processing from a preceding unit during a retry.

When the job being retried is in the *Wait for prev. to end* or *Being held* status

The job being retried enters the *Wait for prev. to end* status when the preceding unit is re-executed, and then the number of retry executions is reset. The job waits for the preceding unit to end. When the preceding unit ends, the job is re-executed.

The following figure shows how the job being retried behaves when you re-execute processing from the preceding unit while it is in the *Wait for prev. to end* status.

Figure 2–121: Behavior of a job in the Wait for prev. to end status during a retry when you re-execute processing from the preceding unit



#1: The job waits until the retry interval expires.  
 #2: The job waits until the preceding ends.

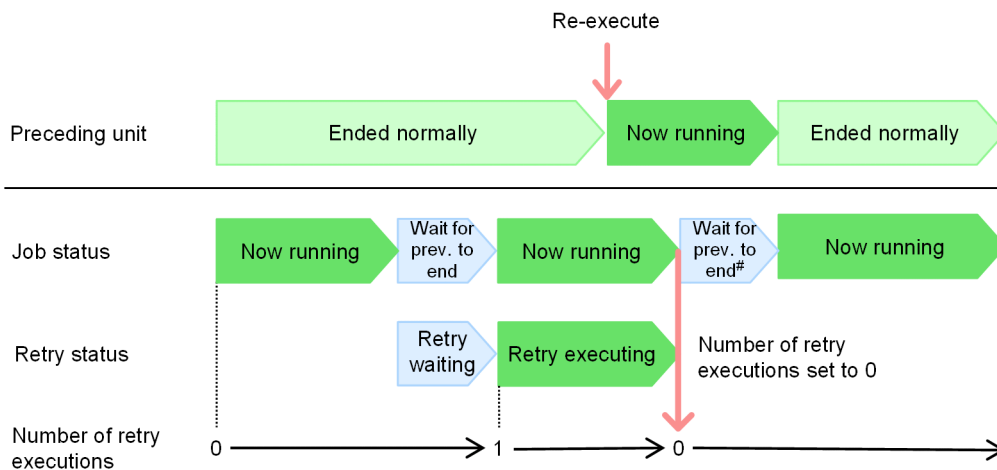
When the preceding unit is re-executed, the number of retry executions is set to 0.

When the job being retried is in the *Waiting to execute*, *Now queuing*, or *Now running* status

The job being retried is executed until it ends. When execution ends, the job enters the *Wait for prev. to end* status, regardless of whether the job ended normally, and the number of retry executions is reset. After that, the job is re-executed when the preceding unit ends.

The following figure shows how the job being retried behaves when you re-execute processing from the preceding unit while it is in the *Now running* status.

Figure 2–122: Behavior of the job in the Now running status during a retry when you re-execute from the preceding unit



#: The job waits until the preceding ends.

When the preceding unit is re-executed, the number of retry executions is set to 0.

### ■ Re-executing only preceding units during a retry

When you re-execute only a preceding unit during a retry, the job transitions to the *Wait for prev. to end* status when the retry execution ends and the job waits for the retry interval to expire. If the preceding unit does not end after the retry

interval has expired, the job waits for the preceding unit to end and will be retried until the number of retry executions is reached.

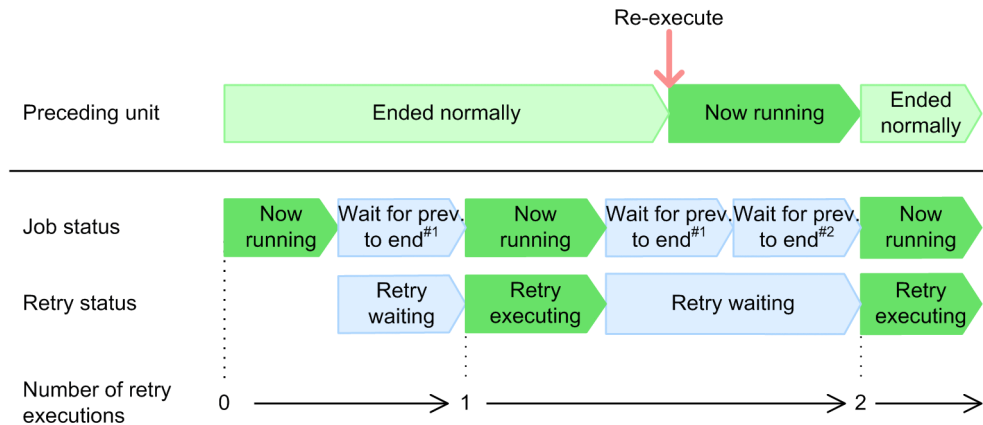
The following describes how the job being retried behaves when you re-execute only the preceding unit.

When the job being retried is in the *Wait for prev. to end* or *Being held* status

The job during a retry waits for the preceding unit to end after the retry interval has expired. When the preceding unit ends, the job is retried again.

The following figure shows how the job being retried behaves when you re-execute only the preceding unit while the job being retried is in the *Wait for prev. to end* status.

Figure 2–123: Behavior of a job in the *Wait for prev. to end* status during a retry when you re-execute only a preceding unit



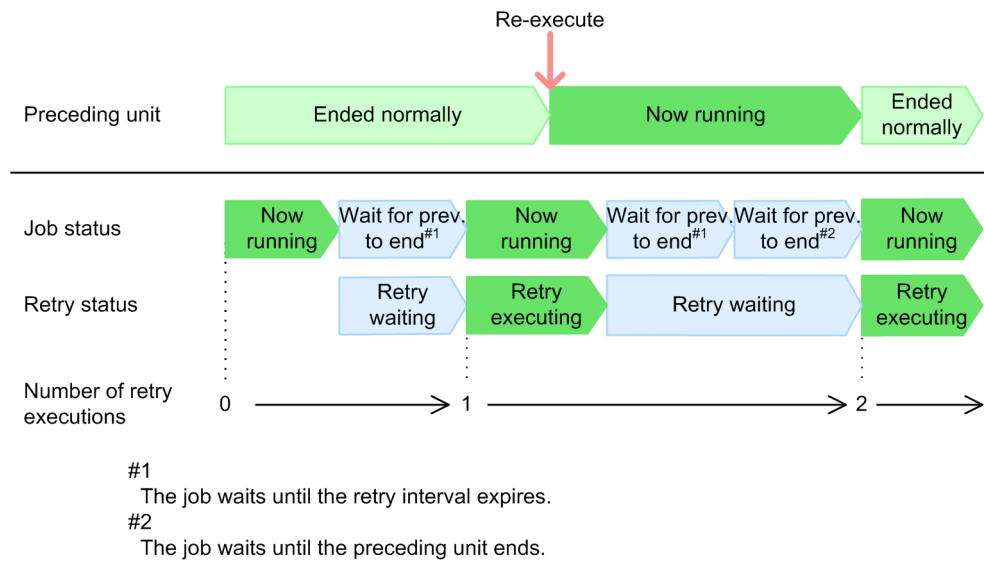
- #1 The job waits until the retry interval expires.
- #2 The job waits until the preceding unit ends.

When the job being retried is in the *Waiting to execute*, *Now queuing*, or *Now running* status

The current retry execution will finish. If the executable file ends abnormally during the current retry execution, the job waits for the preceding unit to end after the retry interval has expired. When the preceding unit ends, the job is retried again.

The following figure shows how a job during a retry behaves when you re-execute only the preceding unit while the job being retried is in the *Now running* status.

Figure 2–124: Behavior of a job in the Now running status during a retry when you re-execute only the preceding unit



### (b) Suspending root jobnets during job retries

You can suspend a root jobnet containing a job being retried or cancel the suspension of a root jobnet.

While the root jobnet is suspended, retry execution is not performed for jobs even when a job has ended abnormally. When you cancel suspension, retry execution is performed after the retry interval expires.

For details about how to suspend root jobnets, see *4.5.17 Changing job and jobnet definitions without unregistering the jobnet* in the manual *JP1/Automatic Job Management System 3 Overview*.

### (c) Interrupting root jobnets containing jobs being retried

When you interrupt a root jobnet containing a job being retried, the root jobnet enters the *Interrupted* status and the job with retry settings enters the *Not executed + Ended* status.

The time that the job being retried enters the *Not executed + Ended* status depends on the status of the job being retried.

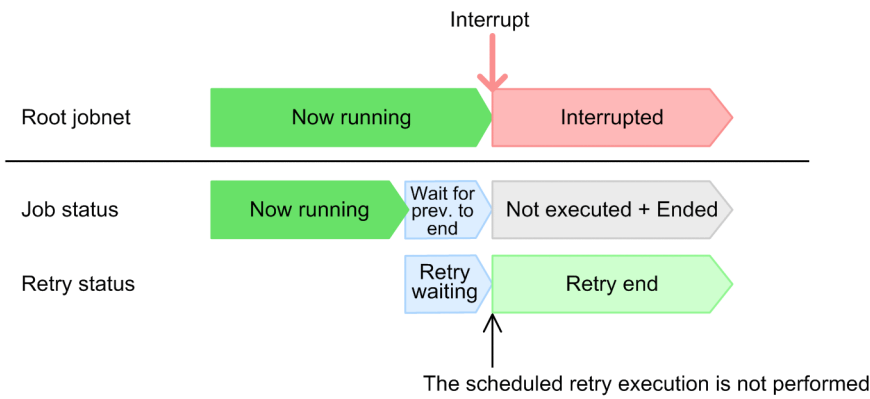
The following describes how a job being retried behaves when the root jobnet is interrupted.

When the job being retried is in the *Wait for prev. to end* or *Being held* status

The scheduled retry execution is not performed. The job enters the *Not executed + Ended* status and the retry status transitions to *Retry end*.

The following figure shows how a job during a retry behaves when you interrupt the root jobnet while the job being retried is in the *Wait for prev. to end* status.

Figure 2–125: Behavior of a job in the Wait for prev. to end status during a retry when you interrupt the root jobnet

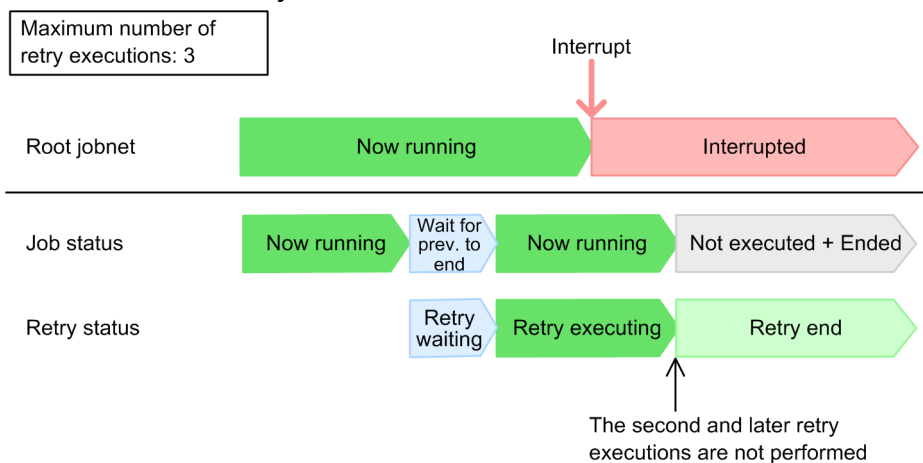


When the job being retried is in the *Waiting to execute*, *Now queuing*, or *Now running* status

The current retry execution will finish. Regardless of whether the job ends normally, the ended job enters the *Not executed + Ended* status and the retry status transitions to *Retry end*.

The following figure shows how the job being retried behaves when you interrupt the root jobnet while the job being retried is in the *Now running* status.

Figure 2–126: Behavior of a job in the Now running status during retry when you interrupt the root jobnet



When you interrupt the root jobnet containing a job being retried, the error in the executable file defined for the job will not be corrected. The job enters the *Not executed + Ended* status and the retry status transitions to *Retry end*.

#### (d) Changing the status of jobs and killing jobs during retries

When a job enters the *Now running*, *Now queuing*, or *Waiting to execute* status due to an automatic retry (for jobs for which **Queueless Agent** is specified as the execution target service), you can change the status of the job to an end status. You can also change the status of a job in an end status due to an automatic retry to another end status. You can kill a job being automatically retried as well. When you change the status of a job or kill a job, no automatic retry will be performed for the job after it enters an end status, regardless of the new end status or the return code.

#### (e) Changing the definitions of jobs during a retry

When *yes* is specified for the `UNITDEFINERLOAD` environment setting parameter and you change the definition of a job during a retry, the definition of the job is read again each time retry execution is performed. Accordingly, when you change the definition of a job during a retry, the new definition takes effect from the next retry execution after the change.

For details about the behavior of jobs when you change the definitions, see *7.4 Changing the unit definition information during registration for execution in the JPI/Automatic Job Management System 3 Administration Guide*.

If you delete retry settings when the retry status is *Retry waiting*, only one retry execution is performed. Thereafter, no more retry executions will be performed.

**Cautionary note**

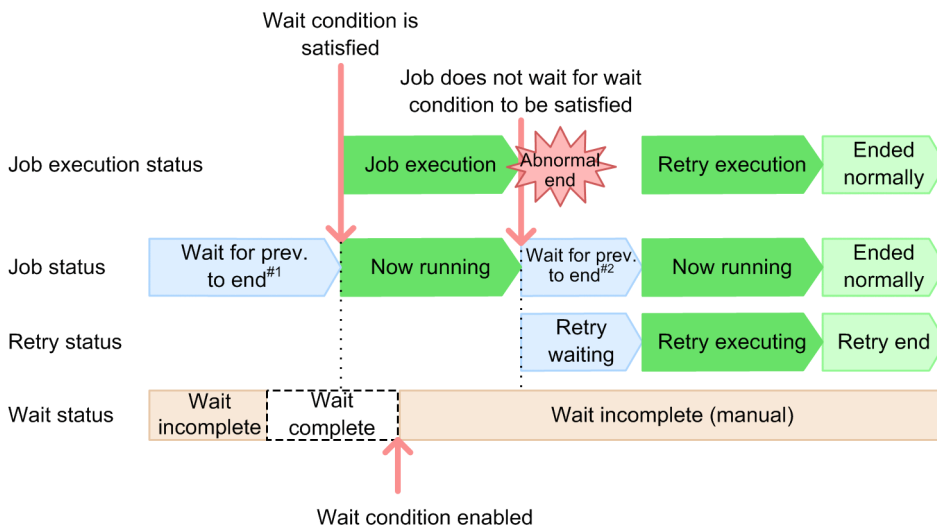
When you change the definition of a job during a retry, the subsequent behavior of the job changes depending on the status of the job at the point when the change is made. The job might therefore behave unexpectedly. We recommend that you do not change any definitions during a retry.

**(f) Waiting during a job retry**

When a wait condition is assigned to a job with retry settings, after the wait condition is satisfied and the wait status is set to *Wait complete*, the job does not wait for the job being waited for during retry executions. If you enable the wait condition again and change the wait status to *Wait incomplete (manual)* before the retry status transitions to *Retry waiting*, the job only waits for the retry interval to expire and does not wait for the wait condition to be satisfied.

The following figure shows how a job with retry settings behaves when you assign a wait condition to it.

Figure 2–127: Behavior of a job with retry settings when you assign a wait condition to it



#1 The job waits for the wait condition to be satisfied.

#2 The job waits for the retry interval to expire.

**(5) Information to be updated by retry execution**

When retry execution is performed, information in addition to the retry information is also updated. If you reference that information while a job is being executed, the results might differ depending on when you reference it. Define jobs taking this point into consideration.

The following table describes the information that is updated by retry execution.

Table 2–40: Information updated by retry execution

| No. | Updated information           | Description  |
|-----|-------------------------------|--|
| 1   | JP1JobID environment variable | The job ID is updated to the ID used at retry execution.   |
| 2   | Standard output file          | When the <b>Append</b> check box is not selected in the Define Details dialog box for jobs<br>The contents of the file are overwritten each time retry execution is performed.<br>When the <b>Append</b> check box is selected in the Define Details dialog box for jobs<br>Information is added to the file each time retry execution is performed. |
| 3   | Standard error output file    | When the <b>Append</b> check box is not selected in the Define Details dialog box for jobs<br>The contents of the file are overwritten each time retry execution is performed.<br>When the <b>Append</b> check box is selected in the Define Details dialog box for jobs<br>Information is added to the file each time retry execution is performed. |
| 4   | Transfer file                 | The file is transferred each time retry execution is performed.  |
| 5   | Execution result details      | Information is added each time retry execution is performed.   |

Cautionary note

When you select the **Append** check box, the sizes of the standard output file and the standard error output file increase, creating a high system load. We recommend that you do not select the **Append** check box. If you need to select the **Append** check box, limit the amount of output information or periodically clear the files.

## (6) Restarting the scheduler service during a retry

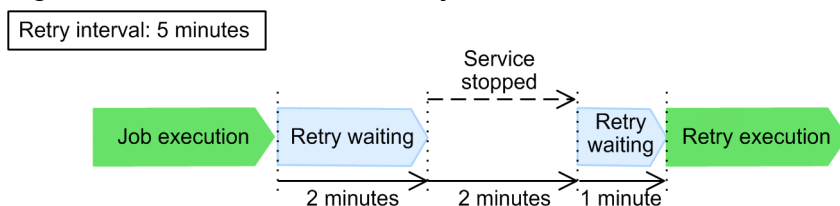
When you stop the scheduler service and restart it, the jobs being retried enter the same status as regular jobs depending on the start mode of the scheduler service. The differences are as follows.

- Starting the scheduler service in warm start mode or disaster recovery start mode  
Automatic retries do not continue. Jobs enter an end status specified for each start mode based on their job status at that time.
- Starting the scheduler service in hot start mode  
Automatic retries continue. The retry interval and the number of retry executions are handled as follows.

Retry interval

The period during which the scheduler service is inactive is included in the elapsed time for the jobs monitored to determine when the retry interval expires. Suppose you have a job for which retry execution is performed every five minutes. When the scheduler service stops for two minutes after two minutes have passed since the retry status transitions to the *Retry waiting* status, and the scheduler service restarts in hot start mode, retry execution starts one minute after the scheduler service restarts.

Figure 2–128: Behavior of the job when the scheduler service starts in hot start mode



Note that if you restart the JP1/AJS3 service or the scheduler service in hot start mode after it has stopped for more than the retry interval, jobs start executing as soon as the scheduler service starts. If that occurs, the number of jobs temporarily increases, possibly degrading job execution performance for a while.

#### Number of retry executions

The number of retry executions is not initialized. The number continues to increase as retry executions are performed. Suppose you have a job for which 5 is specified for **Maximum retry times**. If you restart the scheduler service when two retry executions have finished, a maximum of three more retry executions will be performed after the scheduler service restarts.

For details about the status of jobs for each start mode, see *6.2.1 Temporarily changing the start mode of JP1/AJS3* in the *JP1/Automatic Job Management System 3 Administration Guide*.

## (7) Cautionary notes on automatic retry

Note the following when you use automatic retries:

- When retry settings are specified for a unit under a remote jobnet and either of the following conditions is satisfied for the manager executing the jobnet, an error occurs:
  - The database uses a compatible ISAM configuration.
  - The version of JP1/AJS3 - Manager is 09-50 or earlier.
- An automatic retry is not performed while job restrictions or forced termination of jobs are in effect to stop the scheduler service. Jobs enter an end status without performing retry executions regardless of the return codes and the number of retry executions. When all the running jobs have ended, the scheduler service stops.  
For details about restricting how the scheduler service stops, see *7.5.2 Stopping the scheduler service* in the *JP1/Automatic Job Management System 3 Administration Guide*.
- Sometimes jobs are ended by the OS with return codes instead of by user applications. In such cases, an automatic retry is performed.
- For queueless jobs, an automatic retry is also performed if there are no script files or script files cannot be accessed. Use the definition pre-check function to make sure that script files can be accessed.
- When you execute queueless jobs, use version 10-00 or later of JP1/AJS3 - Manager or JP1/AJS3 - Agent. If the version is 09-50 or earlier, an automatic retry is also performed when the process for starting job processes ends abnormally for some reason.



# 3

## Operation Calendar and Execution Schedule Considerations

After deciding the work tasks that you want to automate, you need to plan a calendar and execution schedule for work tasks in JP1/AJS3.

This chapter describes the considerations necessary for setting a calendar and execution schedule.

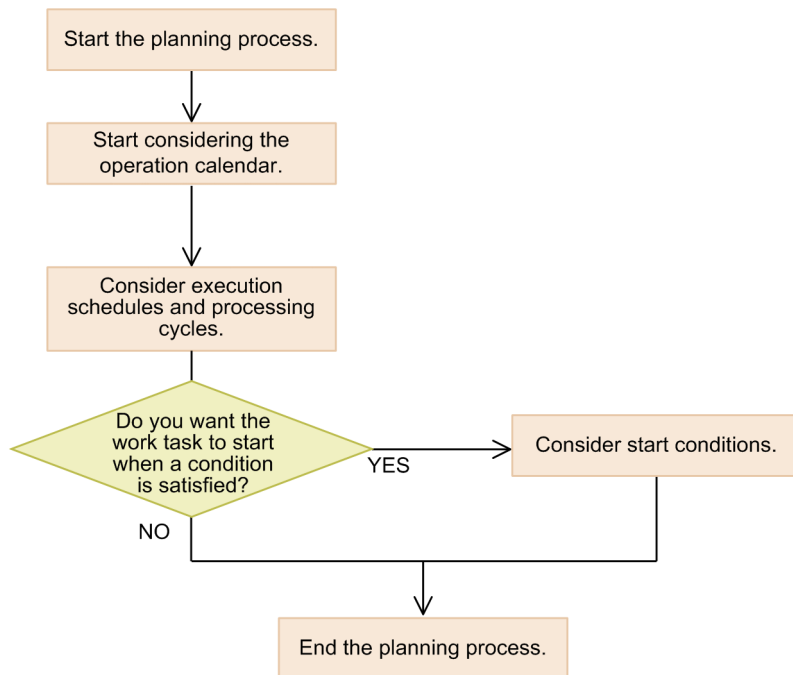
### 3.1 Flow of calendar and schedule planning

---

The flow for setting the schedule and calendar differs depending on the pattern of jobnet execution that you adopt.

The following figure shows the flow for setting the schedule and calendar based on the jobnet execution pattern.

Figure 3–1: Flow of calendar and schedule planning



## 3.2 Considerations when defining a calendar for JP1/AJS3 operation

You can create a JP1/AJS3 operational calendar for jobnet execution, in which open days and closed days such as the Sundays and holidays in an ordinary calendar are defined. When preparing the calendar, you need to consider the base day and base time settings, as well as which days to designate as open days and closed days.

Table 3–1: Matters to consider when creating a calendar

| Consideration             | Description  |
|---------------------------|--|
| Open days and closed days | Consider the days on which jobnets will be executed ( <i>open days</i> ) and the days on which jobnets will not be executed ( <i>closed days</i> ).  |
| Base day                  | Consider which day is to serve as the first day of the month.<br>You can designate the base day as a specific date, or as the <i>n</i> th occurrence of a specific day of the week.<br>For example:<br>If the closing date for calculating salaries is the 20th of each month, by setting the 21st as the base day you can fix the period from the 21st of one month to the 20th of the next month as one month for the purpose of salary calculation. |
| Base time                 | Consider the time at which you want each day to start.<br>For example:<br>By setting 8:00 as the base time, processing that takes place at 1:00 on the following calendar day is treated as part of the current day.   |
| Exclusive schedule        | Consider whether you want a particular jobnet not to be executed if its execution schedule coincides with that of another jobnet.  |

For details about defining a calendar for JP1/AJS3 operation, see 3.2 *Defining a calendar for JP1/AJS3 operation* in the manual *JP1/Automatic Job Management System 3 Overview*.

### 3.3 Considerations when defining a jobnet execution schedule

---

Plan the *schedule rules* that determine the execution schedule for a jobnet. These include the start time and processing cycle for the jobnet.

The following table lists and explains the considerations when you plan jobnet execution schedules.

Table 3–2: Matters to consider when defining a jobnet schedule

| Consideration                       | Description   |
|-------------------------------------|---|
| Execution start date and time       | Consider the following matters, which determine how the start date and time of the jobnet are calculated: <ul style="list-style-type: none"><li>• The method of specifying the start date (Registered day, Absolute day, Relative day, Open day, Closed day)</li><li>• The method of specifying the start time (Absolute time, Relative time)</li><li>• Whether to set a base day and base time</li></ul> |
| Processing cycle                    | Consider whether to execute jobnets on a regular schedule. You can run a jobnet weekly, monthly, or yearly.   |
| Method for closed day substitutions | Consider whether jobnets whose planned execution date according to the processing cycle falls on a closed day are to be executed on another day instead.  |

For details about schedule rules, see *3.3 Defining a schedule* in the manual *JP1/Automatic Job Management System 3 Overview*. For details about how to set schedules for a variety of operation patterns, see *3.5 Setting schedules*. Use these references when you consider how to define a jobnet execution schedule.

## 3.4 Start condition considerations

---

JP1/AJS3 allows you to execute even irregular work tasks, which do not permit specification of a time of execution in advance, by specifying a *start condition* that stipulates the conditions under which the work task starts. As a start condition, you define an event job or a custom event job that monitors for an event like those listed below. You can associate more than one event job or custom event job with a start condition.

- When a particular file is updated (Monitoring Files job)
- When a particular JP1 event is received (Receive JP1 Event job)
- When a specified period of time has elapsed (Interval Control job)
- When a particular character string is output to a log file (Monitoring Log Files job)
- When email is received (Receive Mail job<sup>#</sup>)
- When an event monitored by the linked program occurs (Custom Event job)

<sup>#</sup>: Linkage with a mail system is required.

For details about start conditions, see *3.4 Defining a start condition* in the manual *JP1/Automatic Job Management System 3 Overview*.

### 3.4.1 Detailed considerations

Consider the following when using a start condition to execute a jobnet.

#### (1) Valid range of start conditions

Consider the range in which the occurrence of an event defined as a start condition is to be monitored. You can specify the valid range as a number of executions or a specific time. Set the valid range of a start condition when defining the schedule rules for a jobnet.

Number of executions

Specify the number of times that the jobnet can be executed from the time that monitoring for the start condition begins.

Period

Monitoring of the start condition continues until the specified time arrives. You can specify the time as an absolute or relative time.

#### (2) Operation when multiple event jobs and custom event jobs are used in a start condition

When multiple event jobs and custom event jobs are defined, consider whether you want the start condition to be satisfied when all the events occur (an AND condition), or when any one of the defined events occurs (an OR condition).

#### (3) Concurrent execution of monitoring and execution generations

When you execute a jobnet with a start condition, generations which monitor for the occurrence of the events defined in the start condition (monitoring generations), and generations which are executed when the start condition is satisfied (execution generations), are both generated. For jobnets with start conditions, consider whether to allow concurrent execution for monitoring generations, and for execution generations.

### Concurrent execution of execution generations

In situations where the start condition is satisfied multiple times, consider whether you want a new generation of the jobnet to run concurrently with the previous generation or to wait until the previous generation has ended. This behavior is determined by the concurrent execution setting in the jobnet definition.

If you disable concurrent execution, consider whether to hold execution generations as described in (4) *Holding execution generations*.

### Concurrent execution of monitoring generations

For jobnets with a start condition and a processing cycle, consider how you want a new monitoring generation to be executed if the previous monitoring generation is still running when its start time is reached. You can have the new monitoring generation execute concurrently, wait until the previous monitoring generation has completed, or skip execution altogether.

## (4) Holding execution generations

If you disable concurrent execution, you can choose whether to hold execution generations that satisfy the start condition. You can select from the following behavior:

- Does not stay with skip  
Execution generations in *Wait for start cond.* status are not held, and transition to *Skipped so not exe.* status.
- Stay without skip  
Execution generations in *Wait for start cond.* status are held.

## (5) When a jobnet with a start condition ends abnormally

Consider the behavior of the system after a jobnet with a start condition ends abnormally. You can select from the following behavior:

- Start execution of jobnet  
After ending abnormally, the jobnet starts at every subsequent occurrence of the event being monitored by the start condition.
- Hold start of jobnet  
After ending abnormally, the jobnet is placed in *Held* status, or remains in *Wait for start cond.* status with future execution suspended.
- Stop monitoring of start conditions  
After the jobnet ends abnormally, the system stops monitoring for the start condition.

## (6) Other considerations

Also consider the following:

- How often a condition will be satisfied within a given time frame<sup>#</sup> (how many event jobs and custom event jobs are likely to be executed).  
#: You can use this information when you estimate event job and custom event job performance.

## 3.5 Setting schedules

---

You can adapt JP1/AJS3 to operate in a variety of ways, using functions such as the jobnet scheduling function.

### 3.5.1 Establishing schedules for applications that extend over two days

A work task executed late at night will often extend over more than one day. Sometimes the date change can lead to the system reporting a scheduling error. For example, suppose that you calculate the daily sales data from Monday to Friday at 1:00 a.m. on the following day, and Saturdays are closed days. If you set the start time for the calculation process to 1:00 on the following day, the calculation of Friday's sales data will be scheduled to begin at 1:00 on Saturday. However, because Saturday is a closed day, the calculation process cannot be executed.

By setting 25:00 as the start time, you can ensure that the calculation process is executed even when Saturday is a closed day. However, sometimes the root jobnet and the nested jobnet will be scheduled to execute on different days. As a result, the nested jobnet might fail to execute.

In such cases, you can change the length of time that JP1/AJS3 treats as 1 day, so that a process that runs over two days can fit within 1 day under JP1/AJS3. You can use one of the following methods to change the length of time that JP1/AJS3 treats as one day.

- Set 1 day to 48 hours, and calculate schedules based on a 48-hour day.  
With this method, you define the schedule of the root jobnet using the 48-hour schedule.
- Set the start time for 1 day to a time other than 0:00.  
With this method, you specify a time other than 0:00 as the base time.

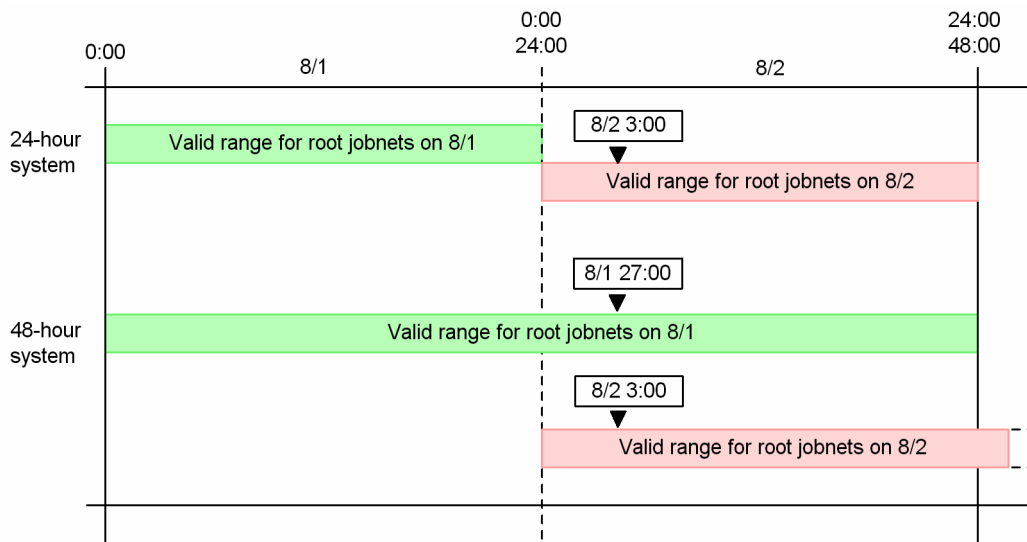
We recommend that you use the 48-hour schedule to define the schedule of the root jobnet. This method enables you to create schedules more easily. Each method is described below.

#### (1) Defining an application that extends over two days using a 48-hour schedule

If you define the schedule of the root jobnet using the 48-hour schedule, the time between 0:00 and 23:59 on the next day in the calendar is treated as part of the current day, from 24:00 to 47:59. For example, you can refer to 1 a.m. on Saturday as 25:00 on Friday. If Saturday is a closed day, as in the example above, you can use the 48-hour schedule and set the execution start time to 25:00 on Friday. This will allow the job to execute on schedule.

The following figure shows the difference between a 24-hour schedule and a 48-hour schedule.

Figure 3–2: Difference between a 24-hour schedule and a 48-hour schedule



If you specify the execution start time as an absolute time under the 48-hour schedule (base time 0:00), it can be expressed in two different ways. For example, a job that executes 3 a.m. can be expressed as 8/1 27:00 or 8/2 3:00. Although the actual time is the same, the scheduled execution date is different. This means that when the schedule is affected by closed days/open days, or by shifting the execution schedule, the application may not be executed.

Note that if you use the 24-hour schedule, you can still specify a time between 24:00 and 47:59. However, if for example you specify the time 8/1 27:00 under the 24-hour schedule, it is interpreted as 8/2 3:00, and the schedule will be set accordingly.

### (a) Changing the time system from the 24-hour schedule to the 48-hour schedule

To set up a schedule for a jobnet based on the 48-hour schedule, you need to set up the environment for the scheduler service.

To change a schedule from the 24-hour schedule to the 48-hour schedule:

1. Change the value of the `ROOTJOBNETSCHEDULERANGE` environment setting parameter of the scheduler service to 00000030.
2. Specify a base time of 0:00 in the details of the job group.  
If you do not specify a base time, the default is used.
3. Specify a time later than 24:00 as the execution start time for the jobnet.

For details about how to set environment setting parameters, see 4.2 *Environment setting parameter settings* in the *JP1/Automatic Job Management System 3 Configuration Guide* (for Windows systems), or 14.2 *Environment setting parameter settings* in the *JP1/Automatic Job Management System 3 Configuration Guide* (for UNIX systems).

### (b) Reviewing schedules after changing to the 48-hour schedule

When you change the schedule of a root jobnet from the 24-hour to the 48-hour schedule, the behavior of the schedule for the root jobnet will change. You should therefore review existing schedules after changing them to the 48-hour schedule. You should change the schedule rules for a jobnet when they meet any of the following conditions:

1. You defined a time later than 24:00 as the execution start time for the root jobnet.  
This causes the execution start date to change for the root jobnet and nested jobnets. Check the schedule definitions.



2. You defined an exclusive schedule for a jobnet or planning group that is in the same hierarchy as a jobnet that meets the condition 1 above.

This causes the execution start date of the exclusive schedule to change. Check the schedule definitions.

3. A jobnet that meets conditions 1 or 2 above contains a nested jobnet whose schedule does not depend on the upper-level jobnet.

This causes the execution start date of the nested jobnet to change. Check the schedule definitions.

4. The base time is other than 0:00.

Review the schedule definitions for root jobnets and nested jobnets because the way time is handled changes for the following items of the schedule rules:

- Execution start time
- Start delay time
- End delay time
- Valid range of a start condition

For information about how time is handled when a time other than 0:00 is specified for the base time, see 3.3.2 *Defining a schedule* in the manual *JP1/Automatic Job Management System 3 Overview*.

## (2) Defining applications that extend over two days by changing the base time

Normally, JP1/AJS3 considers 1 day to be the 24 hours between 0:00 on a certain day and 0:00 on the next day. By shifting the time when 1 day starts, you can fit processes that run over two different days into one day. For example, if you set the base time to 8:00, JP1/AJS3 treats the 24 hours from 8:00 until 8:00 the next day as 1 day. The time from 0:00 until 7:59 on the current day is counted as part of the previous day.

Changing the base time can complicate the applications involved in schedule definition, such as setting execution start times and delay times. For example, you must specify the time in different ways depending on the type of execution start time. When the base time is 8:00, you can specify a start time of 1:00 on August 5 in one of the ways shown below.

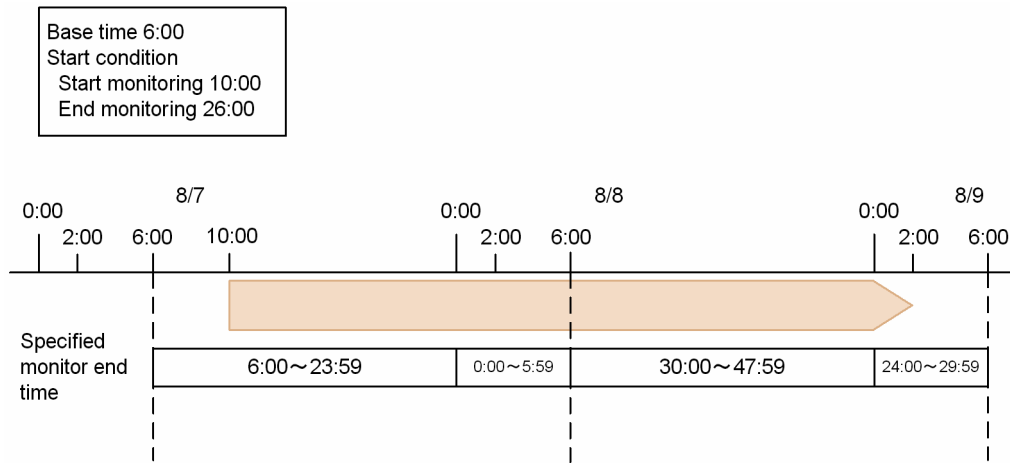
Table 3–3: Setting an execution start time

| Type of execution start time | Type of execution start date               | Time specified                     |
|------------------------------|--|------------------------------------|
| Absolute time                | Registered day, Absolute day, Relative day | 8/5 1:00 (alternatively 8/4 25:00) |
| Absolute time                | Open day, Closed day                       | 8/4 1:00                           |
| Relative time                | All  | 8/4 17:00                          |

### (a) When a start condition is defined for a jobnet

Imagine you are using the 24-hour schedule, and have set the base time to a value other than 0:00. If you specify a start condition for a jobnet, and an end time for the start condition that is later than 24:00 but earlier than the base time, the end time of the start condition is treated as falling on the day after next. The following figure shows an example where this happens.

Figure 3–3: End time of the start condition becomes the day after next



From this example, it is apparent that if you have set the base time to a value other than 0:00, specifying a time between 0:00 (24:00) and the base time is complicated. We therefore recommend that you use the 48-hour schedule to define the schedule.

### 3.5.2 Setting multiple execution start times

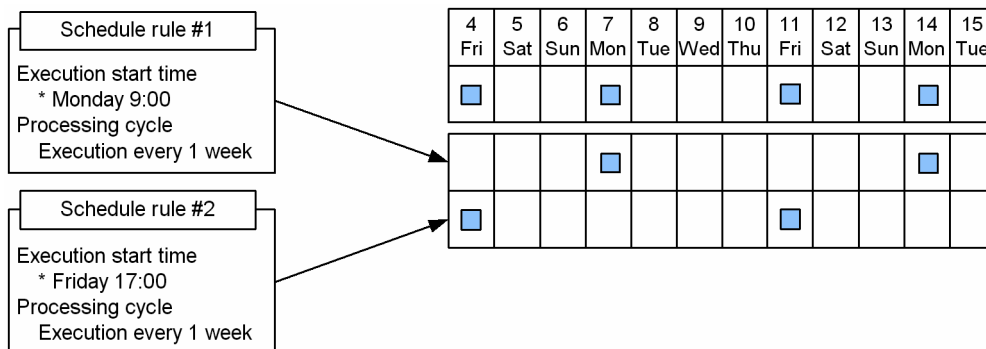
Sometimes, you may need to operate a jobnet according to a complex schedule that cannot be expressed by a single schedule rule. JP1/AJS3 allows you to create multiple schedule rules for a jobnet. You can define up to 144 schedule rules for a single jobnet. When multiple schedule rules are defined, the schedule is set with the earliest schedule rule.

For example, you may want to run a jobnet multiple times a day at fixed times, or for the jobnet to have different execution times on different days of the week. In such cases, you can define the schedule for the jobnet by creating multiple schedule rules. If more than one execution is scheduled for the same execution start date and time, as a result of setting multiple schedule rules, then the jobnet is executed only once.

Note that when you set multiple schedule rules, a jobnet may end up being scheduled for execution more than once at the same execution start time. In this case, the jobnet will be executed only once.

The following is an example of a schedule with multiple schedule rules.

Figure 3–4: Schedule with multiple schedule rules



This schedule is for a jobnet that will execute twice a week, at 9:00 on Monday and 17:00 on Friday.

Schedule rule #1 defines Monday's schedule, and Schedule rule #2 defines Friday's schedule. When you register the jobnet for execution, the execution schedule for the jobnet is calculated based on both of these schedule rules.

### 3.5.3 Defining a different schedule for some jobs in a jobnet

To execute some jobs in a jobnet according to different schedules, use nested jobnets.

In the same way as for root jobnets, you can set schedule rules and schedule options for nested jobnets. You define schedule rules for nested jobnets by linking them with the schedule rules for the root jobnet. You can link multiple nested jobnet schedule rules to one root jobnet schedule rule.

If you do not define a schedule for a nested jobnet, the nested jobnet is executed according to the same schedule as the root jobnet.

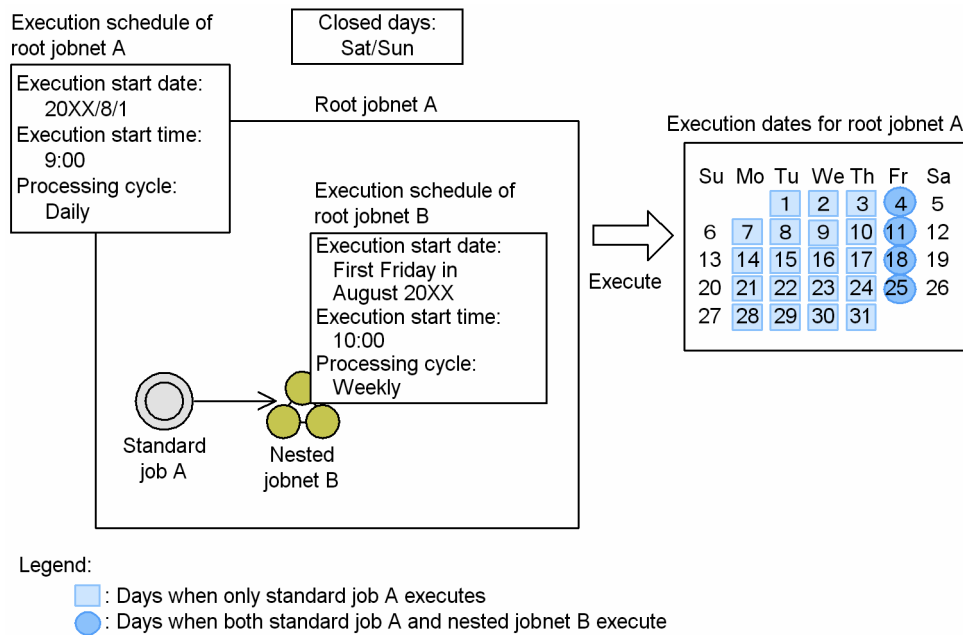
#### (1) Defining a schedule for a nested jobnet

First, create a nested jobnet under a root jobnet. Define the jobs that you want to execute according to a different schedule under the nested jobnet. Once you have defined the nested jobnet, set the schedule rules for the nested jobnet.

Now register your defined root jobnet for execution. The nested jobnet runs according to the schedule you defined. However, the nested jobnet is only executed when the execution conditions of the upper-level jobnet are met. Even if you have defined a schedule for the nested jobnet, it will not be executed on days when the root jobnet is not scheduled for execution.

The following figure shows the use of a schedule for a nested jobnet.

Figure 3–5: Using a schedule to execute a nested jobnet



In this example, the nested jobnet B is executed only on Fridays. This means that nested jobnet B is not executed on Monday through Thursday, regardless of whether jobnet A is executed.

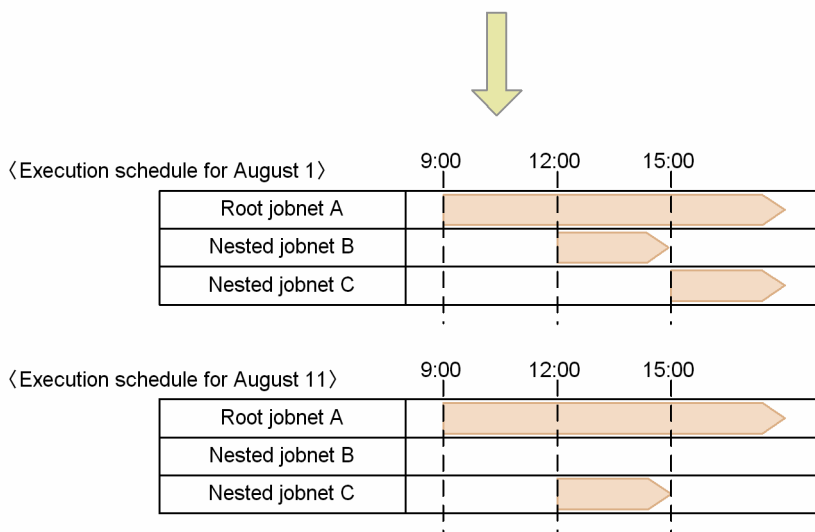
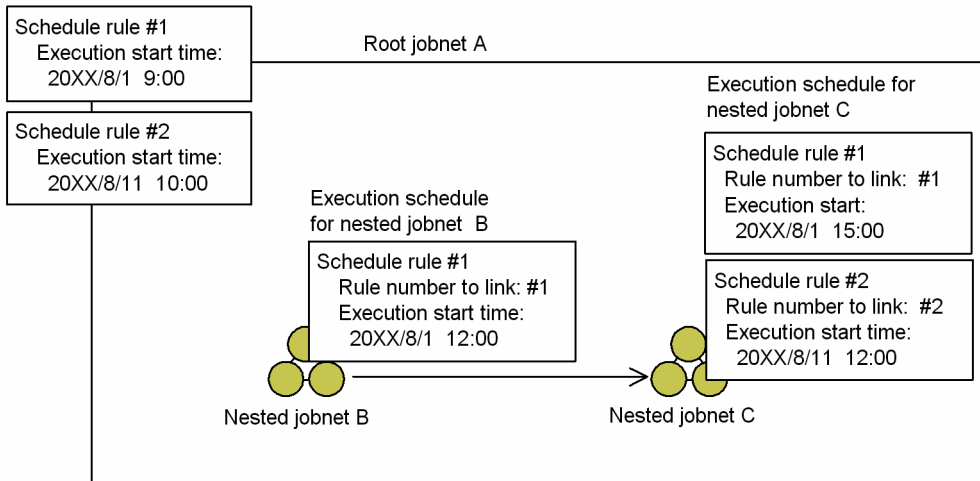
#### (2) Linking to the schedule rules of the root jobnet

You define schedule rules for a nested jobnet by linking them with the schedule rules of the root jobnet. When a schedule rule of the root jobnet comes into effect, the linked schedules of the nested jobnet also come into effect.

The following is an example of linking schedule rules.

Figure 3–6: Linking schedule rules of nested jobnets and root jobnets

Execution schedule of root jobnet A



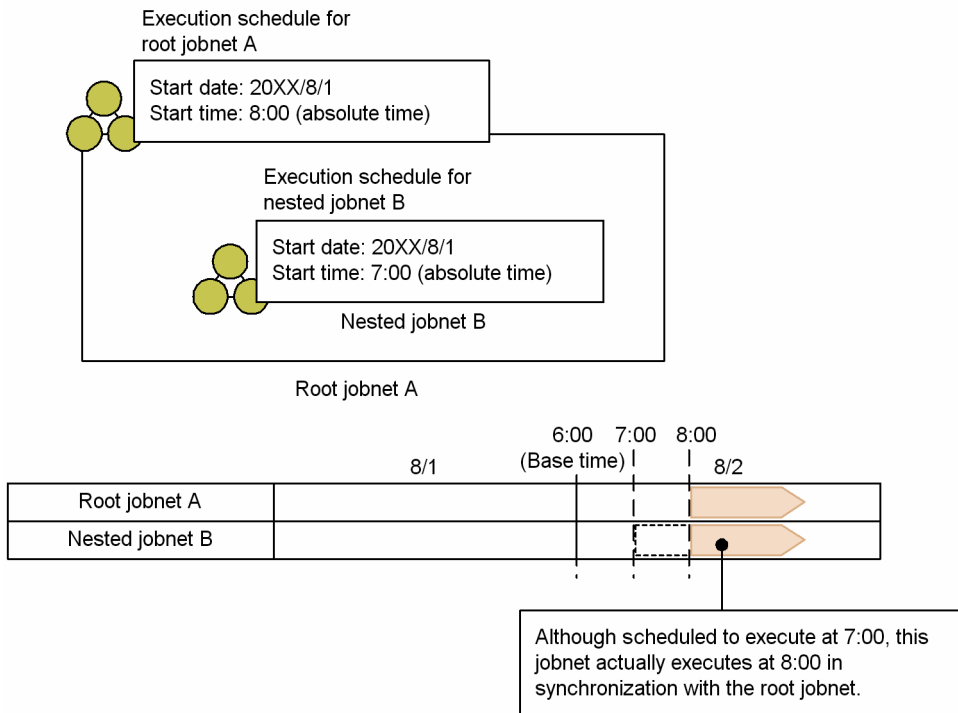
In this example, the schedule rule for Nested jobnet B is not linked to Schedule rule #2 of the root jobnet. This means that on August 11, when the root jobnet is executed according to Schedule rule #2, the Nested jobnet B is not executed.

Take note of the following when you define a schedule for a nested jobnet:

- A nested jobnet is not executed if there is no point at which the schedules of the upper-level jobnet and the nested jobnet overlap. If you copy a nested jobnet from another location, check the upper-level schedule before you define the schedule.
- For a nested jobnet, you can define a schedule that extends over two days. In such a case, specify an execution start time for the nested jobnet between 24:00 and 47:59. For example, assume that the execution start time of the root jobnet is 20XX/8/1 23:00, and you want the execution start time of the nested jobnet to be 2 a.m., on the next day. In this case, specify 20XX/8/1 26:00 as the execution start time. This gives the root jobnet and the nested jobnet the same execution start day. If you specify 20XX/8/2 2:00 as the execution start time for the nested jobnet, the nested jobnet will not execute because the execution start dates for the root jobnet and the nested jobnet are different.
- If you link multiple nested jobnet schedule rules that have the same execution start date to one root jobnet schedule rule, the schedule rule with the earliest start time takes effect. If the execution times of the schedule rules overlap, the schedule rule that has the lowest rule number overrides other schedules. Because other schedule rules are disabled, see [3.5.4 Executing the same jobnet several times a day \(defining cycle jobs\)](#) for information about executing multiple jobnets a day.

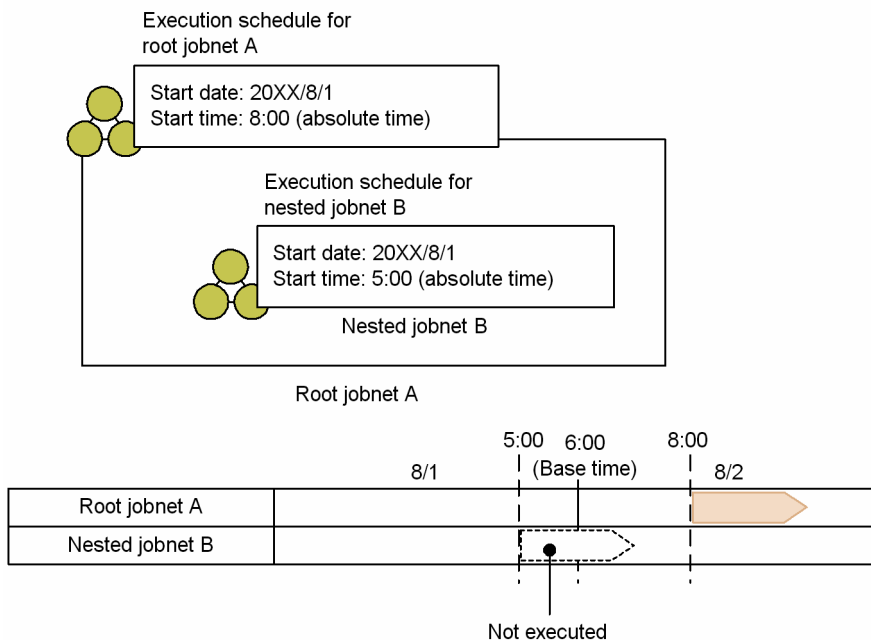
- If the start time of the nested jobnet is earlier than that of the root jobnet, and the start time is earlier than the base time, the root jobnet and the nested jobnet will be scheduled to execute on different days. As a result, the nested jobnet fails to execute. Two examples are shown below, one where the nested jobnet executes and one where it does not.

Figure 3–7: When the nested jobnet start time precedes the root jobnet start time (example where the nested jobnet executes)



In this example, the nested jobnet's start time (7:00) comes after the base time (6:00). This gives the root jobnet and the nested jobnet the same execution start day, so the nested jobnet will execute.

Figure 3–8: When the nested jobnet start time precedes the root jobnet start time (example where the nested jobnet does not execute)

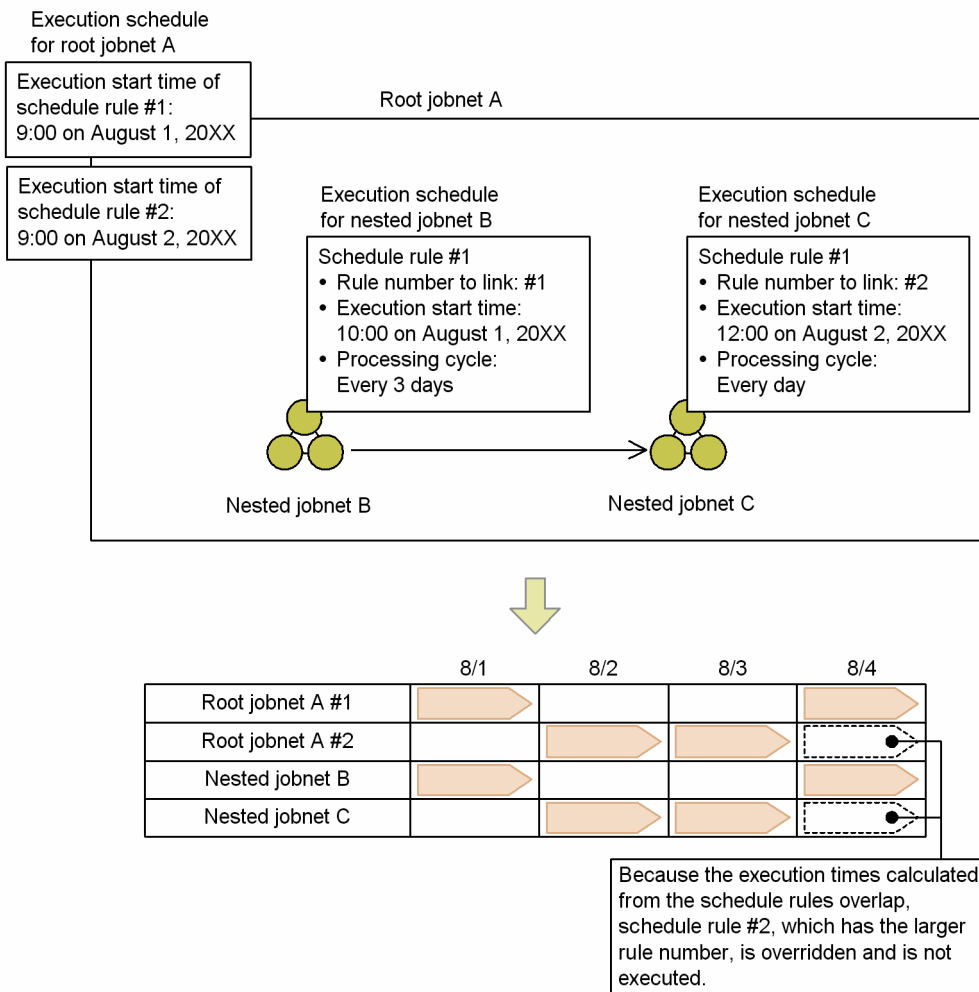


In this example, the nested jobnet's start time (5:00) precedes the base time (6:00). The nested jobnet will not execute because it has a different execution start date from the root jobnet.

- When multiple schedule rules have been defined for a root jobnet, schedules whose execution times overlap might be created, depending on the schedule rules. For the schedules whose execution times overlap, the schedule that has the smallest rule number overrides other schedules. If overridden schedule rules have been linked to nested jobnets, the schedules of the nested jobnets are also overridden.

The following figure shows an example of scheduling when execution times calculated from schedule rules overlap.

Figure 3–9: Example of scheduling when execution times calculated from schedule rules overlap



In this example, execution times of Root jobnet A on August 4 calculated from Schedule rules #1 and #2 overlap. In this case, Schedule rule #1, which is the schedule rule with the smaller rule number, overrides Schedule rule #2. Nested jobnet B linked to Schedule rule #1 is executed, but Nested jobnet C linked to overridden Schedule rule #2 is not executed on August 4.

### (3) Setting schedule options for nested jobnets

In the same way as for root jobnets, you can set schedule options for nested jobnets. For details about schedule options, see 3.3.2 *Defining a schedule* in the manual *JP1/Automatic Job Management System 3 Overview*.

#### (a) Refer to a calendar of another job group

A nested jobnet can also reference a calendar definition that is not of the upper jobnet.

A nested jobnet referencing a calendar definition that is not of the upper jobnet is executed when the execution date of the nested jobnet itself matches the execution date of the upper jobnet.

## (b) Exclusive schedule

If there are nested jobnets that are at the same level, you can prevent a nested jobnet from being executed when its execution date is the same as the execution date of another nested jobnet.

The scheduled execution of a nested jobnet that has exclusive execution enabled is canceled when both of the following conditions are met:

- The linked rule number specified in the schedule rule for one nested jobnet is also specified for another nested jobnet that is specified as an exclusive schedule of that jobnet.
- The execution dates calculated from the schedule rules in which the same linked rule number is specified are the same.

Note that if the execution dates calculated from the schedule rules in which different linked rule numbers are specified are the same, execution of the exclusive nested jobnets will not be canceled.

## 3.5.4 Executing the same jobnet several times a day (defining cycle jobs)

If you want to execute the same jobnet more than once and at regular intervals during the course of a day, define the jobnet using one of the following three methods:

- Method 1: Create a single jobnet, and then define multiple schedule rules.
- Method 2: Create a new jobnet for each start time.
- Method 3: Specify an interval control setting as a start condition for the jobnet.

### (1) Methods for defining the jobnet

Each of these methods is described in the following paragraphs. Each description uses, as an example, the creation of a jobnet that executes 24 times a day in one-hour intervals, between 7 a.m. on the current day and 6 a.m. the following day.

Method 1: Create multiple schedule rules

1. Create a single jobnet, and define the jobs to be executed.
2. Create 24 individual schedule rules with staggered start times of 07:00, 08:00, 09:00 and so on.

When you register the jobnet for execution, the jobnet is executed every hour according to the start times specified in the schedule rules.

When you use JP1/AJS3 - View to define a schedule rule, you can define a schedule that executes a jobnet at regular intervals until a time of your choosing.

Method 2: Create a new jobnet for each start time

1. Create a single jobnet.
2. Create a nested jobnet, and define the jobs to be executed.
3. Copy the nested jobnet you created in step 2, and create a total of 24 identical nested jobnets.
4. Set a different start time between 07:00 and 06:00 for each nested jobnet.

When you register the jobnet for execution, each nested jobnet is executed according to its respective start time.

Method 3: Specify an interval control setting as a start condition for the jobnet

1. Create a single jobnet, and define the jobs to be executed.
2. Create a start condition, and attach an execution interval control job.

3. In the Detailed Definition - [Interval Control] dialog box, in the **Waiting time** section, specify 60 (minutes). In the **Expire right after starting** section, click **Yes**.
4. Create a schedule rule, setting the execution start time to 7:00, and the valid range of the start condition to 24 times.

When you register this jobnet for execution, monitoring for the start condition starts at 7:00, which is the execution start time. The start condition is immediately satisfied and the first execution of the jobnet starts. Thereafter, the jobnet is executed every 60 minutes as indicated by the start condition.

## (2) Advantages and disadvantages of each method

The following table explains the advantages and disadvantages of the three methods.

Table 3–4: Advantages and disadvantages of each method of executing the same jobnet periodically

| Method   | Advantages  | Disadvantages  |
|----------|---|--|
| Method 1 | <ul style="list-style-type: none"> <li>• Defining jobnets is easy.</li> <li>• Jobnets have a simple configuration.</li> <li>• When you define a jobnet as a root jobnet, you can enable concurrent execution or schedule skip.</li> </ul>   | <ul style="list-style-type: none"> <li>• To change the schedule, you must change all the schedule rules.</li> </ul>  |
| Method 2 | <ul style="list-style-type: none"> <li>• You can apply this method to any type of jobnet.</li> <li>• You can check which instance of the jobnet is currently executing using the jobnet monitor.</li> <li>• You can rerun a job or cancel execution of a job from within the jobnet monitor.</li> </ul> | <ul style="list-style-type: none"> <li>• You need to create one nested jobnet each time you want to execute the jobnet.</li> <li>• You need to define a schedule for every nested jobnet.</li> <li>• To change a schedule rule, you must change the schedule rules for every nested jobnet.</li> <li>• Changing the process that is executed is time consuming.</li> </ul> |
| Method 3 | <ul style="list-style-type: none"> <li>• Jobnets have a simple configuration.</li> </ul>  | <ul style="list-style-type: none"> <li>• You can only use this method with a root jobnet.</li> </ul>   |

### 3.5.5 Shifting the scheduled execution date forward or back based on a calculated schedule (Schedule by days from start)

This subsection describes the settings required to shift the execution date of work tasks. The date is shifted by days, to a date before or after the scheduled execution date, which is determined by the calendar definition and schedule rules.

You define these settings using the schedule by days from start function. For details about the schedule by days from start function, see 3.3.2 *Defining a schedule* in the manual *JP1/Automatic Job Management System 3 Overview*.

The following describes how to use the schedule by days from start function, with an example jobnet that performs the salary calculation processing before the salary payment day.

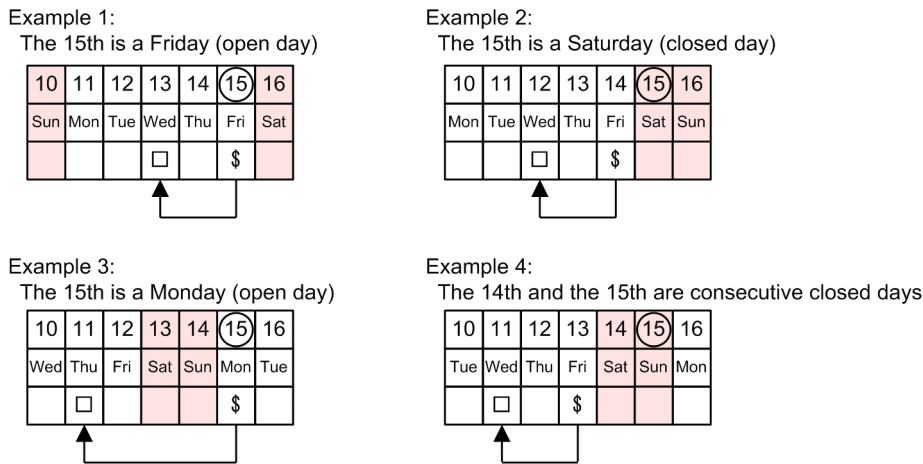
#### (1) Performing salary calculation processing two open days before the salary payment day

The following describes an example of a jobnet that performs the salary calculation processing two open days before the salary payment day (the 15th of every month). If the 15th is a closed day, the salary payment day must be shifted to the nearest open day before the 15th.

The following figure shows an example of performing the salary calculation processing two open days before the salary payment day.



Figure 3–10: Example of executing the salary calculation processing two open days before the salary payment day



Legend:  
 \$ : Salary payment date  
 □ : Scheduled execution date for salary calculation process

- Example 1:**  
 If the 15th is Friday (open day), the jobnet for salary calculation processing is executed on the 13th (two open days before the 15th).
- Example 2:**  
 If the 15th is Saturday (closed day), the salary payment day is shifted to the 14th (nearest open day before the 15th). Therefore, the jobnet for salary calculation processing is executed on the 12th (two open days before the 14th).
- Example 3:**  
 If the 15th is Monday (open day) and both the 13th and 14th are closed days, the jobnet for salary calculation processing is executed on the 11th (two open days before the 15th).
- Example 4:**  
 If the 15th is Sunday (closed day) and the 14th is also a closed day, the salary payment day is shifted to the 13th, which is the nearest open day before the 15th. Therefore, the jobnet for salary calculation processing is executed on the 11th (two open days before the 13th).

To perform the operations above, use the schedule by days from start function, based on open days.

When you use this function based on open days, the scheduled execution day is shifted back by n open days from the base date determined by the calendar definition and substitute schedule for closed days. Therefore, you can set the jobnet for salary calculation processing to be executed on a date two open days before the salary payment day.

To set the schedule shown in the figure by using the schedule by days from start function, define the schedule rule as follows:

**Definition**

- Execution start date: 15th (absolute day)
- Processing cycle: 1 month
- Substitute schedule in case of a closed day job: Execute on previous open day
- Schedule by days from start: Execute 2 open days before start date

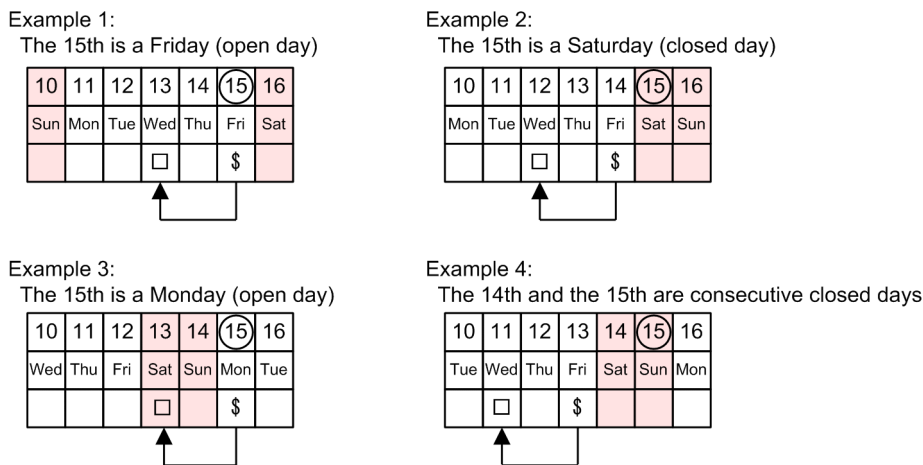
When you register this definition for execution, first the start date (salary payment date) is calculated based on the settings you specified for the execution start date, the processing cycle, and the substitute schedule in case it falls on a closed day. Accordingly, the actual scheduled execution date is set to a date two open days before the start date.

## (2) Performing the salary calculation processing two days before the salary payment day

The following describes an example of a jobnet that performs the salary calculation processing two days before the salary payment day (the 15th of every month). If the 15th is a closed day, the salary payment day must be shifted to the nearest open day before the 15th. The salary calculation processing can be performed on closed days and open days.

The following figure shows an example of performing the salary calculation processing two days before the salary payment day.

Figure 3–11: Example of executing the salary calculation processing two days before the salary payment day



Legend:  
\$ : Salary payment date  
□ : Scheduled execution date for salary calculation process

### Example 1:

If the 15th is Friday (open day), the jobnet for salary calculation processing is executed on the 13th (two days before the 15th).

### Example 2:

If the 15th is Saturday (closed day), the salary payment day is shifted to the 14th (nearest open day before the 15th). Therefore, the jobnet for salary calculation processing is executed on the 12th (two days before the 14th).

### Example 3:

If the 15th is Monday (open day) and both the 13th and 14th are closed days, the jobnet for salary calculation processing is executed on the 13th (two days before the 15th).

### Example 4:

If both the 14th and 15th are closed days, the salary payment day is shifted to the 13th (nearest open day before the 15th). Therefore, the jobnet for salary calculation processing is executed on the 11th (two days before the 13th).

To perform the operations above, use the schedule by days from start function, based on any day.

When you use this function based on any day, the scheduled execution day is shifted back by n days from the base date determined by the calendar definition and substitute schedule for closed days. Therefore, you can set the jobnet for salary calculation processing to be executed on a date two days before the salary payment day.

To set the schedule shown in the figure by using the schedule by days from start function, define the schedule rule as follows:

**Definition**

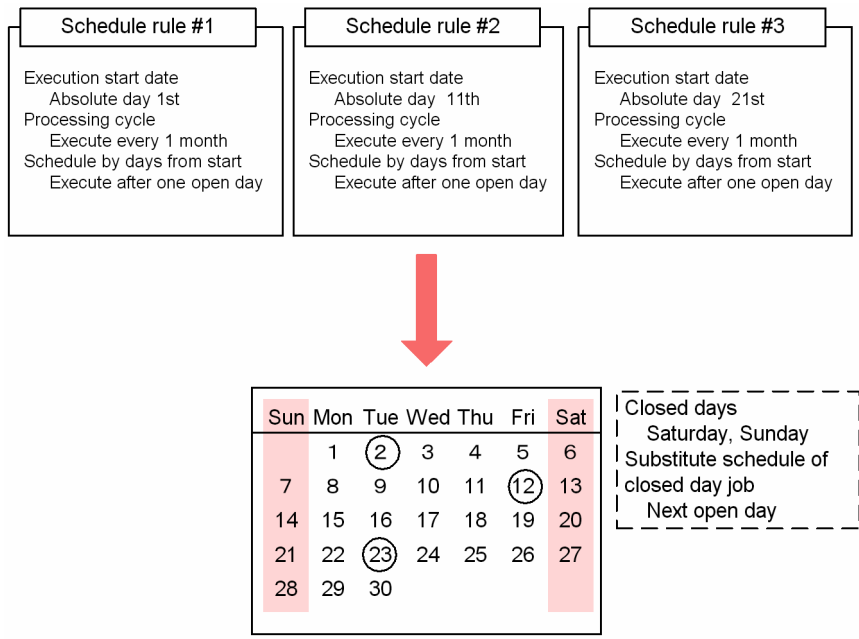
- Execution start date: 15th (absolute day)
- Processing cycle: 1 month
- Substitute schedule in case of a closed day job: Execute on previous open day
- Schedule by days from start: Execute 2 days before start date

When you register this definition for execution, first the start date (salary payment date) is calculated based on the settings you specified for the execution start date, the processing cycle, and the substitute schedule in case it falls on a closed day. Accordingly, the actual scheduled execution date is set to a date two days before the start date.

### 3.5.6 Defining a schedule that has different behavior at different times of the month

This subsection describes how to define a schedule that causes a jobnet to execute in the first ten-day period, the second ten-day period, and the last ten-day period in a month. You can define this type of schedule by creating schedule rules that define execution start dates in the first ten-day period, second ten-day period and last ten-day period of the month. For example, to execute a process on the last open day of each ten-day period in the month, specify the 10th, the 20th and the final day of the month as the execution start dates. Use the schedule by days from start function if you want the execution dates to shift from these dates when required. The following example illustrates the use of schedule by days from start to execute a process once every ten-days.

Figure 3–12: Example of a schedule that executes a process every ten-days



This schedule is structured so that the process is executed on the open day that follows the first open day in each ten-day period in the month. The first day in the last ten-day period is the 21st. However, because the 21st is Sunday and therefore

a closed day, the execution date shifts to the next business day, in this case the 22nd. The **Schedule by days from start** setting then shifts the execution date back by one day. As a result, the scheduled execution date is calculated as the 23rd.

### 3.5.7 Defining a different calendar for each application

Sometimes a certain application or a particular department will have different open days. In such a case, you can define different calendars for different applications.

As an example, consider a case in which the Tokyo head office, the Osaka branch office, and the Nagoya branch office all execute a certain application, but run the application on different days. The respective application calendars are as follows.

- Tokyo head office: Execute daily
- Osaka branch office: Execute Monday through Saturday
- Nagoya branch office: Execute Monday through Friday

There are two ways of structuring the application.

- Group all jobnets with the same application calendar in job groups.
- Create the application within one job group, and refer to calendars of other job groups.

These methods are explained in detail below.

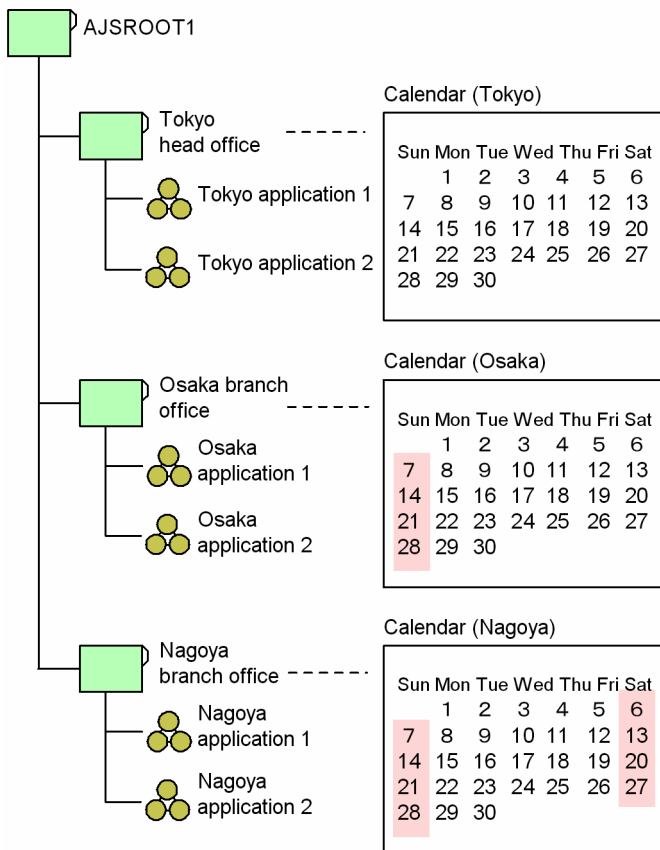
#### (1) Grouping jobnets with the same application calendar in job groups

This method is best used for applications that meet the following conditions:

- There are multiple application groups, but no dependent relationships between the application groups.
- There is a system administrator responsible for each application group.
- Multiple application groups are managed by one manager.

The following figure shows an example of the application configuration.

Figure 3–13: Grouping jobnets with the same application calendar in job groups



An overview of the method used to set the schedule is given below.

1. Create job groups for Tokyo, Osaka, and Nagoya under the scheduler service.
2. Define jobnets under each job group.
3. Define a calendar for each job group.
4. Register the jobnets for execution.

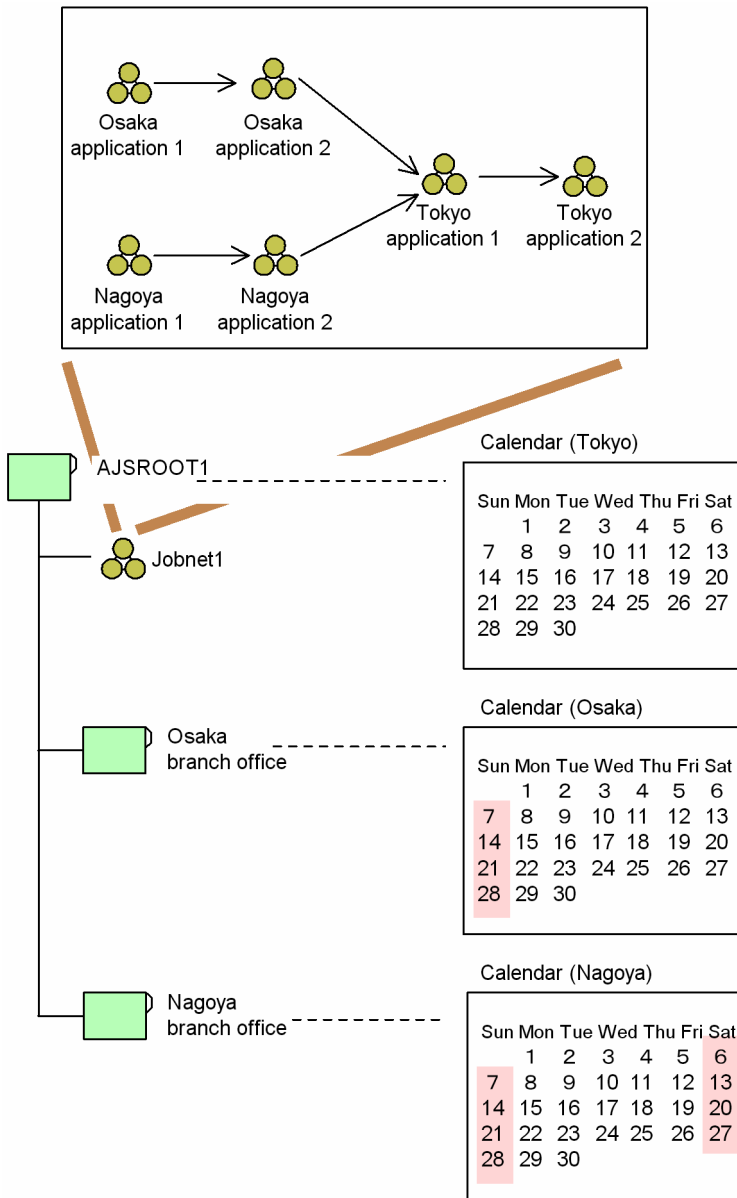
## (2) Creating the application within one job group, and referencing calendars defined for other job groups

This method is best used for applications that meet the following conditions:

- There are multiple application groups, and dependent relationships between the application groups.
- One system administrator manages all applications.

The following figure shows an example of the application configuration.

Figure 3–14: Referring to a calendar of another job group



An overview of the method used to define the schedule is given below.

1. Create a single jobnet (jobnet 1) under the scheduler service.
2. Define the applications to be executed at Tokyo, Osaka, and Nagoya as nested jobnets within the root jobnet.
3. Specify the calendar of the Tokyo head office in the scheduler service.
4. Create job groups for Osaka and Nagoya.
5. Define calendars for the Osaka and Nagoya job groups.  
Define only calendars for these job groups; do not create jobnets for them.
6. Set the nested jobnets you want to execute at each branch to refer to the calendar defined for the respective job group.  
In the Schedule Settings dialog box, select **Refer to a calendar of another job group**. Set Osaka application 1 and Osaka application 2 to refer to the calendar of the Osaka branch job group. Set Nagoya application 1 and Nagoya application 2 to refer to the calendar of the Nagoya branch job group.
7. Register the root jobnet for execution.

With this method, you can monitor all of the jobnets in a single monitor window.

### (3) Examples of calendar work tasks

This subsection presents some examples that show how calendars are applied.

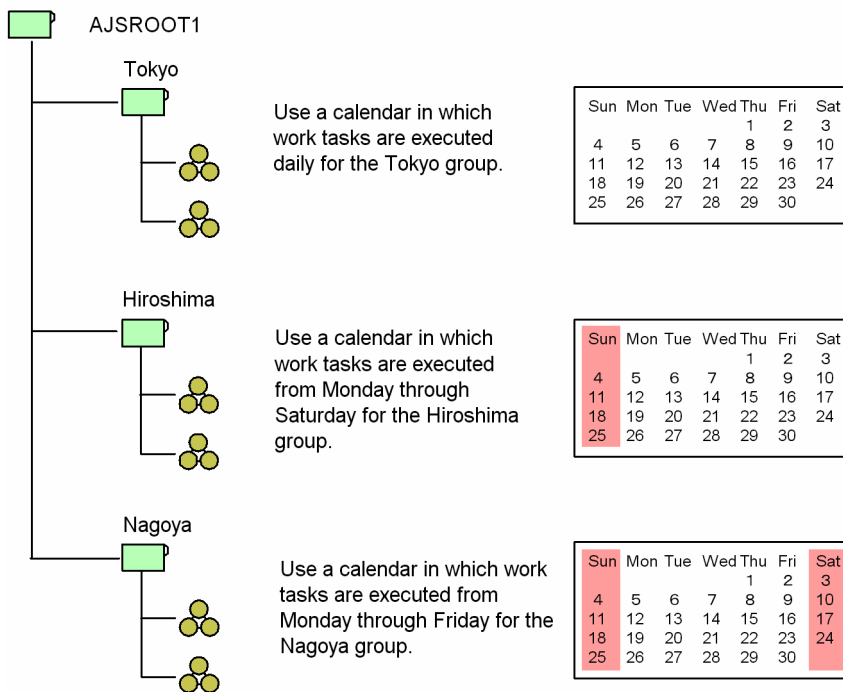
#### Example 1

In the first example we will explain how to set a calendar premised on the following conditions:

- There are multiple work task groups, but there is no interdependency among the information in the individual calendars.
- Each work task group is controlled by its own system administrator.
- The multiple work task groups are managed by a single work task manager host.

Based on these premises, we will create a hierarchy for the work task group and define calendars, as shown in the following figure.

Figure 3–15: Calendar work task example 1



When working from the monitor screen with an example like this, you can only check the execution schedule and results of work tasks under a single work task group (for example the *Tokyo* group) at one time. In a situation where multiple system administrators are managing individual work task groups, this arrangement makes it easy for the responsible system administrators to check the schedules and results for the work tasks they are concerned with.

#### Example 2

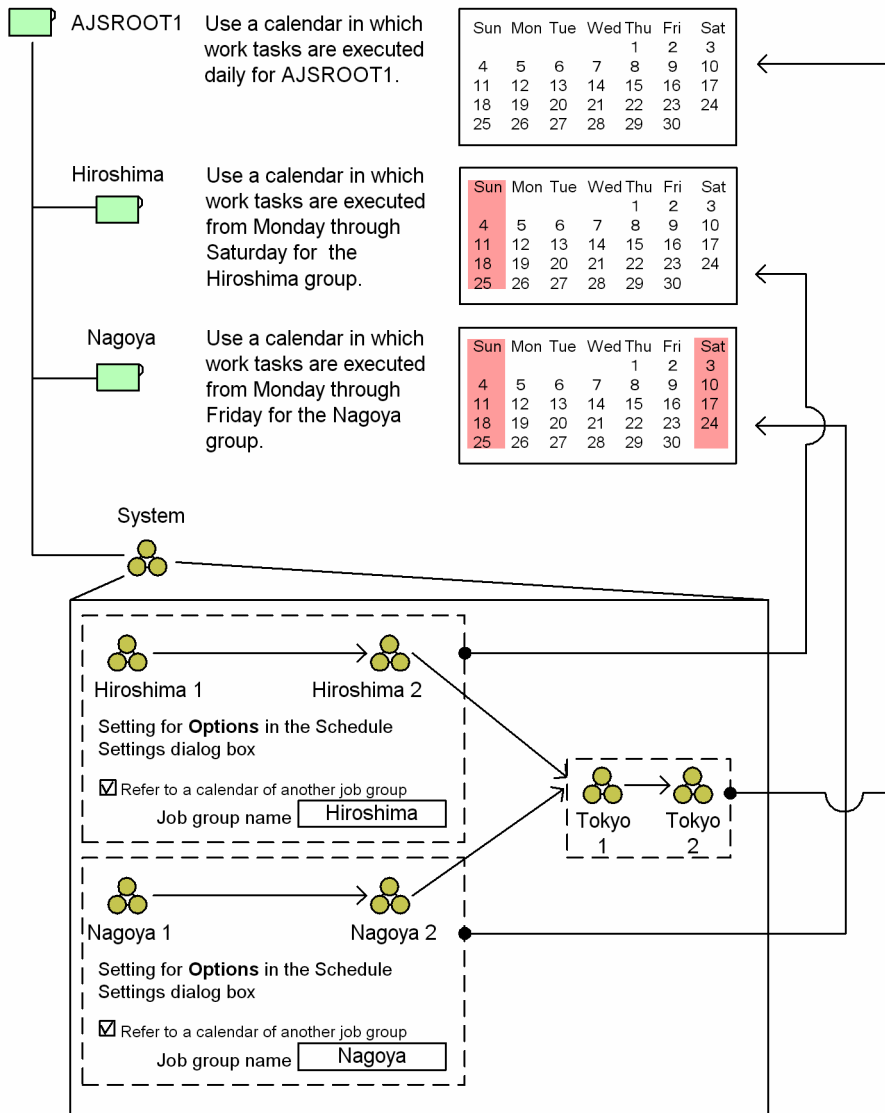
We will explain the setting of a calendar premised on the following conditions.

- There are multiple work task groups and the information in their calendars is interdependent.

- All of the work tasks are managed by a single system administrator.

Taking these conditions as the premises, we will create a hierarchy for the work task group and define a calendar, as shown in the following figure.

Figure 3–16: Calendar work task example 2



Legend:

: Closed day

When you make the settings in the figure above, the individual work tasks are executed as follows.

#### Tokyo work tasks

The work tasks reference a calendar for daily execution so they are executed every day.

#### Hiroshima work tasks

The work tasks reference a calendar in which only Sunday is a closed day, so they are executed from Monday through Saturday.

#### Nagoya work tasks

These work tasks reference a calendar in which Saturdays and Sundays are closed days, so they are executed from Monday through Friday.



In this example, if you check the group AJSROOT1 on the monitor screen, you can check the schedules and results of all the work tasks under this group. This makes it easy for a single system administrator to manage all the work tasks.

# 4

## Execution Registration Method Considerations

When a jobnet with a schedule rule is registered for execution, the jobnet is added to the schedule in JP1/AJS3.

This chapter describes the methods you can use to register jobnets for execution in JP1/AJS3.

## 4.1 Jobnet execution registration methods

---

In JP1/AJS3, you can use the following three methods to register jobnets for execution:

- Planned execution registration
- Fixed execution registration
- Immediate execution registration

For details about each mode, see *4.1.1 Methods of registering a jobnet for execution* in the manual *JP1/Automatic Job Management System 3 Overview*.

Take into account the characteristics of each method, and choose the one that best suits the way you use the system.

### 4.1.1 Planned execution registration

A jobnet registered for planned execution is scheduled and executed according to the calendar information and schedule rules set for the jobnet. The calculated schedule is treated as a dummy schedule (simulated execution schedule).

#### (1) Characteristics of planned execution registration

The characteristics of planned execution registration are as follows:

##### (a) Execution timing

The jobnet is scheduled and executed according to the calendar information and the schedule rule set for the jobnet. When execution of a jobnet starts, the next scheduled execution time is fixed in the schedule.

##### (b) Temporarily rescheduling the jobnet

You can make a temporary change only to the next scheduled execution of a jobnet registered for planned execution. Further rescheduling is not possible because the schedule is simulated.

##### (c) Modifying the calendar information or schedule rules

If you change the calendar information or redefine the jobnet's schedule rule, JP1/AJS3 reschedules the jobnet based on the new information. This allows you to modify calendar information and schedule rule without canceling registration of the jobnet, such as when open or closed days need to be redefined for the next business year, or when you change a schedule rule.

#### (2) Recommended uses

We recommend use of planned execution registration in situations like the following:

- When changes to calendar information or schedule rule might be necessary in the future
- When you want to execute the jobnet a specific number of times per day in response to a trigger such as a file being updated or a specific action by the user (this requires a start condition to be set)
- With jobnets that never need to be temporarily rescheduled

## 4.1.2 Fixed execution registration

A jobnet registered for fixed execution is scheduled and executed according to the calendar information and the schedule rule set for the jobnet. The schedule based on the specified period or generation count is calculated and fixed as soon as the jobnet is registered for execution.

### (1) Characteristics of fixed execution registration

The characteristics of fixed execution registration are as follows:

#### (a) Execution timing

The jobnet is scheduled and executed according to the calendar information and the schedule rule set for the jobnet.

#### (b) Temporarily rescheduling the jobnet

Temporary changes can be made to a fixed schedule scheduled for a specified period or times. You can also add execution dates.

#### (c) Modifying the calendar information or schedule rules

Changes to the calendar information or jobnet's schedule rule do not take effect until the jobnet has run for the specified period or number of generations. To apply changes made to calendar information or schedule rules to the schedule, you must cancel and then re-register the jobnet.

### (2) Recommended uses

We recommend use of fixed execution registration in situations like the following:

- With jobnets that run for a set period or for a set times
- With jobnets that have a fixed schedule that may require a temporary change, addition, or prohibit execution

## 4.1.3 Immediate execution registration

A jobnet registered for immediate execution is executed once only, as soon as it is registered, regardless of the calendar information and the schedule rule set for the jobnet.

### (1) Characteristics of immediate execution registration

The characteristics of immediate execution registration are as follows:

#### (a) Execution timing

The jobnet is executed immediately upon registration. Any schedule rules set for the jobnet are ignored.

#### (b) Temporarily rescheduling the jobnet

You cannot temporarily reschedule a jobnet registered for immediate execution, because the jobnet has no execution schedule.

#### (c) Modifying the calendar information or schedule rules

Not applicable because the calendar information and jobnet schedule rule are not referenced.

## (2) Recommended uses

We recommend use of immediate execution registration in situations like the following:

- With jobnets that are started manually or by command
- With jobnets executed from within an application
- With jobnets whose execution is triggered by a command initiated when a file transfer application program (e.g. JP1/FTP) ends normally
- With jobnets whose execution is triggered by a JP1/IM automated action
- When running a consistency test on the processing flow for jobnets with defined schedules

# 5

## Monitoring Method Considerations

This chapter describes the considerations for monitoring work tasks by using JP1/AJS - View, a Web GUI (Job Portal), or a user application.

## 5.1 Monitoring methods in JP1/AJS3 - View

---

This section describes the considerations necessary for using JP1/AJS3 - View to monitor work tasks.

For details about how to plan the JP1/AJS3 - View environment, see *4.5 Environment settings for JP1/AJS3 - View* in the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*.

### 5.1.1 Planning which work tasks to monitor in the Summary Monitor window

By designating selected jobnets as monitoring targets in the JP1/AJS3 - View window (Summary Monitor window), you can monitor the execution status of the jobnets, including the execution progress and the statuses of subordinate jobs. You can also view a simulation of the jobnet's end time, calculated from the jobnet's execution times in the past. This can be useful when any delay in starting or ending the work task would have a major effect on the execution schedule for subsequent work tasks.

JP1/AJS3 - View also allows you to centrally monitor jobnets that are distributed over different jobnet hierarchies or are governed by different scheduler services.

### 5.1.2 Jobnet delay monitoring considerations

JP1/AJS3 provides the following methods for detecting delays in jobnet execution.

#### (1) Monitoring by date and time specification

With this method, you specify a date and time, and JP1/AJS3 monitors whether a jobnet starts and ends at the scheduled time. JP1/AJS3 provides two methods of monitoring for delays by date and time specification: *Monitor start delay* and *Monitor end delay*.

Use these methods when a delay in starting or ending the jobnet will have a major effect on the execution schedule for subsequent jobnets.

##### Monitor start delay

Set a date and time for judging when a start is delayed. If the jobnet starts within a certain time after the scheduled start time, it is deemed normal. If the jobnet has still not executed when the grace period elapses, some abnormality is assumed to have occurred.

##### Monitor end delay

Set a time for judging when completion is delayed. If the jobnet ends within a certain time after the scheduled start time, it is deemed normal. If jobnet execution has still not ended when the grace period elapses, some abnormality is assumed to have occurred.

You can designate both a start delay and an end delay, or just one or the other.

#### (2) Monitoring by time-required-for-execution

You can also monitor for delays based on the time required for execution of the jobnet.

If the time set is exceeded, the color of the work task icon changes, and an event is issued to report the delay.

### 5.1.3 End delay monitoring based on time-required-for-execution

You can also monitor for end delays based on the time-required-for-execution of a job. In this case, JP1/AJS3 monitors how long the job takes to execute on the manager host, which may differ from how long it takes on the agent host. This discrepancy can affect whether an end delay is detected.

Timeout monitoring, on the other hand, is based on how long the job is active on the agent host. For this reason, there is a time difference between timeout detection and the detection of an end delay.

The following are examples of situations where a job has a different time-required-for-execution on the manager host and agent host:

- The job is started on the agent host while the scheduler service is stopped.  
If a job begins executing on the agent host while the scheduler service is stopped (that is, the scheduler service stops before the manager host receives notification from the agent host that the job has started), the job does not enter *Now running* status until the scheduler service restarts. Here, the job will have a longer execution time on the agent host than on the manager host. For example, suppose the job's time-required-for-execution is 10 minutes, and it has been executing for 20 minutes on the agent host. In this case, if the job finishes five minutes after the scheduler service restarts, an end delay will not be detected.
- The job ends on the agent host while the scheduler service is stopped.  
If a job that was already running on the agent host ends while the scheduler service is stopped (that is, the scheduler service stops before the manager host receives notification from the agent host that the job has ended), the job does not acquire an end status until the scheduler service restarts. Here, the job will have a shorter execution time on the agent host than on the manager host. For example, suppose the job's time-required-for-execution is 10 minutes, and it ends on the agent host after five minutes. Because the job is considered to be running the entire time the scheduler service was stopped, an end delay will be detected if the scheduler service is stopped for longer than 10 minutes.
- The job starts and ends on the agent host while the scheduler service is stopped.  
If a job starts and finishes on the agent host while the scheduler service is stopped (that is, the scheduler service stops before the manager host receives notification from the agent host that the job has started and ended), the job enters *Now running* status then ends as soon as the scheduler service restarts. Here, the job execution time on the manager host will be zero minutes. For example, suppose the job's time-required-for-execution is 10 minutes, and it took 20 minutes to execute on the agent host. In this case, an end delay will not be detected because the job's execution time on the manager host is zero minutes.

#### Supplementary notes

- When an end delay is detected for a job, you can check the job's execution time on the agent host by using the `ajsshow` command. Execute the command with the format specifiers `%V` and `%Q` to acquire the execution start and end times of the job. For the command syntax, see *ajsshow* in *3. Commands Used for Normal Operations* in the manual *JP1/Automatic Job Management System 3 Command Reference*.
- If you specify `yes` in the `JOBDELAYWARNMSG` environment setting parameter of the scheduler service, the message `KAVS0249-W The scheduler services stopped before execution of the job began .` is output to the integrated trace log when a job being monitored for an end delay starts executing while the scheduler service is stopped. You can identify the job name and execution ID from the content of the message. To verify an end delay based on the time-required-for-execution of the job on the agent host, use the `ajsshow` command to acquire the execution start and end times of the job identified in the message `KAVS0249-W`. For information about the environment setting parameter `JOBDELAYWARNMSG`, see *20.4 Setting up the scheduler service environment* in the *JP1/Automatic Job Management System 3 Configuration Guide*.



## 5.2 Considerations for monitoring by using a Web GUI (Job Portal)

---

This section describes the considerations for monitoring jobnets by using a Web GUI (Job Portal).

For details about the considerations for Web GUI (Job Portal) environment setup, see *4.6 Environment settings for JP1/AJS3 - Web Console* in the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*.

### 5.2.1 Considering what to monitor in the Dashboard screen

In the Dashboard screen, you can monitor, for example, the progress of jobnets, the number of subordinate jobs in each status, and the delay status. The Dashboard screen is suitable for checking the jobnet execution results of the previous day, or for monitoring the progress of jobnet execution today.

Before you can use the Dashboard screen to monitor jobnets, you must register the jobnets to be monitored by using a Web GUI (Job Portal). Therefore, in advance, you must consider the jobnets that you want to monitor with the Web GUI (Job Portal).

## 5.3 Considerations for monitoring by using an API-based user application

---

When you monitor a jobnet by using a user application, you can select the monitoring-target units according to the work tasks of the users who use the user application. However, we do not recommend that you develop a user application that can display a large number of units as monitoring targets or that can perform operations on multiple units at one time. If you develop such a user application, the processing of the user application might become slow or the load on JP1/AJS3 - Manager might increase.

Before you develop a user application, consider the following issues:

- Number and level of units to be monitored
- Work tasks of the users who use the user application

For details of such considerations and notes regarding implementation of a user application by using an API, see 6. *Overview of APIs* in the manual *JP1/Automatic Job Management System 3 Command Reference*.

# 6

## Access Permission Considerations

This chapter describes matters you must consider when setting the access permissions for jobnets and for associating JP1 users and OS users.

## 6.1 Steps for considering work task access permissions

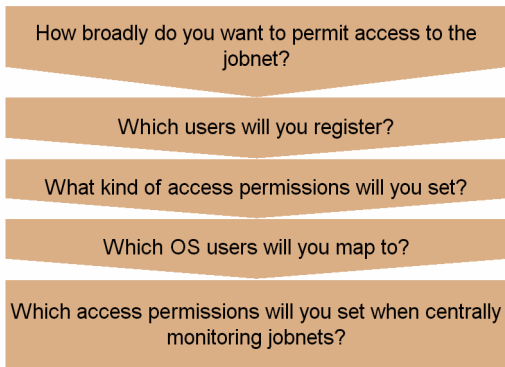
---

This section explains the considerations involved when setting work task access permissions for individual users. By granting JP1/AJS3 users the permission to execute processing only, or the permission to view processing results only, you can prevent erroneous operations.

The figure below shows the steps when considering work task access permissions.

Use the user management functionality of JP1/Base (user authentication, user mapping), to set and manage work task access permissions. For details on the user management functionality, see the *JP1/Base User's Guide*.

Figure 6–1: Steps for considering work task access permissions



## 6.2 Ranges for setting access permissions

JP1/AJS3 manages access permissions, such as the permission to execute processing and the permission to change processing details, based on the unique user names of the JP1 users.

Use the user authentication functionality provided by JP1/Base to register and manage the JP1 users. For the purposes of the user authentication functionality, a JP1/Base host that manages the access permissions of the JP1 users is called an *authentication server*. Once you have specified a JP1/Base host among those on the network that is to be the authentication server, you can decide the range of hosts where that authentication server manages the access permissions (the user authentication bloc). You must set user authentication blocs when you introduce JP1/AJS3.

Consult the following table when considering how to set user authentication blocs.

Table 6–1: Number of authentication blocs and merits/demerits

| Number of authentication blocs                         | Merits and demerits   |
|--|---|
| Only one user authentication bloc is set in the system | The system administrator can manage all the JP1 users centrally, and not much time and effort are taken up with registration and changes.   |
| Multiple authentication blocs are set in the system    | The system administrator has to manage a number of JP1 users equivalent to the number of authentication blocs set. Some time and effort is expended managing JP1 user registration, changes, and so on, and login authentications are conducted at each of the authentication servers. However, since the individual authentication servers are independent, the resilience of the system as a whole is good. |

You can set two authentication servers within a single user authentication bloc. The authentication server in normal use is called the *primary authentication server*, while another that serves as a backup and is used in the event of trouble is called the *secondary authentication server*. If you set only one authentication server in a user authentication bloc, there is the risk that if it is not possible to make contact with the authentication server for any reason - for example because the authentication server will not start or a communication fault has occurred - it will not be possible to execute jobs or remote commands, and work tasks will stop.

If you set two authentication servers, even if trouble occurs at the one used under normal circumstances (the primary authentication server), you can still execute jobs and remote commands at the backup authentication server (secondary authentication server). Even if there is only one authentication bloc in the system, setting two authentication servers will increase the resilience of the system. Use two authentication servers if you consider it necessary.

If you set primary and secondary authentication servers, make the settings at both the same by copying the JP1 users, JP1 resource group and other information from the primary authentication server to the secondary authentication server. If the settings are not the same, an authentication error will occur when you switch servers.

For details on the settings at the primary and secondary authentication servers, see the *JP1/Base User's Guide*.

For details on how to specify the authentication server when primary and secondary authentication servers are set, consult the following references.

- For a Windows host running JP1/AJS3 - Manager:  
See 3.1.1 *Setting up JP1/Base* in the *JP1/Automatic Job Management System 3 Configuration Guide*.
- For a Windows host running JP1/AJS3 - Agent:  
See 3.2.1 *Setting up JP1/Base* in the *JP1/Automatic Job Management System 3 Configuration Guide*.
- For a UNIX host running JP1/AJS3 - Manager:  
See 13.1.1 *Setting up JP1/Base* in the *JP1/Automatic Job Management System 3 Configuration Guide*.
- For a UNIX host running JP1/AJS3 - Agent:  
See 13.2.1 *Setting up JP1/Base* in the *JP1/Automatic Job Management System 3 Configuration Guide*.

## 6.3 Setting registered users

When you have settled the range in which to manage user access permissions and the host that will manage access permissions, you must register the JP1 users at JP1/AJS3. Register JP1 users according to their purpose, and set the appropriate permissions for each JP1 user.

The following table gives an example of the registration of JP1 users.

Table 6–2: Example of registration of JP1 users

| JP1 User name | Purpose   |
|---------------|---|
| jpladmin      | <p>Used to manage the entire system.</p> <p>This user has administrator's permissions for JP1/AJS3, JP1/IM, and JP1/Base set as defaults. The following permissions are set:</p> <p>jpladmin: *=JP1_AJS_Admin, JP1_JPQ_Admin, JP1_Console_Admin</p> <p>JP1_AJS_Admin : Administrator's permission for JP1/AJS3</p> <p>JP1_JPQ_Admin : Administrator's permission for JP1/AJS3 job execution environment, queues, etc., and Administrator's permission for agent management information</p> <p>JP1_Console_Admin : Administrator's permission for JP1/IM</p> <p>We recommend that JP1 users who perform normal operations such as defining or executing jobnets use jplmngnr, not jpladmin.</p>  |
| jplmngnr      | <p>Used to manage work tasks (user who is a work task administrator) and execute processing (user for executing jobs).</p> <p>Set for this user the permission to refresh, execute and monitor all work tasks. Also set the permission that enables execution of processing that - owing to the specifications of the OS or program - cannot be executed without an administrator's permission or superuser permission. The following is an example of a permission setting.</p> <p>jplmngnr: *=JP1_AJS_Manager, JP1_JPQ_Operator</p> <p>JP1_AJS_Manager : permission to edit and execute work tasks</p> <p>JP1_JPQ_Operator : permission to operate job execution environments, queues, etc., permission to operate jobs, and permission to operate agent management information</p> |
| jpluser       | <p>Used to execute processing (user who executes jobs)</p> <p>To ensure efficient operation management, standardize the users who execute jobs at the work task manager host. The following is an example of a permission setting.</p> <p>jpluser: *=JP1_AJS_Manager, JP1_JPQ_Operator</p> <p>JP1_AJS_Manager : permission to edit and execute work tasks</p> <p>JP1_JPQ_Operator : permission to operate job execution environments, queues, etc., permission to operate jobs, and permission to operate agent management information</p>  |
| jpllope       | <p>Used to monitor work tasks and to rerun them manually. This user is intended for operators, allowing them to monitor work tasks and perform manual operations. The following is an example of a permission setting.</p> <p>jpllope: *=JP1_AJS_Operator</p> <p>JP1_AJS_Operator : permission to execute work tasks, permission to view work tasks</p>   |
| jplguest      | <p>Used to monitor work tasks only. This is a work task monitoring user for operators. The following is an example of a permission setting.</p> <p>jplguest: *=JP1_AJS_Guest</p> <p>JP1_AJS_Guest : permission to reference work tasks</p>  |
| root          | <p>Used to monitor JP1/AJS3 job execution environments, queues, etc.</p> <p>Note that when the jpqxxx command is executed from the command prompt, the OS user that executes the command must be registered as a JP1 user. The JP1 user called <i>root</i> explained in this example means the root user of the OS (UNIX). (When using Windows, the OS user with administrator permissions must be registered as the JP1 user <i>Administrator</i>.)</p> <p>The following is an example of a permission setting.</p> <p>root: *=JP1_JPQ_Admin</p> <p>JP1_JPQ_Admin : permission to manage job execution environments, queues, etc.</p>  |

For details on the permissions that can be set for JP1 users, see [6.4.1\(2\) Determining JP1 permission levels](#).

You can omit registration of JP1 users but you need to register such JP1 users to execute jobs.

If you omit the registration of JP1 users and set access permissions as described in [6.4 Setting access permissions](#), JP1/AJS3 assumes that JP1 users having the same names as the OS users are registered.

## 6.4 Setting access permissions

---

This section explains the access permissions assigned to JP1 users.

### 6.4.1 Access permission considerations

When you have registered the required JP1 users, you must decide what access permissions (JP1 permission levels) to grant to these JP1 users in regard to which groups (JP1 resource groups).

#### (1) Determining the JP1 resource groups to be defined

The access permissions for JP1/AJS3 units are managed in groups known as *JP1 resource groups*. You can define any name you like for a JP1 resource group. For example, if access permissions are grouped by company department, you can assign corresponding names such as `purchasingdep` or `personneldep`.

Access permissions can be set for each JP1 resource group. The set access permissions are associated with JP1 users. Each JP1 user is allowed to operate on the units in the associated JP1 resource group only within the set range of access permissions.

A unit that does not belong to any JP1 resource group is not subject to access control. This means that all JP1 users can view or change that unit. Conversely, if a JP1 resource group is not associated with any JP1 users, it cannot be accessed by any JP1 users. When deploying JP1/AJS3, make sure that all JP1 users are granted the appropriate access permissions for all units, as follows:

- Have JP1 resource groups been set for all units?
- Have appropriate access permissions been set for the JP1 resource groups, and are all JP1 resource groups associated with JP1 users?
- Have all JP1 users been associated with a JP1 resource group?

Multiple JP1 resource groups can be associated with a single JP1 user. For example, the same JP1 user can be associated with General Affairs Department JP1 resource group and with the Sales Department JP1 resource group. This allows the access permissions to the two departments to be controlled separately for a particular JP1 user.

#### (2) Determining JP1 permission levels

The operations that can be performed on a JP1 resource group are determined by its *JP1 permission level*.

A JP1 permission level is set for each JP1 resource group associated with JP1 users. For example, if a JP1 user is associated with the General Affairs Department JP1 resource group and with the Sales Department JP1 resource group, you can set a JP1 permission level for the General Affairs Department JP1 resource group allowing units to be executed and edited, and a different JP1 permission level for the Sales Department JP1 resource group allowing units to be viewed only. Thus, the associated JP1 user will have permission to execute and edit units in the General Affairs Department JP1 resource group, and to view units in the Sales Department JP1 resource group.

There are three types of JP1 permission level:

- Access permission for defining and executing jobnets
- Access permissions for executing and working with commands in the execution environment of QUEUE jobs and submit jobs
- Access permissions for operating agent management information



The names of the permission levels and the operations possible with each are covered below. Refer to the details on operations given here when setting JP1 permission levels.

### (a) Access permissions when defining and executing jobnets

There are the following five kinds of access permissions for defining and executing jobnets:

- **JP1\_AJS\_Admin**  
This is the administrator's permission. It confers ownership of the unit, the permission to operate resource groups, the permissions to define, execute, and edit jobnets, and so on.
- **JP1\_AJS\_Manager**  
Confers permissions including the permissions to define, execute, and edit jobnets.
- **JP1\_AJS\_Editor**  
Confers permissions including the permissions to define and edit jobnets.
- **JP1\_AJS\_Operator**  
Confers permissions including the permissions to execute and view jobnets.
- **JP1\_AJS\_Guest**  
Confers permissions including the permissions to view jobnets.

The following table lists the JP1 permission level names and details of their operation when defining and executing jobnets.

**Table 6–3: JP1 permission level names and available operations when jobnets are defined and executed**

| Operation details   | JP1_AJS_Admin | JP1_AJS_Manager | JP1_AJS_Editor  | JP1_AJS_Operator | JP1_AJS_Guest |
|---|---------------|-----------------|-----------------|------------------|---------------|
| Changing the owner, JP1 resource group, or execution-user type of a unit for which you do not have owner permission (by using the <b>Executed by</b> setting) <sup>#1</sup> | P             | --              | --              | --               | --            |
| Defining units  | P             | P               | P               | --               | --            |
| Changing the definition of units under the jobnets  | P             | p <sup>#2</sup> | p <sup>#2</sup> | --               | --            |
| Changing jobnet definitions   | P             | P               | P               | --               | --            |
| Replicating and moving units, and changing their names <sup>#3</sup>  | P             | P               | P               | --               | --            |
| Deleting units <sup>#4</sup>  | P             | P               | P               | --               | --            |
| Outputting units to standard output files   | P             | P               | P               | P                | P             |
| Outputting definitions of units to standard output files  | P             | P               | P               | P                | P             |
| Backing up units  | P             | P               | P               | P                | P             |
| Recovering units  | P             | P               | P               | --               | --            |
| Defining calendar information for a job group   | P             | P               | P               | --               | --            |
| Defining a jobnet execution schedule for a specific period  | P             | P               | P               | P                | P             |

| Operation details  | JP1_AJS_Admin | JP1_AJS_Manager | JP1_AJS_Editor | JP1_AJS_Operator | JP1_AJS_Guest |
|--|---------------|-----------------|----------------|------------------|---------------|
| Registering a defined jobnet for execution   | P             | P               | --             | P                | --            |
| Canceling the registration of a jobnet for execution   | P             | P               | --             | P                | --            |
| Registering a jobnet for release   | P             | P               | p#5            | p#5              | --            |
| Canceling the release of a jobnet  | P             | P               | p#5            | p#5              | --            |
| Viewing the release information for a jobnet   | P             | P               | P              | P                | P             |
| Outputting information such as the job execution history, current status, and next planned execution to a standard output file | P             | P               | P              | P                | P             |
| Temporarily changing a schedule defined in a jobnet  | P             | P               | --             | P                | --            |
| Temporarily changing the status of a job   | P             | P               | --             | P                | --            |
| Changing the status of a job   | P             | P               | --             | P                | --            |
| Suspending the execution of a jobnet   | P             | P               | --             | P                | --            |
| Rerunning a jobnet   | P             | P               | --             | P                | --            |
| Killing a job or jobnet  | P             | P               | --             | P                | --            |
| Exporting units  | P             | P               | P              | P                | P             |
| Importing units  | P             | P               | P              | --               | --            |
| Exporting the execution registration status of a jobnet  | P             | P               | P              | P                | P             |
| Importing the execution registration status of a jobnet  | P             | P               | --             | P                | --            |

Legend:

P: Possible

--: Not possible

#1

If no owner has been set for a unit, all users can change the JP1 resource group, owner, and execution-user type (by using the **Executed by** setting).

For details on unit owner permission, see (3) *Unit owner permission*.

#2

The items you can change in a job definition depend on the execution-user type (in the **Executed by** setting). For details, see (6) *Access permissions for changing a job definition*.

#3

The permission is required for the parent unit of the unit to be copied, moved, or renamed. For example, when you want to move the unit /AAA/BBB/CCC, the permission must be set for the JP1 resource group of the unit /AAA/BBB. Note that you need the following permissions for a unit that you want to copy, move, or rename.

To copy a unit, the JP1\_AJS\_Admin, JP1\_AJS\_Manager, JP1\_AJS\_Editor, JP1\_AJS\_Operator, or JP1\_AJS\_Guest permission must be set for the JP1 resource group of the unit (including subordinate units).

To move or rename a unit, the JP1\_AJS\_Admin, JP1\_AJS\_Manager, or JP1\_AJS\_Editor permission must be set for the JP1 resource group of the unit.

#4

The permission is also required for the parent unit of the unit to be deleted. For example, when you want to delete the unit /AAA/BBB/CCC, the permission must be set for the JP1 resource groups of the unit /AAA/BBB/CCC (including subordinate units) and the unit /AAA/BBB.

#5

JP1\_AJS\_Editor and JP1\_AJS\_Operator permissions are both required.

This is because registering a jobnet for release and canceling the release involves changing the jobnet definition and submitting the redefined jobnet for execution.

A user who performs operations on a unit must have permission to operate the JP1 resource group set for that unit. In addition, the JP1\_AJS\_Admin, JP1\_AJS\_Manager, JP1\_AJS\_Editor, JP1\_AJS\_Operator, or JP1\_AJS\_Guest permission must be set for the JP1 resource groups of the upper-level units.

JP1 users mapped to OS users with administrator's permissions or superuser permissions can execute all operations regardless of their JP1 permission level. These JP1 users can also execute commands to perform all operations on units regardless of the permissions set in the JP1\_USERNAME environment variable for the JP1 users. However, if *yes* is set for the ADMACLIMIT environment setting parameter instead of the default, only the operations within the JP1 permission level are allowed.

For details about the ADMACLIMIT environment setting parameter, see *20.11.2(4) ADMACLIMIT* in the *JP1/Automatic Job Management System 3 Configuration Guide*.

Note that if no JP1 resource group has been set for a unit, all users can perform all JP1/AJS3 operations with respect to that unit.

#### Cautionary notes

- Access permission for the referenced JP1/AJS3 - Manager is granted for manager job groups and manager jobnets.
- While JP1/AJS3 - View is connected to JP1/AJS3 - Manager, access permission information is stored in a JP1/AJS3 - Manager cache. For this reason, when access permissions are changed, the change might not be applied to the connected JP1/AJS3 - View. When you change access permissions, disconnect JP1/AJS3 - View from JP1/AJS3 - Manager, change the access permissions, and then connect JP1/AJS3 - View again.

### **(b) Access permissions for executing and working with commands in the execution environment of QUEUE jobs and submit jobs**

There are three types of access permissions for executing and working with commands in the execution environment of QUEUE jobs and submit jobs.

- **JP1\_JPQ\_Admin**  
This is the administrator's permission. It confers the permission to set job execution environments, the permission to operate queues and job executing agents, and the permission to operate jobs queued by other users.
- **JP1\_JPQ\_Operator**  
This confers the permission to operate queues and agents, and the permission to operate jobs queued by other users.
- **JP1\_JPQ\_User**  
This confers the permission to register submit jobs and to operate jobs that you have queued yourself.

When setting access permissions for executing and working with commands in the execution environment of QUEUE jobs and submit jobs, assign these permission levels to the JP1 resource group JP1\_Queue. Note that the entry JP1\_Queue is case sensitive.

The following table lists the JP1 permission level names and details of their operation for executing and working with commands used in the execution environment of QUEUE jobs and submit jobs.

Table 6–4: JP1 permission level names and available operations for executing and working with commands in the execution environment of QUEUE and submit jobs

| Operation details   | JP1_JPQ_Admin | JP1_JPQ_Operator | JP1_JPQ_User |
|---|---------------|------------------|--------------|
| Registering submit jobs                                     | P             | P                | P            |
| Canceling/killing job execution                             | P             | P                | O            |
| Holding job execution/releasing a job from held status      | P             | P                | O            |
| Moving a job  | P             | P                | O            |
| Outputting job information                                  | P             | P                | O            |
| Outputting ended job information                            | P             | P                | O            |
| Deleting ended job information from the database            | P             | P                | --           |
| Opening a queue   | P             | P                | --           |
| Closing a queue   | P             | P                | --           |
| Adding a queue  | P             | --               | --           |
| Deleting a queue  | P             | --               | --           |
| Outputting queue information                                | P             | P                | P            |
| Changing the definition of a queue                          | P             | --               | --           |
| Connecting a queue to an agent                              | P             | --               | --           |
| Disconnecting a queue from an agent                         | P             | --               | --           |
| Changing the maximum number of concurrently executable jobs | P             | --               | --           |
| Adding an agent   | P             | --               | --           |
| Deleting an agent   | P             | --               | --           |
| Outputting agent host information                           | P             | --               | --           |
| Adding an execution-locked resource                         | P             | --               | --           |
| Deleting an execution-locked resource                       | P             | --               | --           |
| Outputting information about an execution-locked resource   | P             | P                | P            |

Legend:

P: Possible.

O: Possible, but not for jobs executed by other users.

--: Not possible

Cautionary note

The execution and operation of commands used in the execution environment of QUEUE and submit jobs is subject to the access permissions defined for the manager that requested the processing.

### (c) Access permissions for working with agent management information

There are three kinds of access permissions for working with agent management information:

- **JP1\_JPQ\_Admin**  
This is the administrator's permission. It is the permission required to add, edit, and delete execution agent and execution agent group definitions.
- **JP1\_JPQ\_Operator**  
This is the permission required to change the job transfer restriction status of execution agents and execution agent groups.
- **JP1\_JPQ\_User**  
This is the permission required to view statuses and definition information for execution agents and execution agent groups.

When setting access permissions for working with agent management information, assign these permission levels to the JP1 resource group JP1\_Queue. Note that the entry JP1\_Queue is case sensitive.

The following table lists the JP1 permission level names and operational details for working with agent management information.

**Table 6–5: JP1 permission level names and available operations for working with agent management information**

| Operation details   | JP1_JPQ_Admin | JP1_JPQ_Operator | JP1_JPQ_User |
|---|---------------|------------------|--------------|
| Adding an execution agent   | P             | --               | --           |
| Adding an execution agent group   | P             | --               | --           |
| Deleting an execution agent   | P             | --               | --           |
| Deleting an execution agent group   | P             | --               | --           |
| Changing the execution host for an execution agent                                    | P             | --               | --           |
| Changing the number of concurrently executable jobs at an execution agent             | P             | --               | --           |
| Changing the description of an execution agent  | P             | --               | --           |
| Changing the description of an execution agent group                                  | P             | --               | --           |
| Adding an execution agent that connects to an execution agent group                   | P             | --               | --           |
| Changing the priority of an execution agent connected to an execution agent group     | P             | --               | --           |
| Removing an execution agent as a connection-destination of an execution agent group   | P             | --               | --           |
| Changing the job transfer restriction status of an execution agent                    | P             | P                | --           |
| Changing the job transfer restriction status of an execution agent group              | P             | P                | --           |
| Displaying the status of an execution agent <sup>#</sup>                              | P             | P                | P            |
| Displaying the status of an execution agent group <sup>#</sup>                        | P             | P                | P            |
| Displaying the status of all execution agents and execution agent groups <sup>#</sup> | P             | P                | P            |
| Displaying the names of all execution agents and execution agent groups <sup>#</sup>  | P             | P                | P            |

| Operation details   | JP1_JPQ_Admin | JP1_JPQ_Operator | JP1_JPQ_User |
|---|---------------|------------------|--------------|
| Outputting the definition of an execution agent <sup>#</sup>                              | P             | P                | P            |
| Outputting the definition of an execution agent group <sup>#</sup>                        | P             | P                | P            |
| Outputting the definition of all execution agents and execution agent groups <sup>#</sup> | P             | P                | P            |

Legend:

P: Possible.

--: Not possible

#

Users with administrator's permissions or superuser permissions can perform all operations regardless of their JP1 permission levels.

### (3) Unit owner permission

The user who defines a job or jobnet is set as the owner and holds owner permission. Owner permission allows the user to change the following settings for the job or jobnet, regardless of the set JP1 permission level:

- JP1 resource group
- Owner
- Execution-user type (through the **Executed by** setting for jobs)

Even if you are the unit owner, unless you have permission to view the unit's JP1 resource group, you cannot open the Define Details dialog box in JP1/AJS3 - View. That is, although you are the unit owner, you cannot change the unit's **JP1 resource group**, **Owner**, or **Executed by** setting in JP1/AJS3 - View. To change the settings in this situation, execute the `ajschange` command and change the JP1 resource group to a JP1 resource group that gives you view permission. For details on the `ajschange` command, see *ajschange* in 3. *Commands Used for Normal Operations* in the manual *JP1/Automatic Job Management System 3 Command Reference*.

If no owner has been set for the unit, all users can change the **JP1 resource group**, **Owner**, and **Executed by** setting.

### (4) Access permission for creating or copying a unit, or making a release entry

When a user creates or copies a unit, or makes a release entry for a unit, the unit owner and JP1 resource group are set by default as follows:

Creating a unit in JP1/AJS3 - View

Owner: The JP1 user who created the unit

JP1 resource group: The JP1 resource group of the upper-level unit

Creating a unit using the `ajsdefine` command

Owner: The owner set in the unit definition file

JP1 resource group: The JP1 resource group set in the unit definition file

Copying a unit using JP1/AJS3 - View or the `ajscopy` command

Owner: The JP1 user who copied the unit

JP1 resource group: The JP1 resource group of the copy-source unit

Registering a unit for release using JP1/AJS3 - View or the `ajsrelease` command

Owner: The owner of the unit to be released

JP1 resource group: The JP1 resource group of the unit to be released

Creating a unit by centrally or individually exporting using JP1/AJS3 - Definition Assistant

Owner: The owner set in the definition management template

JP1 resource group: The JP1 resource group set in the definition management template

You can apply functionality so that a unit that you created, copied, or made a release entry for inherits the owner and JP1 resource group from its upper-level unit. This functionality is known as the *upper-level unit-attribute inheritance function*. Using this functionality you can, if you wish, set the same owner for all units in the system without having to change the owner or JP1 resource group of the unit you are creating, copying, or releasing.

Cautionary note

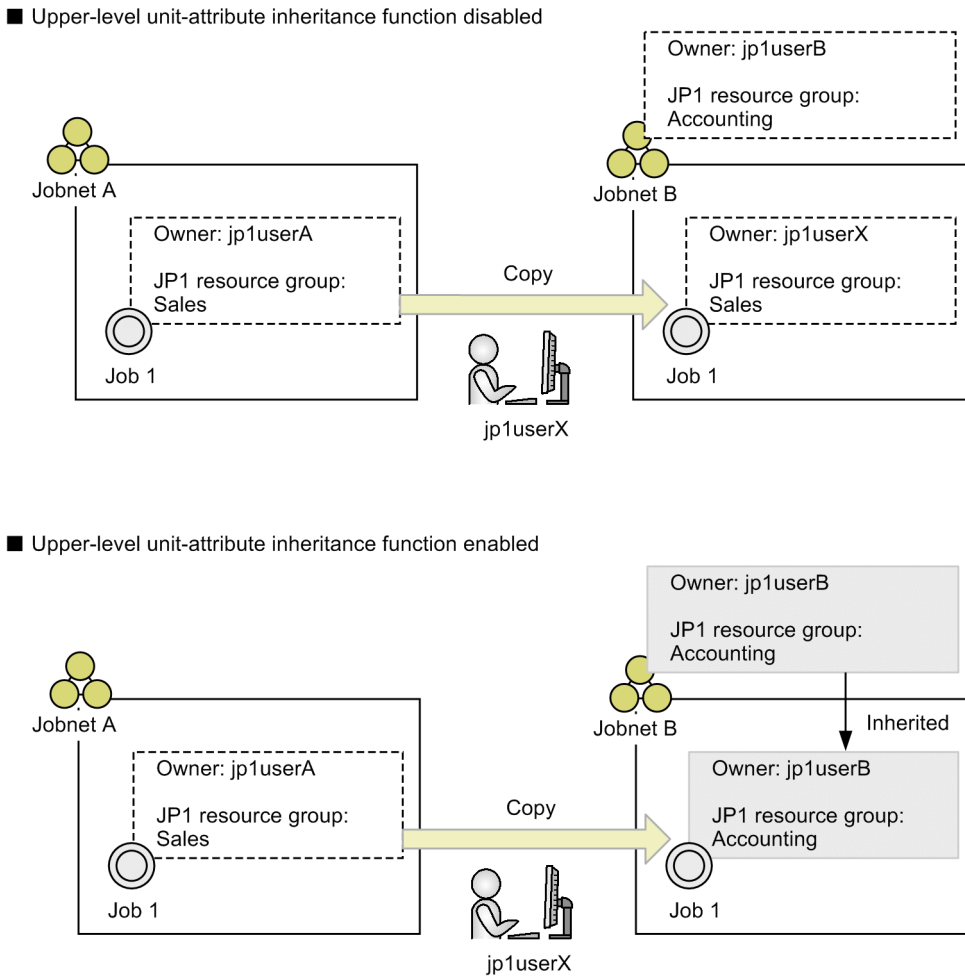
It is not recommended that you change the JP1 resource group of the release-target unit by means of a release entry. If you use a release entry to change the JP1 resource group, a JP1 user who has access rights to the unit that existed before the release must execute commands or use JP1/AJS3 - View even after the release.

### **(a) Overview of the upper-level unit-attribute inheritance function**

The upper-level unit-attribute inheritance function can be applied to a job group or jobnet. Every unit that is created, copied, or set for release in the job group or jobnet for which this functionality has been set will inherit the owner and JP1 resource group of the upper-level unit. This functionality does not change the owner or JP1 resource group of units that you have already defined or cut and pasted.

The following figure shows an example of the upper-level unit-attribute inheritance function.

Figure 6–2: Example of the upper-level unit-attribute inheritance function



When the upper-level unit-attribute inheritance function is disabled, the JP1 user who performed the copy operation (*jp1userX*) is set as the owner, and the JP1 resource group (*Sales*) of the copy-source unit is set as the JP1 resource group of the new Job 1 when Job 1 is copied to Jobnet B.

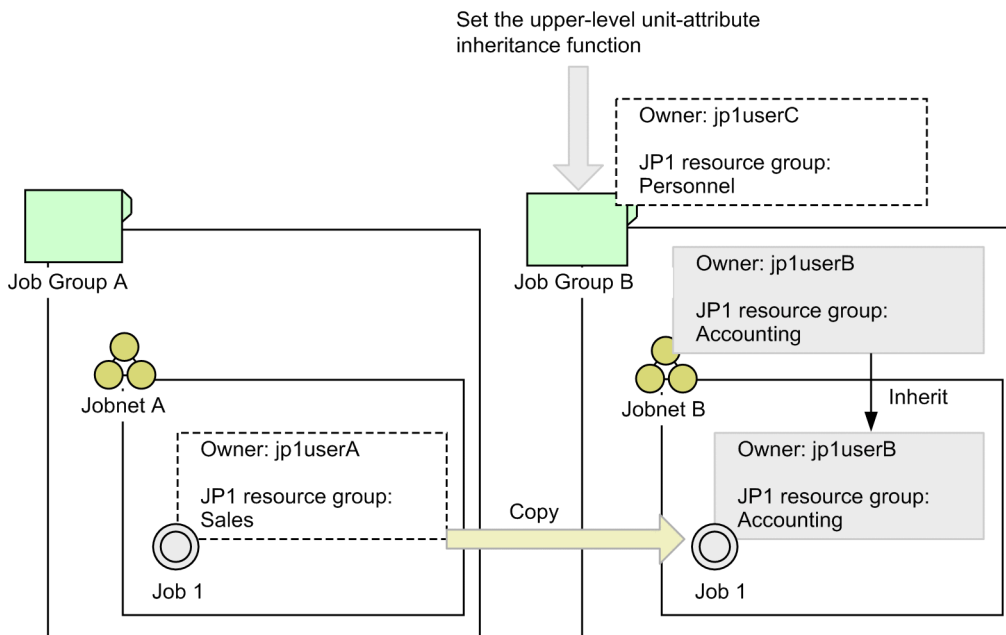
When the upper-level unit-attribute inheritance function is enabled, the owner and JP1 resource group set for Jobnet B are inherited when Job 1 is copied to Jobnet B.

When the upper-level unit-attribute inheritance function is set for a job group or jobnet configured with two or more levels, the owner and JP1 resource group are inherited from the unit directly above the newly created or copied unit.

The following figure shows an example of the upper-level unit-attribute inheritance function set for a multi-level job group.



Figure 6–3: Example of the upper-level unit-attribute inheritance function set for a multi-level job group



In this example, the upper-level unit-attribute inheritance function is set for Job Group B. When you copy Job 1 from Jobnet A to Jobnet B, the new unit inherits the owner and JP1 resource group settings from Jobnet B, which is directly above it. It does not inherit settings from Job Group B.

#### Cautionary notes

- The upper-level unit-attribute inheritance function is supported in JP1/AJS3 - Manager 09-50 and later versions. However, if you are using JP1/AJS3 - View 09-10 or an earlier version to connect to the Manager program, copied units will inherit the settings from the upper-level unit, but any new units you create will not inherit upper-level settings.
- When you restore a unit by using the `ajsrestore` command or JP1/AJS3 - View, or when you import a unit by using the `ajsimport` command, the original definition is restored. For this reason, the upper-level unit-attribute inheritance function is not enabled when it is set for the restoration-destination upper-level unit.
- When you make a release entry, if the upper-level unit-attribute inheritance function is set for the release-target upper-level unit, the function settings take precedence. If you want to apply the owner and JP1 resource group settings of the release-source unit, disable the upper-level unit-attribute inheritance function or define unit-attribute profiles so that the source and target units have the same attributes.
- When you create a new unit by copying an existing unit, and the full name of the new unit is the same as the full name of a unit for which the upper-level unit-attribute inheritance function is enabled, the new unit inherits the owner and JP1 resource group settings of the unit that was copied.

### (b) Setting the upper-level unit-attribute inheritance function

To set the upper-level unit-attribute inheritance function, you need to define a *unit-attribute profile*. The unit-attribute profile sets the full path name of the unit to which the function applies and defines how the function is to behave.

- Full path name of the target unit

Specify the full path name of the job group or jobnet to which the upper-level unit-attribute inheritance function will apply. To set the function for all units controlled by the scheduler services, enter a slash (/).

The unit or units that you specify in the profile are the same as those to which the execution-user fixing function applies. For details on the execution-user fixing function, see (5) *Job execution user*.

- Function behavior

Set either of the following values to specify how the function is to behave:

`entryuser`

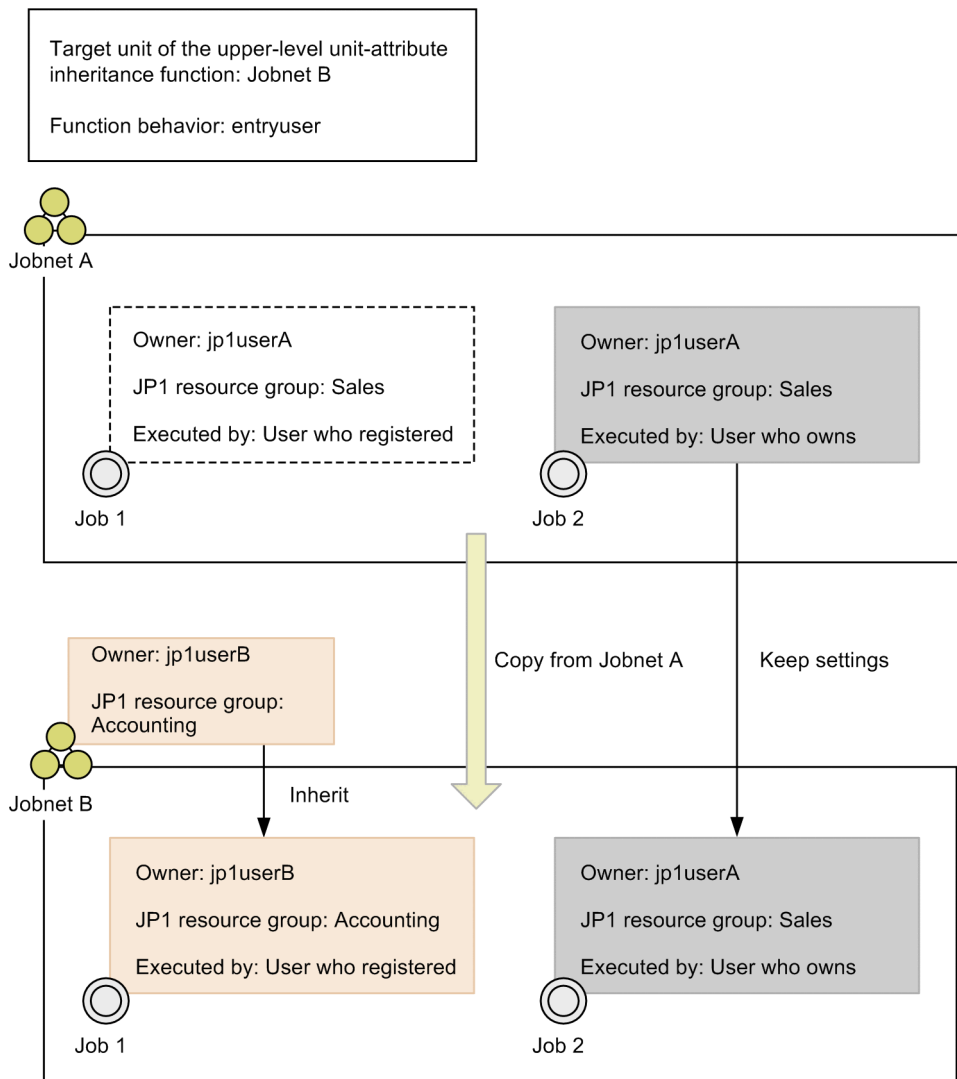
A unit copied to or created in the target job group or jobnet inherits the owner and JP1 resource group settings from the upper-level unit. However, if the new unit was copied from a source job whose execution-user type (**Executed by**) is the unit owner, the source job settings are kept and the settings of the upper-level unit are not inherited.

`all`

A unit copied to or created in the target job group or jobnet inherits the owner and JP1 resource group settings from the upper-level unit. Even if the new unit was copied from a source job whose execution-user type (**Executed by**) is the unit owner, the settings of the upper-level unit are still inherited.

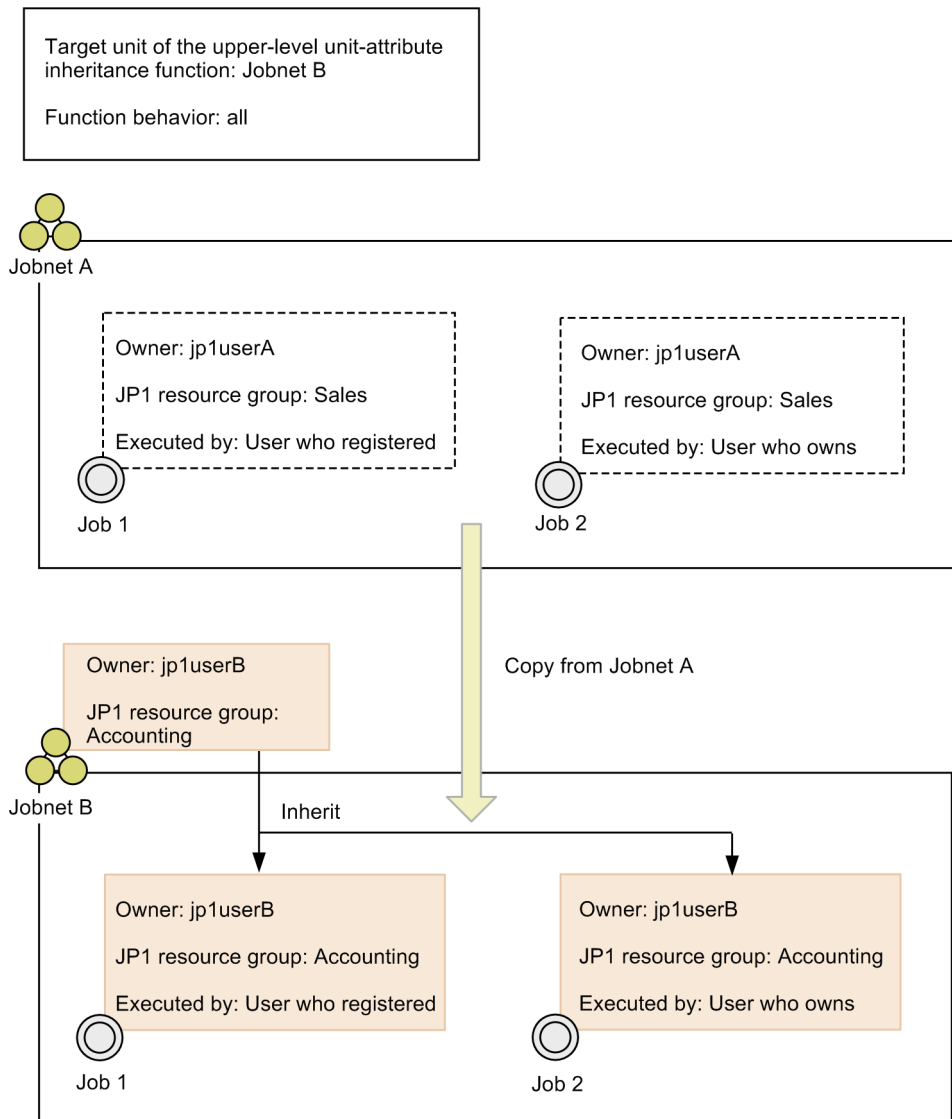
The following figures show the differences in function behavior between the `entryuser` and `all` settings.

Figure 6–4: Behavior of the upper-level unit-attribute inheritance function ("entryuser" set)



In this example, `entryuser` is set for the upper-level unit-attribute inheritance function. When Job 1 and Job 2 are copied to Jobnet B, Job 1 inherits the owner and JP1 resource group settings from Jobnet B because the **Executed by** setting is **User who registered** for Job 1. Job 2, on the other hand, keeps the owner and JP1 resource group settings of the copy-source unit because its **Executed by** setting is **User who owns**.

Figure 6–5: Behavior of the upper-level unit-attribute inheritance function ("all" set)



In this example, `all` is set for the upper-level unit-attribute inheritance function. When Jobs 1 and 2 are copied to Jobnet B, both jobs inherit the owner and JP1 resource group settings from Jobnet B.

**Supplementary note**

When the upper-level unit-attribute inheritance function is set for multiple units at different levels in a job group or jobnet, the function behavior set for the upper-level unit closest to the copied or created unit takes precedence. For example, if the function is set for a job group and for a jobnet in that job group, the function behavior set for the jobnet will take precedence for any jobs added to the jobnet.

For details on setting a unit-attribute profile, see *21.1.3 Setting up the upper-level unit-attribute inheritance function and execution-user fixing function* in the *JP1/Automatic Job Management System 3 Configuration Guide*. For more information about unit-attribute profiles, see *21.1.4 Details of unit-attribute profile* in the *JP1/Automatic Job Management System 3 Configuration Guide*.

**(c) Example of using the upper-level unit-attribute inheritance function**

The following describes an example of using the upper-level unit-attribute inheritance function.

For the purpose of this example, assume that jobnets are defined according to the following policies:

### Policy for defining access permissions

The access permissions (owner and JP1 resource group) are set as `user1` and `Accounting`, respectively, for all units.

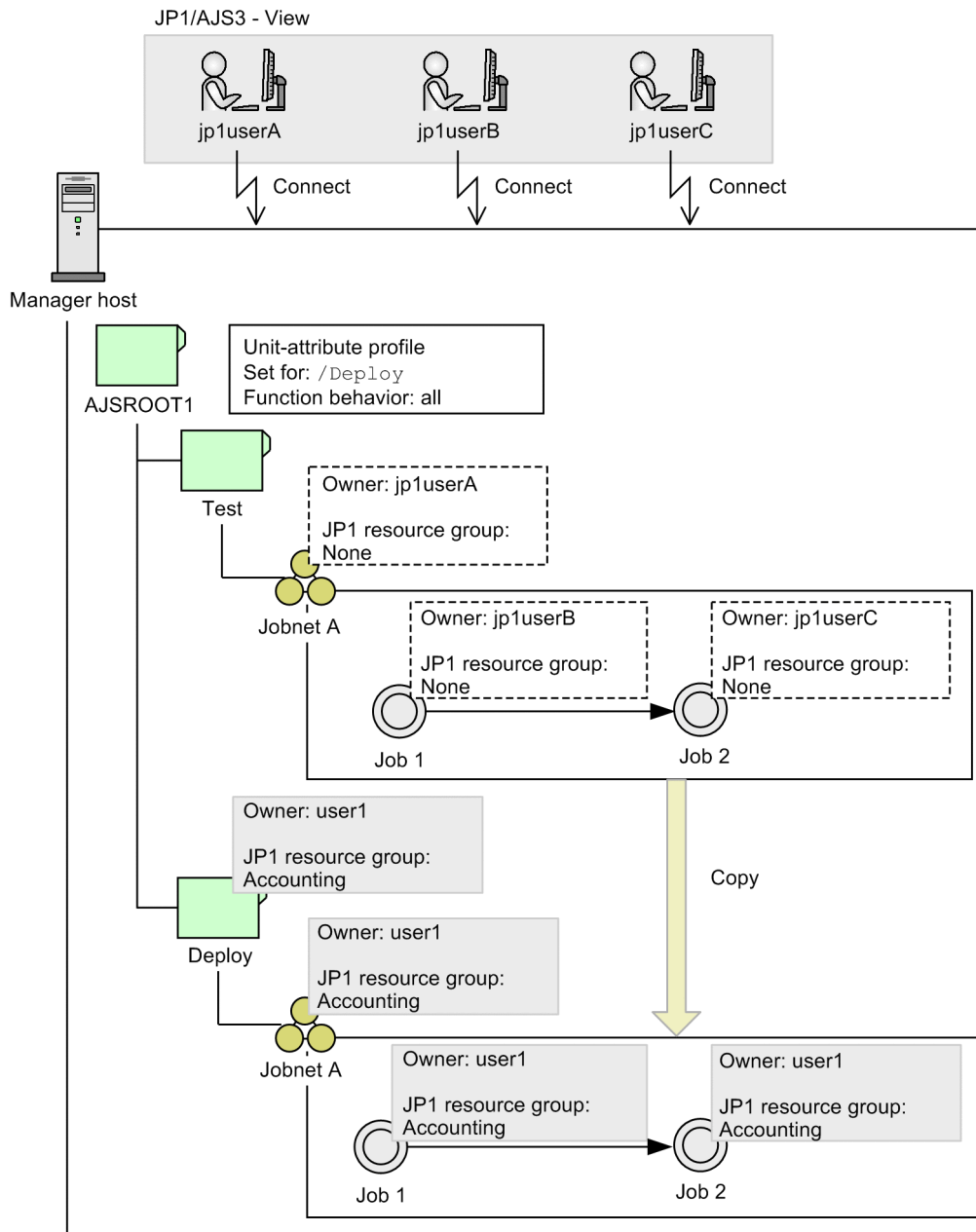
### Policy for defining units

- When two or more JP1 users are connected to a manager host by JP1/AJS3 - View, each JP1 user defines units for the particular tasks he or she is responsible for.
- Create Jobnet A in a job group *Test* for testing purposes, and define Jobs 1 and 2 under *Jobnet A*.
- Perform a test run of Jobnet A in the job group *Test*. Then copy Jobnet A to the job group *Deploy* and start operation.

Let's assume that the manager host environment has been configured and the *Test* and *Deploy* job groups have been set up. Assume also that `user1` is set as the owner and `Accounting` is set as the JP1 resource group of job group *Deploy*.

The following figure shows an example of defining a jobnet with these settings using the upper-level unit-attribute inheritance function.

Figure 6–6: Example of using the upper-level unit-attribute inheritance function



In this example, JP1 users *jp1userA*, *jp1userB*, and *jp1userC* are connected to the manager host by JP1/AJS3 - View. The unit-attribute profile for *AJSROOT1* defines */Deploy* as the target unit to which the upper-level unit-attribute inheritance function applies, and *all* as the function behavior.

The JP1 users each define Jobnet A, Job 1, and Job 2 in *Test*, the job group for testing units. In each case, the JP1 user name is set as the owner of the defined unit. Having been tested in the *Test* job group, Jobnet A can now be copied to the *Deploy* job group. The copied Jobnet A inherits the access permissions of the upper-level *Deploy* job group. Similarly, the copied Job 1 and Job 2 inherit the access permissions of the upper-level Jobnet A, and hence the access permissions of the *Deploy* job group. This means that all units copied to the *Deploy* job group can be run with the same owner and JP1 resource group.

## (5) Job execution user

When a jobnet is executed, the JP1 user who executes the jobs defined in the jobnet is known as the *execution user*.

You can select either of the following as the execution user of a job (by using the **Executed by** setting):

#### **User who registered**

The job is to be executed by the JP1 user who registered the jobnet for execution, if the user has execution permission for the job.

#### **User who owns**

The job is to be executed by the JP1 user set as the job owner, if the user has execution permission for the job. It does not matter whether execution permission has been granted to the JP1 user who registers the jobnet for execution.

OR jobs, judgment jobs, and event jobs are executed under the account of the user who started JP1/AJS3. For this reason, you cannot set **Executed by** for OR jobs or judgment jobs. If you set **Executed by** for an event job, the setting will be ignored.

At execution, the job is forwarded to the target host system (the agent host). The job execution user must therefore be mapped to the OS user on the target host system. If you set **User who registered** in the job's **Executed by** setting, map the JP1 user who registers the jobnet for execution to the OS user. If you set **User who owns**, map the job owner to the OS user.

Regardless of a job's **Executed by** setting, you can fix its execution user as the JP1 user defined as the owner of the upper-level unit. This functionality is called the *execution-user fixing function*. It allows all the jobs in a jobnet to be executed under the same JP1 user account, regardless of who registered the jobnet for execution.

#### Cautionary note

The execution-user fixing function is supported in JP1/AJS3 - Manager 09-50 and later versions.

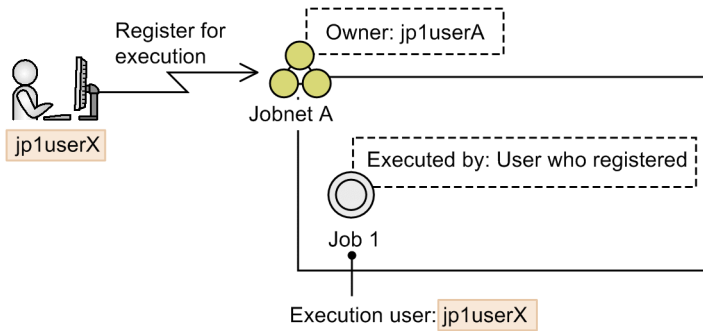
### **(a) Overview of the execution-user fixing function**

The execution-user fixing function can be set for a job group or jobnet. Jobs in the job group or jobnet for which the function has been set are executed by the JP1 user set as the owner of that job group or jobnet, regardless of the **Executed by** setting in the detailed definition of each job.

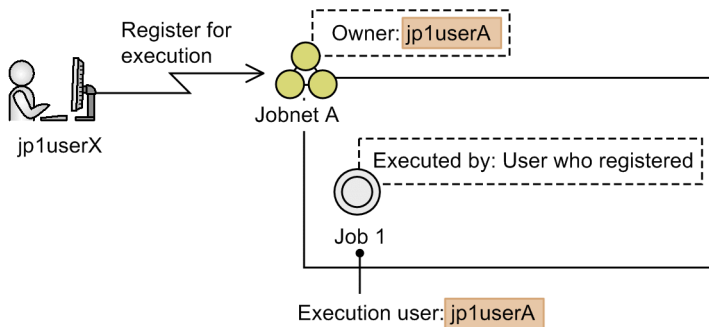
The following figure shows an example of the execution-user fixing function.

Figure 6–7: Example of the execution-user fixing function

■ Execution-user fixing function disabled



■ Execution-user fixing function enabled



In the first case above, the execution-user fixing function is disabled. When *jp1userX* registers Jobnet A for execution, Job 1 will be executed by *jp1userX* because the job's **Executed by** setting is **User who registered**.

In the second case above, the execution-user fixing function is enabled. When *jp1userX* registers Jobnet A for execution, Job 1 will be executed by *jp1userA*, the owner of Jobnet A, regardless of the job's **Executed by** setting.

Cautionary notes

- When the execution-user fixing function is enabled, the actual execution user differs from the **Executed by** setting in the detailed definition. As a result, a permission error might occur when a job is executed if the execution user set by the execution-user fixing function does not have execution permission for the job, even though access permission for executing the job is set for that user in **Executed by** in the detailed definition.
- Setting the execution-user fixing function does not alter the definition information of any job. Jobs are executed by the JP1 user set by the execution-user fixing function, regardless of the **Executed by** setting in the detailed job definition.

By setting the execution-user fixing function for the upper-level unit, you can fix the execution user of the types of jobs listed below. Event jobs are excluded because they are executed independently of the execution user.

- Standard jobs<sup>#</sup>
- Action jobs
- Custom jobs
- Passing information setting jobs
- HTTP connection jobs
- Custom event jobs

#

You can fix the execution user for Unix jobs and PC jobs even if they are queueless jobs.

Cautionary note

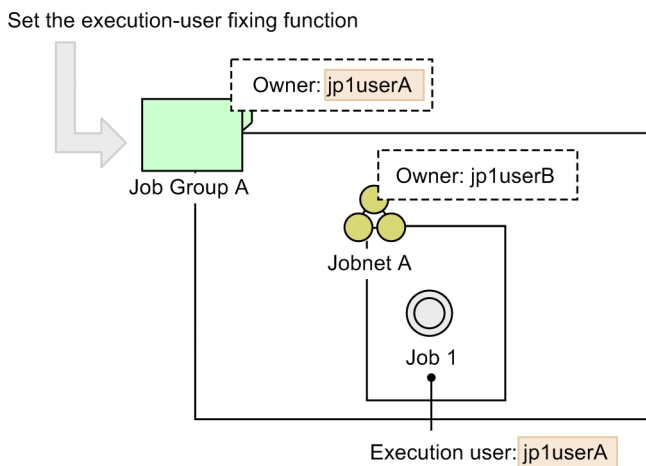
The settings for the execution-user fixing function at the connection-destination host apply to jobs in a remote jobnet.

When the execution-user fixing function is set for an upper-level unit configured over two or more levels, the JP1 user set as the job execution user will be the owner of the upper-level unit closest to the job among those units for which the execution-user fixing function has been set.

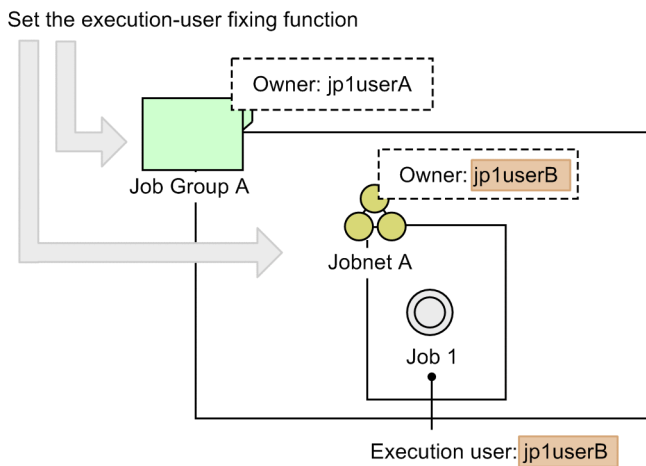
The following figure shows an example of the execution-user fixing function set for a multi-level job group.

Figure 6–8: Example of the execution-user fixing function set for a multi-level job group

Example 1:



Example 2:



In Example 1, the execution-user fixing function is set for Job Group A. When you register Jobnet A for execution, the execution user of Job 1 will be *jp1userA*, the owner of Job Group A for which the function is set.

In Example 2, the execution-user fixing function is set for Job Group A and Jobnet A. When you register Jobnet A for execution, the execution user of Job 1 will be *jp1userB*, the owner of Jobnet A, which is the closest upper-level unit for which the execution-user fixing function has been set.



## (b) Setting the execution-user fixing function

To set the execution-user fixing function, you need to define a *unit-attribute profile*. The unit-attribute profile sets the full path name of the unit to which the function applies and defines how the function is to behave.

- Full path name of the target unit

Specify the full path name of the job group or jobnet to which the execution-user fixing function will apply. To set the function for all units controlled by the scheduler services, enter a slash (/).

The unit or units that you specify in the profile are the same as those to which the upper-level unit-attribute inheritance function applies. For details on the execution-user fixing function, see (4) *Access permission for creating or copying a unit, or making a release entry*.

- Function behavior

Set either of the following values to specify how the function is to behave:

`entryuser`

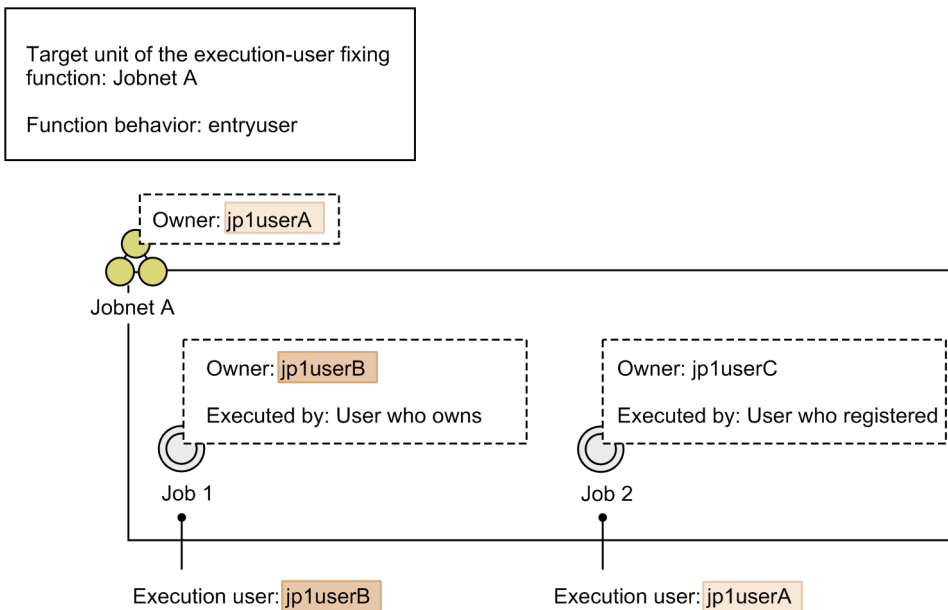
The execution user of the jobs in the unit subject to the execution-user fixing function is fixed as the owner of that unit. However, if the **Executed by** setting of any of the jobs in the unit is **User who owns**, the job owner will be the execution user for that job.

`all`

The execution user of the jobs in the unit subject to the execution-user fixing function is fixed as the owner of that unit. This applies even if the **Executed by** setting of any of the jobs in the unit is **User who owns**.

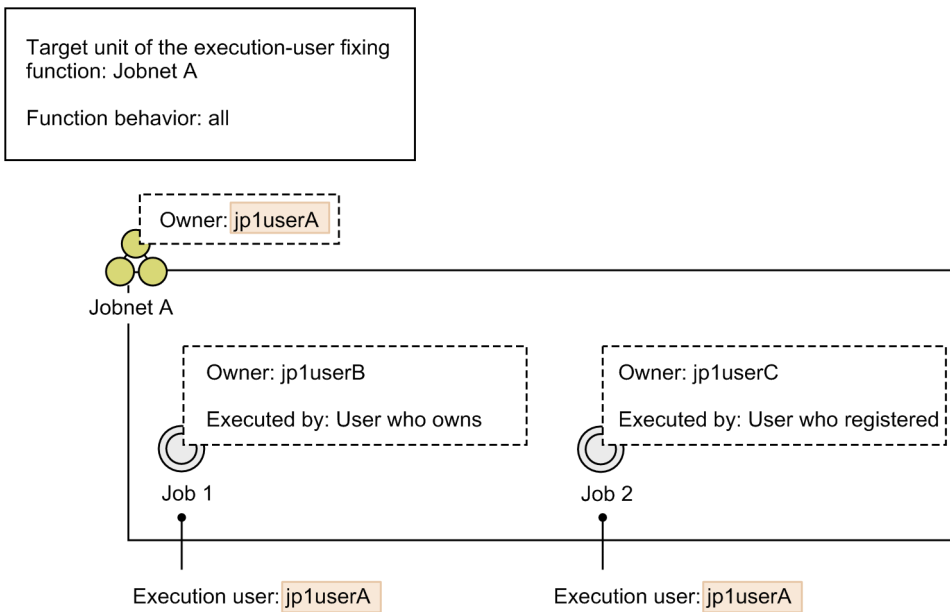
The following figures show the differences in function behavior between the `entryuser` and `all` settings.

Figure 6–9: Behavior of the execution-user fixing function ("entryuser" set)



In this example, `entryuser` is set for the execution-user fixing function. When Jobnet A is registered for execution, Job 1 will be executed by its owner (*jp1userB*) because the job's **Executed by** setting is **User who owns**. The execution user of Job 2, on the other hand, is fixed as the owner of Jobnet A (*jp1userA*) because the **Executed by** setting of Job 2 is **User who registered**.

Figure 6–10: Behavior of the execution-user fixing function ("all" set)



In this example, `all` is set for the execution-user fixing function. When Jobnet A is registered for execution, the execution user of Jobs 1 and 2 is fixed as the owner of Jobnet A (`jp1userA`).

#### Cautionary note

When the execution-user fixing function is set for multiple units at different levels in a job group or jobnet, the function behavior set for the upper-level unit closest to each job takes precedence. For example, if the function is set for a job group and for a jobnet in that job group, the function behavior set for the jobnet will take precedence for the jobs in that jobnet.

For details on setting a unit-attribute profile, see *21.1.3 Setting up the upper-level unit-attribute inheritance function and execution-user fixing function* in the *JP1/Automatic Job Management System 3 Configuration Guide*. For more information about unit-attribute profiles, see *21.1.4 Details of unit-attribute profile* in the *JP1/Automatic Job Management System 3 Configuration Guide*.

### (c) Example of using the execution-user fixing function

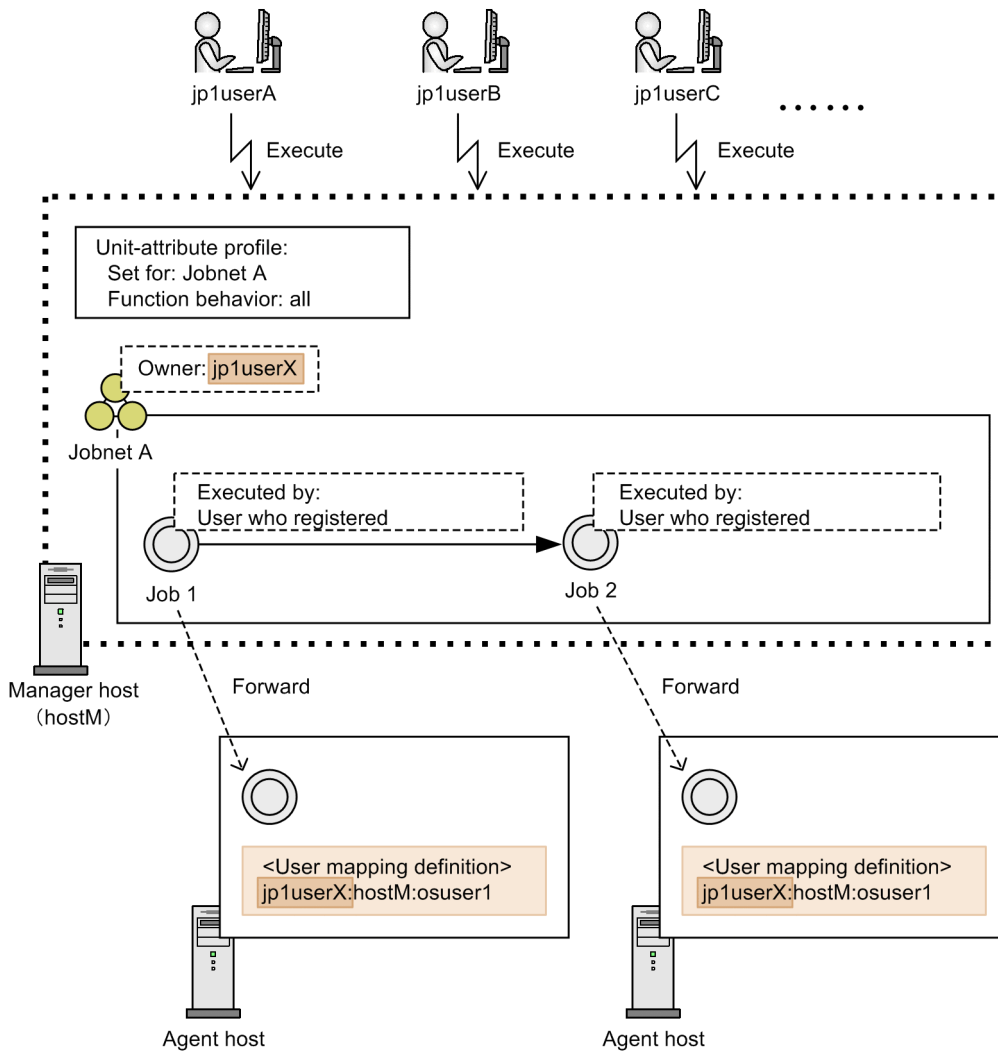
The following describes an example of using the execution-user fixing function.

For the purpose of this example, assume the following jobnet configuration:

- Jobnet A is defined on manager host `hostM` and is owned by `jp1userX`.
- Jobs 1 and 2 are defined in Jobnet A. The **Executed by** setting of both jobs is **User who registered**.

The following figure shows an example of executing Jobnet A defined as above and using the execution-user fixing function.

Figure 6–11: Example of using the execution-user fixing function



Legend:

<User mapping definition>  
 JP1-user-name-(execution-user-name):server-host-name:OS-user-name

In this example, the execution-user fixing function is set for Jobnet A and the function behavior is set as `all`. When *jp1userA*, *jp1userB*, or *jp1userC* registers the jobnet for execution, Job 1 and Job 2 are forwarded to an agent host. The execution user of both jobs is fixed as the Jobnet A owner (*jp1userX*) regardless of who registered each job. As a result, to execute the jobs on their respective agent host, *jp1userX* must be mapped with the OS user (*osuser1*). User mapping is not required for the user who registered the jobs (*jp1userA*, *jp1userB*, or *jp1userC*). There is no need to add a user mapping definition if another JP1 user who registers Jobnet A for execution is added later.

#### (d) Combining the execution-user fixing function and the upper-level unit-attribute inheritance function

You can set both the execution-user fixing function and the upper-level unit-attribute inheritance function using the same unit-attribute profile. When you use both functions, you can select the behavior of each function separately as either `entryuser` or `all`. If you select a different behavior for each function, the job execution process becomes more difficult to predict. We therefore recommend that you set the same behavior value for both functions.

The following table describes how the different function behaviors work in combination.

Table 6–6: Combining the function behavior values of the execution-user fixing function and the upper-level unit-attribute inheritance function

| No. | Behavior value of the execution-user fixing function | Behavior value of the upper-level unit-attribute inheritance function | Description  |
|-----|--|---|--|
| 1   | entryuser  | entryuser   | The execution user can be fixed as the job owner for selected jobs only. This setting remains unchanged when the job is copied. Specify this combination of values when you create a special job as a part and then copy it to incorporate it in the system. |
| 2   | all  | all   | The execution user, owner, and JP1 resource group are systemized for all the jobs in the specified unit.   |
| 3   | entryuser  | all   | We do not recommend this combination. The execution user can be fixed as the job owner for selected jobs only, but the setting changes when the job is copied.   |
| 4   | all  | entryuser   | We do not recommend this combination. You might want to keep the execution user of the copy-source job when you copy a particular job, but the execution user will be fixed as the owner of a different unit when the job is executed.                       |

## (6) Access permissions for changing a job definition

The items that can be changed in the detailed definition of a job depend on the definition contents and whether the JP1 user has access permission for the job. You can change all items in a detailed definition if any of the following conditions is met:

- The ADMACLIMIT environment setting parameter is omitted or is set to no, and the JP1 user who is to perform the operation is mapped to one of the following OS users:
  - In Windows  
An OS user who has administrator permissions
  - In UNIX  
An OS user who has superuser permissions
- A resource group and owner are set for the operation-target job, and the JP1 user who is to perform the operation has the JP1\_AJS\_Admin permission for that resource group.
- No resource group is specified for the operation-target job.

If the JP1 user does not have administrator's permissions, JP1\_AJS\_Manager or JP1\_AJS\_Editor permission level must be granted to change the detailed definition of a job. The definition items that can be changed depend on whether the JP1 user has owner permission for the job and the job's **Executed by** setting.

The following table describes which definition items can be changed according to whether the JP1 user has owner permission and the **Executed by** setting.

Table 6–7: Editable items in a detailed job definition

| Whether the JP1 user has owner permission | "Executed by" setting |                    |
|---|-----------------------|--------------------|
|   | User who registered   | User who owns      |
| Yes                                       | All                   | All <sup>#1</sup>  |
| No  | Some                  | None <sup>#2</sup> |

## Legend:

All: The JP1 user can change all items.

Some: The JP1 user can change items other than the job owner, JP1 resource group, and execution-user type.

None: The JP1 user cannot change any items.

#1

If a user with owner permission changes the job owner, the job's **Executed by** setting will automatically change to **User who registered**.

This is to prevent execution by an unauthorized user. If the **Executed by** setting had remained as **User who owns**, and the JP1 user changed the owner to a user who is not permitted to execute the job, the job would be executed under an unauthorized user account.

#2

Users without owner permission cannot change detailed information for a job when the **Executed by** setting is **User who owns**. This is to prevent users from freely changing a job definition and executing it with the account of the owner previously set for the job.

If no owner has been set for a job, all definition items can be changed regardless of the job's access permission or **Executed by** setting.

## 6.5 Mapping users

Consider how the execution users of a job on the manager host are to be associated with the OS user on the job's target host system (agent host).

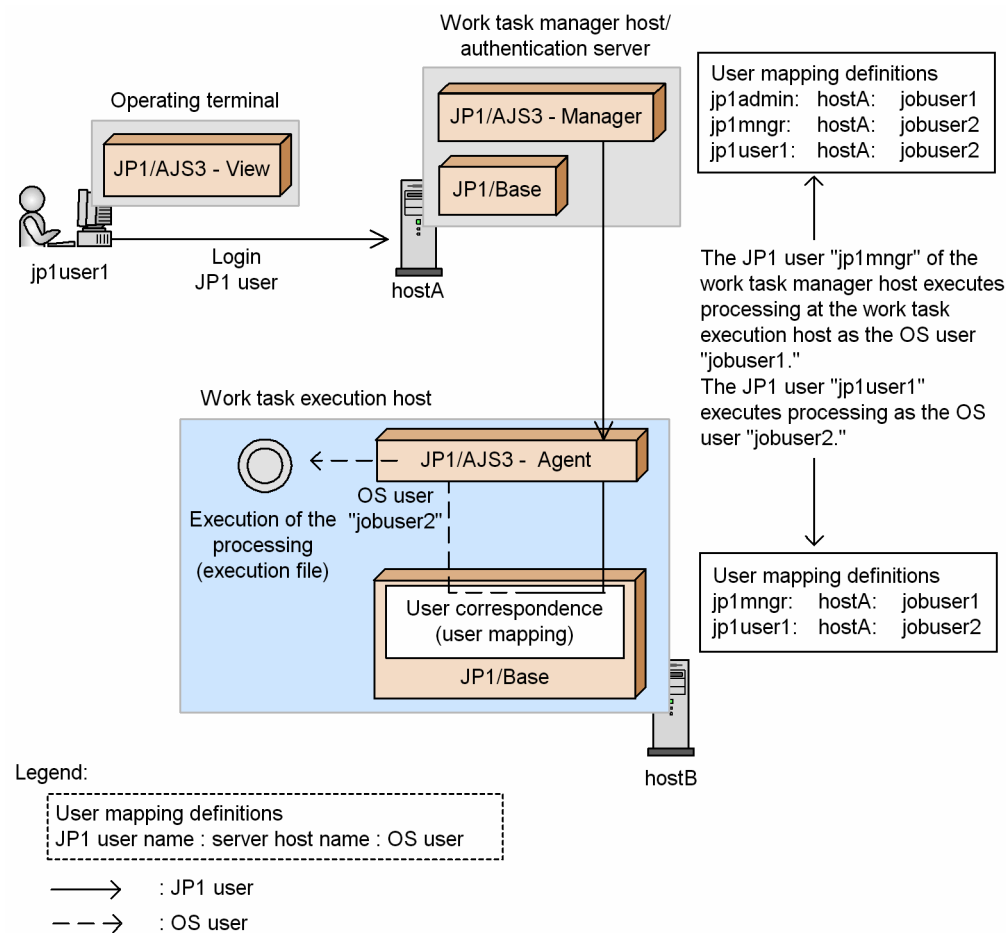
### 6.5.1 Considerations when mapping users

When you execute a job in JP1/AJS3, the manager host forwards it to an agent host. The forwarded job is executed under the account of the OS user associated with the execution user of the job. This is called *user mapping* and utilizes the JP1/Base user mapping function.

User mapping is also necessary when you log in from JP1/AJS3 - View. You must set the user mapping before using JP1/AJS3 - View.

The figure below gives an overview of processing execution using user mapping.

Figure 6–12: Overview of processing execution using user mapping



In the figure above, the following mapping is performed on the agent host:

- jp1mngr: jobuser1
- jp1user1: jobuser2

For the OS user `jobuser1`, set a user with administrator's permissions or superuser permissions. These permissions are used when they are required by the program specifications; e.g. for rebooting.

For the OS user `jobuser2`, set permissions for the executed processing (OS user account, file access permissions, etc.) so that the processing does not end abnormally. Remember that standardizing the OS user name (job-executing user) at all agent hosts makes administration easier.

The way that the user names and user mapping used when operating jobs and jobnets are decided differs according to the command used. Cases where units (jobs and jobnets) are operated with an `aj sxxxx` command and JP1/AJS3 - View, where a job in the job execution environment is operated and executed with a `jp qxxxx` command, and agent management information is operated with commands are shown below. Approach mapping by referring to the rules described below.

Note that since commands that operate event jobs and custom event jobs do not rely on the JP1 permissions level, they do not use a JP1 user name.

## (1) JP1 user names when a job network element is operated with JP1/AJS3 - View and commands

When you operate on a job network element from JP1/AJS3 - View, the JP1 user name used to check the permissions is the one used to log in to JP1/AJS3 - View.

When you operate on a job network element with an `aj sxxxx` command, the JP1 user name is decided in accordance with the following rules:

- When the environment variable `JP1_USERNAME` is set

When the environment variable `JP1_USERNAME` is set, the setting made for it is taken as the JP1 user name.

You must ensure that the OS user name at command execution and the setting for the environment variable `JP1_USERNAME` are mapped by user mapping, except when the OS user when the command is executed is a user with administrator privileges or superuser privileges, in which case mapping is not necessary.

The user mapping also differs depending on whether the environment variable `JP1_HOSTNAME` is set.

When the environment variable `JP1_HOSTNAME` is set

The user mapping defined at the logical host set for the environment variable `JP1_HOSTNAME` is used.

When the environment variable `JP1_HOSTNAME` is not set

The user mapping defined at the physical host is used.

- When the environment variable `JP1_USERNAME` is not set

When the environment variable `JP1_USERNAME` is not set, the OS user name is taken as the JP1 user name. When a job is executed the user mapping is checked, so a JP1 user with the same name as the OS user must be registered.

If a JP1 resource group name is specified in the attributes of the jobs and jobnets operated, JP1/AJS3 checks with the authentication server about access permissions. If the environment variable `JP1_HOSTNAME` is set, the logical server defined in the logical host in the setting is used, and if the environment variable `JP1_HOSTNAME` is not set, the authentication server defined in the physical host is used. However, if the OS user when the command is executed is a user with administrator privileges or superuser privileges, the authentication server is not asked about access permissions.

Next, we explain how to remotely execute a command for operating units. For details about the commands that can be remotely executed, see *1.1 Command syntax* in the manual *JP1/Automatic Job Management System 3 Command Reference*.

The following settings are required on the hosts that remotely execute commands:

- When environment variable `JP1_USERNAME` is set

If environment variable `JP1_USERNAME` is set when a command is remotely executed, the value set for `JP1_USERNAME` is used as the JP1 user name when the command is executed on the target host. One of the OS user names on the command execution destination host and the value set for environment variable `JP1_USERNAME` must be mapped by user mapping on the command execution host.

The type of user mapping to be performed differs depending on whether a logical host name or a physical host name is specified as the command execution destination host.

When a logical host name is specified for the command execution destination host:

The user mapping defined for the specified logical host is used.

When a physical host name is specified for the command execution destination host:

The user mapping defined for the specified physical host is used.

- When environment variable `JP1_USERNAME` is not set

If environment variable `JP1_USERNAME` is not set, JP1/AJS3 treats the OS user name of the command execution source host as the JP1 user name.

If a JP1 resource group name is specified in the attributes of the job or jobnet to be operated, JP1/AJS3 checks with the authentication server about access permissions. If you specify a logical host name for the command execution destination host, the authentication server defined in the logical host is used. If you specify a physical host name for the command execution destination host, the authentication server defined in the physical host is used. Set the JP1 permission level required for using the command. However, if the mapped primary user is a user with administrator privileges or superuser privileges, the authentication server is not asked about access permissions.

## **(2) JP1 user names when a job in the job execution environment is executed and operated with commands**

When you use a `jpqxxx` command to perform operations on a job in the job execution environment, or you perform operations on the job execution environment itself, the permissions are checked based on the JP1 user name with the same name as the OS user who executes the command. For that reason, register the OS user who executes the command as a JP1 user regardless of the settings in the `JP1_USERNAME` environment variable.

For details on how to register JP1 users and how to set JP1 permissions levels, see *3.1.1 Setting up JPI/Base* in the *JPI/Automatic Job Management System 3 Configuration Guide* (for Windows hosts) or see *13.1.1 Setting up JPI/Base* in the *JPI/Automatic Job Management System 3 Configuration Guide* (for UNIX hosts).

In addition, for details on the permission levels required to use the various commands, see *1.5 Commands* in the manual *JPI/Automatic Job Management System 3 Command Reference*.

## **(3) JP1 user names when agent management information is operated on with commands**

When you use a command to perform operations on agent management information, the JP1 user name is decided in accordance with the following rules:

- When the environment variable `JP1_USERNAME` is set

The user name set in the environment variable `JP1_USERNAME` is used as the JP1 user name. Note that you must ensure that the user name set in the environment variable `JP1_USERNAME` and the OS user at command execution are mapped in the JP1/Base user mapping definition. User mapping is unnecessary if the OS user at command execution is a user with administrator privileges or superuser privileges.

The user mapping also differs depending on whether the environment variable `JP1_HOSTNAME` is set.



If the environment variable `JP1_HOSTNAME` is set

The user mapping defined on the logical host set for the environment variable `JP1_HOSTNAME` is used.

If the environment variable `JP1_HOSTNAME` is not set

The user mapping defined on the physical host is used.

- When the environment variable `JP1_USERNAME` is not set

Because the OS user name is used as the JP1 user name, a JP1 user with the same name as the OS user must be registered.

When you attempt to perform an operation on agent management information, JP1/AJS3 queries the authentication server about access permissions. If you specify a logical host as the target host for the agent management information, the authentication server defined on the logical host is used. If you specify a physical host as the target host, the authentication server defined on the physical host is used. Note, however, that when you use the `ajsagtshow` and `ajsagtprint` commands as a user with administrator privileges or superuser privileges, user mapping is unnecessary and the authentication server is not queried about access permissions.

#### **(4) JP1 user names when flexible jobs are executed**

A flexible job performs user mapping on the relay agent or on the destination agent. On the relay agent, user mapping is performed in the same way as when normal jobs are executed, and performs relay processing of the mapped OS users. On the destination agent, user mapping is performed and jobs are executed by the mapped OS users. Note that if you choose to use a relay agent, set either the host name of the relay agent or \* (asterisk) for **Server host** in the user mapping definition on the destination agent.

# 7

## Cautionary Notes on Designing Work Tasks

This chapter provides cautionary notes on designing work tasks.

## 7.1 Notes on the number of root jobnets registered for execution

---

Without taking into account available disk and memory resources and processing performance, the maximum number of root jobnets that can be registered for execution in theory is 2,147,483,647. As long as four thousand root jobnets are registered in JP1/AJS3, there must be no functional problem. However, when executing jobnets and jobs in JP1/AJS3, consider the following points from a performance viewpoint.

- Notes on defining thousands of root jobnets in one hierarchy level

If 4,000 root jobnets are defined in one hierarchy level, system performance might suffer. Take the following measures if necessary. These measures assist not only execution performance, but also GUI-based monitoring and command performance.

  - When there are a large number of root jobnets that contain only a few units, aggregate the units into as few root jobnets as possible.
  - Arrange job groups into hierarchies.  
Do not group more than 500 root jobnets, including nested root jobnets, in one job group.
  - Consider the following points about the jobnets you register:
    - Provide 2 or 3 levels of hierarchy, including the root jobnet.
    - Define no more than 50 to 80 nested jobnets and jobs in the hierarchy below one root jobnet or nested jobnet.
    - Make sure that no more than approximately 500 nested jobnets and jobs are located below one root jobnet.
  - Divide the root jobnets so that no more than approximately 4,000 root jobnets are registered for execution by the scheduler service, and then run multiple scheduler services.
- Other notes
  - If you specify a *number of logs to keep*, *number-of-logs-to-keep* x *number-of-registered-root-jobnets* is used as the number of registered root jobnets for disk capacity management and resource management purposes. Therefore, be careful about the disk capacity and other resource management items.
  - Although a large number of root jobnets can be registered, this does not mean that a large number of jobs can be executed concurrently. When you design a work task, be careful about system resources and processing performance. For example, when designing a work task that uses a large number of jobs, design the work task so that there is no time period where a large number of jobs are required to run concurrently.
  - Design work tasks with a view to distributing tasks to more than one time period and CPU.
  - When many root jobnets are defined, consider root jobnet registration methods (all or divided), registration time, time for start-up, and performance of commands for batch definition, operation, and display.
  - If you register a large number of root jobnets for execution, release, or a cancellation of release within a short period, and you specify a long fixed schedule period for fixed execution registration, the system load might temporarily spike, delaying execution of root jobnet generations created when start conditions are satisfied.  
If you want to register many root jobnets for execution, release, or a cancellation of release, reduce the number of root jobnets you register at one time. If you want to register a root jobnet for fixed execution, shorten the fixed schedule period to reduce the load on the system.
  - Defining a large number of jobnets or jobs in one root jobnet, or using too many relation settings, could cause a deterioration in operability and performance of JP1/AJS3 - View and JP1/AJS3 - Web Console. To maintain operability and performance, the jobnet should have two or three hierarchy levels, and each root jobnet and nested jobnet should have about 50 to 80 units.

## 7.2 Relationship between number of logs to keep and performance

---

The *number of logs to keep* is the number of generations to be saved as the execution result of a jobnet. You can set a number of logs to keep for a root jobnet. When you set a number of logs to keep, you can view the execution result of the set generations (times) in the Daily Schedule window or Monthly Schedule window, or by using the `ajsshow` command. You can set a value from 1 to 99 as the number of logs to keep, which can be expanded to 999 in the environment settings of the JP1/AJS3 - Manager scheduler service.

However, as the number of logs to keep increases, the number of records calculated by *number-of-logs-to-keep* x *number-of-registered-units* also increases, in proportion to the scale of the jobnet. This has a significant effect on operations that access the database. For this reason, carefully consider the effect on system performance when you determine a value to set as the number of logs to keep. This setting affects the following main functionality (processing):

- Startup of a scheduler service
- Cancellation of registration for execution
- Display of the Monthly Schedule window and Daily Schedule window of JP1/AJS3 - View
- Display of the Dashboard screen and Monthly Schedule screen of JP1/AJS3 - Web Console
- Creation of generations when a start condition is satisfied
- Execution of `ajsshow` and other commands
- Job execution

For the above processing, there is a tendency for the number of records and the processing time to increase in proportion to each other. From an unloaded state, it will take about one or two minutes to cancel the registration for processing of 1,000,000 records, and 8 to 10 minutes to display the Monthly Schedule window (although this might vary depending on the system performance). This processing will take even longer if the system is experiencing heavy loads such as high CPU utilization or frequent disk access. In contrast, it will take only a few seconds to cancel the registration for processing of 100,000 records, and one or two minutes to display the Monthly Schedule window.

If you use the recommended configuration of two or three jobnet hierarchy levels with each jobnet containing about 50 to 80 units, the processing speed will be acceptable for operation even when the number of logs to keep is set to the maximum of 999.

If you want to increase the number of logs to keep, try to design jobnets on a small scale. If you want to create a large-scale jobnet, decrease the number of logs to keep. Keep this balance in mind when you design a jobnet.

Under most circumstances, design jobnets on a small scale and set a small number of logs to keep, bearing in mind that processing might be performed in parallel. For example, a job might be running during cancellation of registration while its status is being monitored in the Monthly Schedule window.

The processing time for cancellation of registration mentioned above is the time that applies when the registration cancellation mode is asynchronous, as described in *6.1.7 Changing the mode in which unregistration or generation management deletes the generations of a jobnet* in the *JP1/Automatic Job Management System 3 Configuration Guide* (for Windows), or *15.1.7 Changing the mode in which unregistration or generation management deletes the generations of a jobnet* in the *JP1/Automatic Job Management System 3 Configuration Guide* (for UNIX). If the registration cancellation mode is synchronous, the processing will take dozens of minutes.

In new installations, the method of deleting generations is set to asynchronous (`yes`) by default. In the case of an upgrade installation, however, the method might have been set to synchronous (`no`) at some point. We recommend that you check the value of the `BACKGROUNDLEAVE` environment setting parameter, and ensure that it specifies asynchronous mode.

If you have upgraded from version 8 or earlier, in the case of a jobnet for which a start condition is set, monitored generations (generations that enter *Now monitoring* status) and execution generations (generations produced when a start condition is satisfied) are managed separately by default. For this reason, setting a large number of logs to keep will have an even greater effect for such jobnets. In this case, the number of records is: *number-of-registered-units x number-of-logs-to-keep x number-of-times-start-condition-is-satisfied-per-schedule*. For a jobnet with a start condition, we recommend that you keep fewer than 10 logs and use the `ajsshow` command to save the execution results as needed. For details about the `ajsshow` command, see `ajsshow` in *3. Commands Used for Normal Operations* in the manual *JP1/Automatic Job Management System 3 Command Reference*. For points to note when increasing the number of logs to keep for jobnets for which start conditions are set after upgrading from version 8 or earlier, see *4.2.3(4) Notes applying when JP1/AJS3 has been upgraded from JP1 Version 8 or an earlier version of JP1/AJS2* in the manual *JP1/Automatic Job Management System 3 Overview*.

## 7.3 Notes on using PC jobs

---

This section gives notes on using PC jobs.

Read this section in conjunction with 2.6.2 *Troubleshooting problems related to standard jobs, HTTP connection jobs, action jobs, and custom jobs* in the manual *JP1/Automatic Job Management System 3 Troubleshooting*, which provides information about what might cause a PC job to fail to start or end abnormally, and cautionary notes about using PC jobs.

- *Avoiding a system resource shortage (in Windows)*

When the agent host is a Windows host, if you execute more than a specific number of jobs concurrently, you might encounter a shortage of a system resource (desktop heap) and an error might occur depending on the system environment. In this case, consider taking the following action:

- Stop sharing the desktop heap with other applications

Change the JP1/AJS3 service account to a user account. By setting the JP1/AJS3 service account to a user account that differs from the accounts of other services and the user account of the logon user, you can use the system without sharing the desktop heap.

- Decrease usage of the desktop heap

You can decrease consumption of the desktop heap by setting the account of the OS user frequently used for job execution to the same account as the JP1/AJS3 service account.

- *Using space characters in an application file name (in Windows)*

When the agent host is a Windows host, if an application file name with an extension associated with a file type contains a space character, check whether the file type is registered correctly in Windows in the File Types page in Explorer, and enclose the application file name with double quotation marks ("").

- *Preventing jobs from entering Failed-to-start status (in Windows)*

When the agent host is a Windows host, and a job enters *Failed to start* status, make sure that the user who started the JP1/AJS3 service has the permissions listed below. When using queueless jobs, make sure that the following permissions have been set for the account of the queueless agent service:

- For the executable file of the job: Read and execute permissions
- For the environment variable file for the job: Read permission
- For the standard input file for the job: Read permission
- For the standard output file for the job: Read and write permissions
- For the standard error output file for the job: Read and write permissions
- For the transfer source file for the job: Read permission

- *Do not include a command that shuts down the OS in a PC job*

Do not execute a PC job that includes a command for shutting down the operating system of the host itself or the agent host. Because Windows does not wait for JP1/AJS3 to stop before it shuts down, there might be damage to the JP1/AJS3 data files if Windows shuts down while JP1/AJS3 is running. If you want to shut down the OS from a job, use a Local Power Control job in conjunction with the JP1/Power Monitor.

To shut down the system manually, use the power control command (`aompwcon` command) of JP1/Power Monitor. If you want to shut down the system manually but do not have JP1/Power Monitor installed, stop the JP1/AJS3 service manually before shutting down the OS.

- *Executing the `jajs_dbbackup` command from a PC job with the backup enhancement function enabled*

If you execute the `jajs_dbbackup` command by specifying an embedded database with a PC job defined, the status of the PC might not change from the following statuses even if the job has been executed. In this case, however, the PC job status will change normally after the `jajs_dbbackup` command terminates.

- Waiting to execute

- Now queuing

To avoid such situations, make sure that the PC job that executes the `jajs_dbbackup` command is defined in the scheduler service that runs on an embedded database that is not to be backed up.

For an overview of the backup enhancement function, see *5.2.5 Backing up and recovering an embedded database by using the backup enhancement function* in the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*.

- *Use file names of 254 bytes or less in jobs (in Windows)*

A job might enter *Failed to start* or *Ended abnormally* status if any of the following file names specified in the job is 255 bytes or longer:

- The executable file of the job
- The environment variable file for the job
- The standard input file for the job
- The standard output file for the job
- The standard error output file for the job
- The end judgment file for the job
- The transfer source file for the job
- The transfer destination file for the job

This problem also applies to queueless jobs. Make sure that the above file names have no more than 254 characters.

- *When the executable file name specifies a command or program incompatible with data input from a standard input file*

To execute a job for which a command or program incompatible with data input from a standard input file is specified in the **File name** field of the Define Details dialog box, you specify `CON` in the **Standard input**, **Standard output**, and **Standard error** fields. In this case, JP1/AJS3 will not read data from standard input or acquire any data output to standard output and standard error output, and the job's execution results do not appear in the Execution Result Details dialog box of JP1/AJS3 - View. The `timeout` command is one example of a command that does not support data input from the standard input file.

- *Put the shortcut files for necessary programs into the Startup folder of Windows*

When you put the shortcut files for necessary programs into the Startup folder of Windows, put it in the following folder.

`%ALLUSERSPROFILE%\Microsoft\Windows\Start Menu\Programs\StartUp`

The default location of `%ALLUSERSPROFILE%` is `system-drive\ProgramData`.

## 7.4 Notes on using Unix jobs

---

This section gives notes on using Unix jobs.

Read this section in conjunction with *2.6.2 Troubleshooting problems related to standard jobs, HTTP connection jobs, action jobs, and custom jobs* in the manual *JP1/Automatic Job Management System 3 Troubleshooting*, which provides information about what might cause a Unix job to fail to start or end abnormally, and cautionary notes about using Unix jobs.

- *Return codes that can be set by jobs executed in UNIX*

Jobs executed in UNIX can set return code values from 0 to 255. When a job fails to start, or when acquisition of standard output data or standard error output data fails, the return code is set to -1.

- *Preventing jobs from entering Failed-to-start status (in UNIX)*

When the agent host is a UNIX host, a job might enter *Failed to start* status. To prevent this, make sure that the user who starts the JP1/AJS3 service and the OS user who executes the job have read and write permissions for the following files and directories:

- Standard output file for the job
- Standard error output file for the job
- Work directory at agent process execution
- Work path specified in the job's detailed definition<sup>#1</sup>
- Home directory of the OS user who executed the job<sup>#1</sup>
- Log files used for job execution control<sup>#2</sup>

#1

The work path specified in the detailed definition of the job is the current directory at the time the job is executed. If you do not specify a work path, the home directory of the OS user is assumed. If no home directory is defined for the OS user, root (/) is assumed.

#2

For details about the log files used in job execution control, see *1.2.5 List of log files and directories* in the manual *JP1/Automatic Job Management System 3 Troubleshooting*.

- *Resource limits when Unix jobs are executed*

In JP1/AJS3 for UNIX, the resource limits in effect at JP1/AJS3 startup apply when a job is executed. To apply resource limits, set them for the root user who starts JP1/AJS3. However, if you specify a limit in the job to be executed, the value specified in the job takes effect. For details, see *20.5 Setting up the job execution environment* in the *JP1/Automatic Job Management System 3 Configuration Guide*.

The following is an example of changing file size limits:

1. In the login profile for the root user (usually [/ .profile] (\$HOME/ .profile)), include the setting below:  
For *fsize*, specify the required file size. If you do not want to impose a limit, specify *unlimited*.  

```
ulimit -f fsize
```
2. Log in as the root user.
3. From the root account, start the JP1/AJS3 service.  
The *fsize* value takes effect.

### Cautionary note

In AIX and Linux, the resource limits you define in the resource settings file for the OS (AIX: /etc/security/limits, Linux: /etc/security/limits.conf) apply only to processes started by users



using the `login` command over a telnet connection or similar. Because jobs started by JP1/AJS3 take the form of processes started by a service, the settings in the file do not apply.

- *Precautions applying when the JP1/AJS3 service starts automatically*

In a system where the JP1/AJS3 service starts automatically, the login profile of the root user is not loaded. For this reason, even if you change the resource limits in the login profile of the root user, different limits might apply when you log in manually and start the JP1/AJS3 service. In this case, set resource limits in the environment setting parameters of the job execution environment. For details about these parameters, see *20.5 Setting up the job execution environment* in the *JP1/Automatic Job Management System 3 Configuration Guide*.

You can also write resource limits into the automatic start script (`/etc/opt/jplajs2/jajs_start`) of JP1/AJS3. If you do so, test your settings thoroughly before using the system.

Note that jobs might be executed under a group ID that differs from the group ID set for the root user at login. For details, see *5.4.12 Group ID for job execution (UNIX only)* in the manual *JP1/Automatic Job Management System 3 Overview*.

- *Explicitly declaring the umask value for a job process in the profile of the job execution user or the executing shell*

Job processes started by the JP1/AJS3 service adopt the umask value of the shell that started the JP1/AJS3 service, unless a umask value is explicitly declared.

If you specify the value of umask in a profile of the job-execution user such as `/etc/profile` and `$/HOME/.profile`, the value in the profile overwrites the value of umask in the shell that started the JP1/AJS3 service.

You must explicitly specify the value of umask for the job process in a profile of the job execution user or the shell. To change the value of umask for the standard output file and the standard error output file that are specified in a job definition, see *5.4.7 Value of umask set for the standard output file and the standard error output file (UNIX only)* in the manual *JP1/Automatic Job Management System 3 Overview*.

- *Using a value other than /bin:/usr/bin as the PATH environment variable*

When you start a job from JP1/AJS3, JP1/AJS3 explicitly specifies `/bin:/usr/bin` for the PATH environment variable. If you want to specify a different value, define it in the command or script file that you specify when defining the job, or in the local login script.

- *Executing a user program that requires a terminal as a job*

In the UNIX version of JP1/AJS3, a user program that requires a terminal might not operate correctly if executed as a job (the job might end abnormally).

- *Changing OS user information for the login shell or other feature when using queueless jobs*

If OS user information for the login shell or other feature is changed when you use a queueless job, use either of the following methods to clear the cache:

- Execute the `ajsqlalter` command with the `-r` option specified.
- Restart the queueless agent service.

- *Registering or changing an OS user*

While a job is running, do not use the `passwd` command with administrator privileges to register or update the OS user associated with the job. Register or update the OS user information before you run the job.

- *Executing the jajs\_dbbackup command from a Unix job with the backup enhancement function enabled*

If you execute the `jajs_dbbackup` command by specifying an embedded database with a Unix job defined, the status of the Unix might not change from the following statuses even if the job has been executed. In this case, however, the Unix job status will change normally after the `jajs_dbbackup` command terminates.

- Waiting to execute
- Now queuing

To avoid such situations, make sure that the Unix job that executes the `jajs_dbbackup` command is defined in the scheduler service that runs on an embedded database that is not to be backed up.

For an overview of the backup enhancement function, see *5.2.5 Backing up and recovering an embedded database by using the backup enhancement function* in the *JPI/Automatic Job Management System 3 System Design (Configuration) Guide*.

## 7.5 Notes on using a recovery unit

The following provides precautions for defining a recovery unit.

- Even if a recovery unit ends normally, the jobnet containing the recovery unit is treated as having ended abnormally.
- Only a recovery unit can be defined as the succeeding unit to another recovery unit. If you define a normal unit as the succeeding unit to a recovery unit, the normal unit is not executed.
- A unit defined as the succeeding unit of a recovery unit is not executed when the jobnet executes normally. It is executed only after the preceding recovery unit is executed due to an abnormal end, and the recovery unit ends normally.
- A warning issued by a recovery unit or the abnormal status of a recovery unit does not affect the status of upper-level jobnets. However, if a recovery jobnet starts or ends later than expected, the statuses of upper-level jobnets are set as delayed.
- The following table shows the status of the preceding unit and whether a recovery unit can be executed.

**Table 7–1: Whether a recovery unit can be executed depending on the status of the preceding unit**

| No. | Status of preceding unit | Whether a recovery unit can be executed |
|-----|--------------------------|---|
| 1   | Not sched. to exe.       | N                                       |
| 2   | Wait for start time      | N                                       |
| 3   | Wait for prev. to end    | N                                       |
| 4   | Being held               | N                                       |
| 5   | Waiting to execute       | N                                       |
| 6   | Now queuing              | N                                       |
| 7   | Not executed + Ended     | N                                       |
| 8   | unexec-W                 | N                                       |
| 9   | Bypassed                 | N                                       |
| 10  | Now running              | N                                       |
| 11  | Running + Abend          | N                                       |
| 12  | Running + Warning        | N                                       |
| 13  | Ended normally           | N                                       |
| 14  | Normal end + False       | N                                       |
| 15  | Ended with warning       | N                                       |
| 16  | Ended abnormally         | Y <sup>#</sup>                          |
| 17  | abnormal-WR              | Y <sup>#</sup>                          |
| 18  | Skipped so not exe.      | N                                       |
| 19  | Invalid exe. seq.        | Y <sup>#</sup>                          |
| 20  | Interrupted              | Y <sup>#</sup>                          |
| 21  | Killed                   | Y <sup>#</sup>                          |
| 22  | kill-WR                  | Y <sup>#</sup>                          |

| No. | Status of preceding unit | Whether a recovery unit can be executed |
|-----|--------------------------|---|
| 23  | Failed to start          | Y <sup>#</sup>                          |
| 24  | fail-WR                  | Y <sup>#</sup>                          |
| 25  | Unknown end status       | N                                       |
| 26  | unknown-WR               | N                                       |
| 27  | Shutdown                 | N                                       |
| 28  | Wait for start cond.     | --                                      |
| 29  | Now monitoring           | --                                      |
| 30  | Unmonitored + Ended      | --                                      |
| 31  | Monitor terminated       | --                                      |
| 32  | Interrupted monitoring   | --                                      |
| 33  | Monitor-end normal       | --                                      |

Legend:

Y: The recovery unit can be executed.

N: The recovery unit cannot be executed.

--: No status exists as the status of the preceding unit.

#

If the JP1/AJS3 service is restarted in warm-start mode or disaster-recovery-start mode, or if the root jobnet is interrupted or killed, the recovery unit will not be executed even if the preceding job ends with an abnormal status.

## 7.6 Notes on using event jobs

---

This section gives notes on using event jobs.

- Event jobs do not support operations that use execution agent groups. If an event job without a specified execution agent is included in a root jobnet or nested jobnet for which an execution agent group is specified, JP1/AJS3 will attempt to use the agent group specified for the jobnet as the execution agent for the event job. If an execution agent with the same name as the agent group exists, the event job will be executed by that execution agent. If there is no execution agent with the same name as the agent group, an error occurs and the following message is output to the integrated trace log: `KAVT0403-E The specified agent is not defined in the job execution environment. (host=exec-agent, maintenance-information)`. Therefore, if you want to specify an execution agent group for a root jobnet or nested jobnet, make sure that an execution agent is explicitly specified for the event job in the jobnet.
- You cannot specify the name of an execution agent group in the **Exec-agent** field of the detailed definition of an event job.
- JP1/AJS3 may not correctly detect events issued by the JP1/AJS3 program itself, including JP1 events, events logged to the Windows event log or syslog, or events recorded in HNTRLlib and Scheduler log files. For the notes on monitoring the information issued by JP1/AJS3 and details about the targets of monitoring by an event job, see [7.6.9 Monitoring events or messages issued by JP1/AJS3](#).
- For an event job waiting for a start condition to be satisfied, if JP1/AJS3 - Manager stops during start condition monitoring, you can resume event monitoring after JP1/AJS3 - Manager restarts. If multiple events were being monitored for a start condition, you can hold the reception information of the events that satisfy the condition even after JP1/AJS3 - Manager restarts.
- When an event job is defined at the beginning of a jobnet, the jobnet is executed after the condition is satisfied, in the same way as a jobnet with a start condition. A jobnet with a start condition stays in *Wait for start cond.* status while event reception is being monitored. A jobnet with an event job defined at the beginning is kept in *Now running* status while event reception is being monitored. When you define an event job at the beginning of a jobnet, we recommend that you use the event job to wait for events you expect to occur.
- When you monitor multiple events, you can improve performance by using regular expressions to create an event job that monitors all of them in a batch.

For example, if you use the Receive JP1 Event job to monitor JP1 events in the form of messages KAVS0272-E and KAVS0273-E with event ID 00004131, group the event jobs by specifying only the event ID or by specifying 00004131 for the event ID and KAVS for the message. You can identify the contents of events by using passing information.

- You can use regular expressions to define the following items in JP1 event reception monitoring jobs, log file monitoring jobs, and Windows event log monitoring jobs:
  - User name (JP1 event reception monitoring jobs)
  - Group name (JP1 event reception monitoring jobs)
  - Host name (JP1 event reception monitoring jobs)
  - Message (JP1 event reception monitoring jobs)
  - Event details (JP1 event reception monitoring jobs)
  - Data to be trapped (log file monitoring jobs)
  - Data lines except log (log file monitoring jobs)
  - Explanation (Windows event log monitoring jobs)

Frequently using a regular expression that matches all characters ( . \* ) can significantly increase the time needed to compare detected events and monitoring conditions, thereby delaying event detection. Use this expression ( . \* ) only when necessary.

Note also that the items defined using regular expressions are case sensitive. Use regular expressions depending on the events to be monitored.

- If a timeout period is specified for an event job, the counting is done by the execution host. Therefore, when event monitoring resumes after the execution host is restarted due to a power failure or similar condition, counting restarts upon the restarting of the execution host.

Note that you can verify the restart of counting and the restart time in the message KAVT0603-W Elapsed time since *restart-time* is used for time-out due to temporary interruption of monitoring. that is output in the event job execution result details.

If you want to stop monitoring at an absolute time regardless of the status of the agent host, define an event job as a jobnet start condition, and then specify the effective range of the start condition by using an absolute time. For details about a start condition, see 3.4 *Defining a start condition* in the manual *JP1/Automatic Job Management System 3 Overview*.

- In Windows, JP1/AJS3 event jobs do not depend on the settings of the JP1 users that run them. Rather, they depend on the account permissions for the JP1/AJS3 service. Therefore, when using file monitoring jobs, which are a type of event job, make sure that write permissions for files and folders that are monitored by file monitoring jobs are assigned to JP1/AJS3 service accounts.

If a required permission is not granted, the following will occur:

- Monitoring Files jobs end abnormally.
  - Events do not occur.
- JP1/AJS3 event jobs are independent of the permissions set for JP1 users defined in the JP1/Base environment settings and in the respective jobs.
  - When using event jobs, system load or temporary network failure might cause a time lag between the job execution time and the time that the execution agent actually starts monitoring the events. The system detects only events that occur after event monitoring begins. Therefore, you must provide some leeway when executing an event job, to allow the system to be ready and monitoring when a target event occurs.
  - When you execute event jobs (including those with a start condition), definition data for the executed event jobs and information about events that satisfy monitoring conditions is transmitted between processes such as the event/action control manager and the event/action control agent. If transmission fails due to some problem such as a temporary network error or because the remote process is busy, the information is saved to a file and transmission is attempted again after a predetermined interval. In JP1/AJS3, this is referred to as an *unreported information file*. At successful re-transmission, the information is deleted.
  - When using JP1/AJS3, in the API settings file for the JP1/Base event service, set *keep-alive* for the communication type of the *server* parameter. Setting *close* as the communication type can result in operational problems, including JP1/AJS3 being unable to issue JP1 events at startup, causing message KAVT1040-E to be output to the integrated trace log. This in turn renders Receive JP1 Event jobs, Monitoring Log Files jobs, Monitoring Event Log jobs, and Send JP1 Event jobs unable to detect events, and causes Send JP1 Event jobs to end abnormally. For details about the API settings file and how to set parameters, see the *JP1/Base User's Guide*.
  - In a manager/agent configuration, if the event/action control manager and the event/action control agent are unable to communicate due to a network error or other problem, inconsistencies in event job monitoring (including event jobs defined as start conditions) can occur if you perform any of the operations listed below. In this case, the event job will continue monitoring on the agent despite having ended on the manager.
    - Killing an event job
    - Killing a jobnet that has a start condition
    - Changing the status of an event job to *Ended*

These operations can result in problems such as failure to restart the event job where the inconsistency occurred, and delays to processing of other normal event jobs.

For this reason, if you perform one of the above operations in a system affected by a network error or similar problem, execute the `jpomanjobshow` command on the manager host, and the `jpogtjobshow` command on the agent host. Compare the results of the commands to see whether any event jobs that have ended on the manager are still monitoring on the agent.

If you discover an event job that is in *Now monitoring* status only on the agent host, restart the JP1/AJS3 service on the agent host, and then terminate the event job that is still monitoring on the agent.

## 7.6.1 Notes on the Receive JP1 Event job

The following provides precautions (items you must know in advance) for using the Receive JP1 Event job.

JP1 events are events that are managed by JP1/Base and issued when events occur in JP1 programs. JP1 events contain information about various event levels, ranging from errors and warnings to reports and messages. You can execute a different succeeding job for each event level, or execute the succeeding job only when a specific message is received. You can use regular expressions to extract parts of messages or detailed information from within JP1 events and pass the information to succeeding jobs.

Examples of jobnets that use the Receive JP1 Event job are as follows:

- Execute the succeeding job when an error occurs in a JP1 program or a warning is reported.
- Execute the succeeding job at the completion of all processing in situations where processing is executed by two or more JP1 programs.
- Execute the Send JP1 Event job when a jobnet executed by JP1/AJS3 - Manager in another host ends, receive the sent JP1 event in another jobnet, and execute the succeeding job.
- Use the JP1/Base event conversion facility to execute the succeeding job when a non-JP1 application is terminated. For details about the event conversion facility, see the *JP1/Base User's Guide*.

### (1) Notes on detecting JP1 events

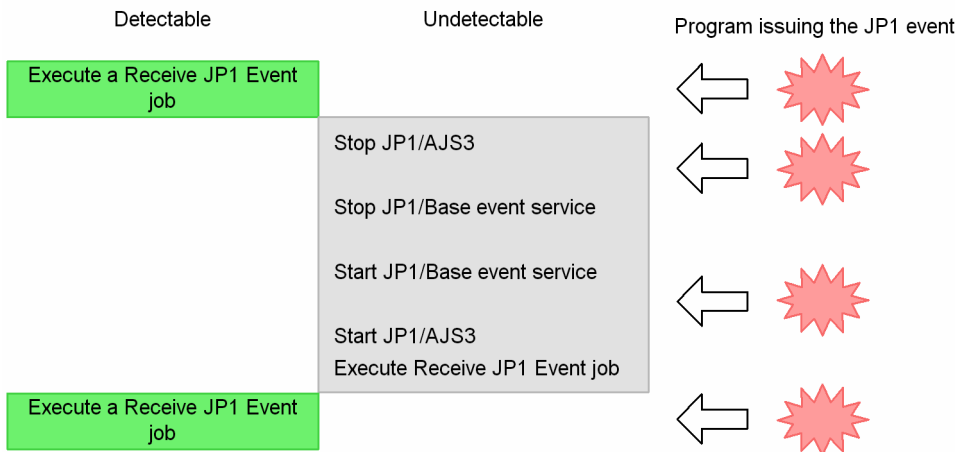
The Receive JP1 Event job can only monitor events that occur after it begins monitoring. Therefore, JP1/AJS3 does not detect the following JP1 events:

- JP1 events that occur while JP1/AJS3 is inactive
- JP1 events that occur between the time when JP1/AJS3 starts and the job begins monitoring

As the JP1 events to be monitored by the Receive JP1 Event job, choose JP1 events that will be issued after the JP1/AJS3 event jobs enter monitoring status.

The following figure shows the timing under which detection cannot take place for a Receive JP1 Event job.

Figure 7–1: Cases where a Receive JP1 Event job cannot detect events



## (2) Options available with the Receive JP1 Event job

With the Receive JP1 Event job, you can set an option that governs whether to monitor reception of JP1 events that occur before the job starts (and before monitoring of JP1 events starts). This is called the *Find events prior to exec.* option (the option for finding events before starting event monitoring). This option takes effect when you specify an event ID and a search range that dictates how many minutes before the start of the job are to be included in the scope of the Receive JP1 Event job. You can specify a value from 1 to 720 (in minutes). The reference time for this option is based on the local time of the host on which the Receive JP1 Event job is executed.

If you do not use this option, the job only monitors reception of JP1 events that occur after JP1 event monitoring starts.

The following cautionary notes apply when this option is used:

- The greater the search range of the Find events prior to exec. option (the number of minutes prior to the Receive JP1 Event job starting that are included in its scope), the longer it takes to search for events that occurred prior to executing the Receive JP1 Event job. The search will also take longer if a large number of JP1 events occurred in the specified search range. Therefore, we recommend that you specify the smallest possible value (at most 10 minutes) as the search range.
- The receive JP1 Event job acquires JP1 events from the JP1/Base event database. Therefore, even if JP1 events are generated within the event search range defined in a Receive JP1 Event job, the Receive JP1 Event job will not detect the events if the event database is changed and the JP1 events to be searched for are deleted from the event database. If there is a possibility that JP1 events to be searched for will be deleted from the event database, in the Find events prior to exec. option, specify the shortest possible time for the search range defined in the Receive JP1 Event job, and configure the operation settings so that monitored events are generated within the search range. If JP1 events to be searched for continue to be deleted from the event database even after you specify the shortest possible time, increase the size of the event database. For details about the event database, see the *JP1/Base User's Guide*.
- For the JP1 event reception monitoring job that uses the *Find events prior to exec.* option, if there are many JP1 events in the search range, the following problems might occur:
  - CPU usage increases.
  - The JP1 event reception monitoring job requires a long time to terminate.
  - The JP1 event reception monitoring job ends abnormally.

Accordingly, use the Find events prior to exec. option only when monitoring a limited number of specific JP1 events. Do not use the Find events prior to exec. option when monitoring events that can be generated in large quantities or that can be generated repeatedly. If searching many JP1 events is necessary, specify detailed monitoring conditions for the Receive JP1 Event job to filter the JP1 events that will be searched or else narrow the search range when you



specify the Find events prior to exec. option. If searching many JP1 events is necessary, specify detailed monitoring conditions for the JP1 event reception monitoring job to filter the JP1 events that will be searched. Alternatively, reduce the search range specified when you specify the *Find events prior to exec.* option.

If you concurrently execute multiple JP1 event reception monitoring jobs with the *Find events prior to exec.* option specified, the search range might be expanded<sup>#</sup> if the search conditions are joined. In this case, the expanded search range might contain more JP1 events than the case when JP1 event reception monitoring jobs are executed separately. Therefore, when you concurrently execute multiple JP1 event reception monitoring jobs with the *Find events prior to exec.* option specified, we recommend that you use a moderate range of event IDs in the search range. For example, specify event IDs in the range from 0 to 1FFF for the monitoring condition.

#

For example, if search ranges are joined when two JP1 event reception monitoring jobs specified as follows are executed, the resulting search range contains event IDs in the range from 1 to 7FFFFFFF:

- Monitoring condition specified for JP1 event reception monitoring job A

Event ID: 1

- Monitoring condition specified for JP1 event reception monitoring job B

Event ID: 7FFFFFFF

- We recommend that you use the Find events prior to exec. option from within a start condition job. You can still use this option from within other kinds of event jobs, but if the Find events prior to exec. option is specified in these jobs, the same event might satisfy a condition multiple times. Keep the usage method in mind when using a Receive JP1 Event job as an event job.

The following shows an example when the same JP1 event satisfies a monitored condition multiple times.

#### Example

Suppose that the Receive JP1 Event job is registered as an event job with the following conditions in the jobnet recv.

- Start condition:  
The jobnet recv is set to be started by an Interval Control job at 9:00 and 9:10.
- Search range of Find events prior to exec. option:  
30 (minutes)
- Event ID:  
111
- Issuance conditions of the JP1 event (event ID: 111):  
8:20 and 8:50

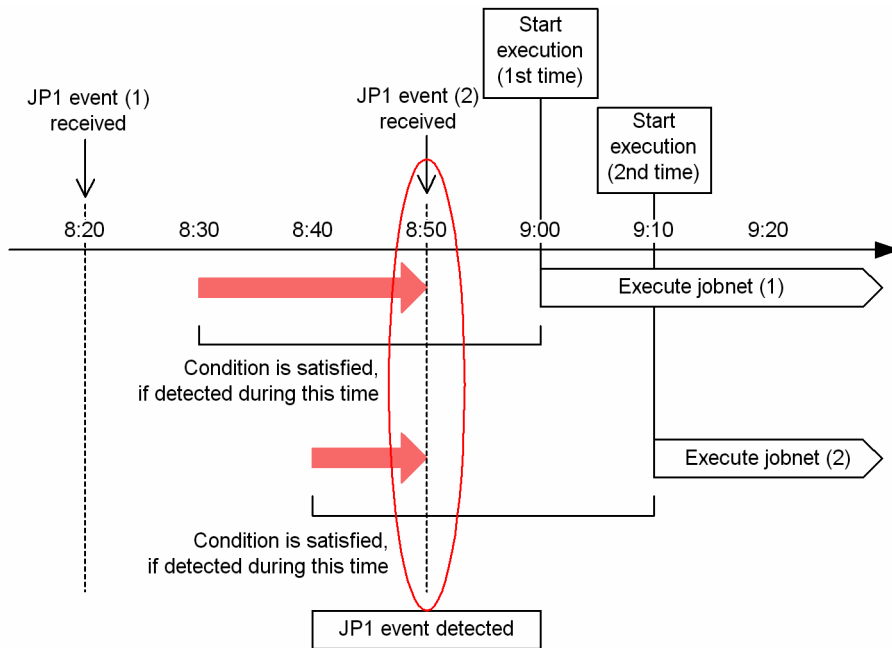
#### Operation

1. The Interval Control job starts the jobnet recv at 9:00.
2. The Receive JP1 Event job searches for JP1 events that occurred from 8:30 according to the conditions of the Find events prior to exec. option.
3. The Interval Control job starts the jobnet recv at 9:10.
4. The Receive JP1 Event job searches for JP1 events that occurred from 8:40 according to the conditions of the Find events prior to exec. option.

#### Result

If the only JP1 event that satisfied the search condition occurred at 8:50, the Receive JP1 Event jobs started at 9:00 and 9:10 both detect the same JP1 event that occurred at 8:50.

Figure 7–2: Example when the same JP1 event is detected more than once



### (3) Notes on defining the Receive JP1 Event job

The following are cautionary notes on defining the Receive JP1 Event job:

- Start the JP1/Base event service before executing a Receive JP1 Event job, and make sure that the API setting of the JP1/Base event service is `keep-alive`. If the JP1/Base event service is not started, the Receive JP1 Event job waits until the event service has started before executing.
- The Receive JP1 Event job cannot receive JP1 events that occur before it enters *Now running* status. Make sure that the JP1 events you want to receive occur after the Receive JP1 Event job enters this status. Alternatively, you can use the `Find events prior to exec.` option to find events that occurred prior to the job entering *Now running* status.
- A JP1 event contains messages and detailed information. If you want to use a Receive JP1 Event job to monitor JP1 events for a specific character string, specify the character string by using a regular expression. If you want to use extended regular expressions, specify the JP1/Base settings in Windows. In UNIX, regardless of the JP1/Base settings, you can use extended regular expressions by using the operating system's syntax for extended regular expressions. For details about configuring Windows to use extended regular expressions, see the explanation about extending the default regular expressions in the *JPI/Base User's Guide*. For details about the regular expressions available in UNIX, see the UNIX documentation.
- If you overuse the expression `.*`, which matches any character or characters, it could take a long time to compare the expression against the JP1 event. For long messages, use `.*` only where necessary. Executing multiple Receive JP1 Event jobs that use the expression `.*` might exponentially increase the time it takes to compare monitoring conditions for each job against the JP1 event, greatly delaying event detection. For this reason, we recommend that you use the JP1/Base event transfer function and filter conditions on a separate host to reduce the number of JP1 events before monitoring, or add monitoring conditions to reduce the frequency with which `.*` is used in comparisons. Note that in UNIX you can reduce the time it takes to compare the expression against the JP1 event if you use the expression `[^ ]*` (matches repeated characters other than spaces) instead of `.*`.
- In the start condition for a Receive JP1 Event job, you must define one or more items. If no item is defined, the Receive JP1 Event job terminates each time a JP1 event occurs on the host that performs event reception monitoring. Because the monitoring condition is satisfied even for a JP1 event issued by JP1/AJS3, the jobnet starts each time a job is executed.

- When the JP1/Base event service on a Windows host receives JP1 events issued by the event service of JP1/SES Version 5 or JP1/AJS Version 5 or earlier, or by the JP1/SES protocol, originating IP addresses are not set in the JP1 events. As a result, in Windows, even when you specify an IP address for the event source in the monitoring conditions of a Receive JP1 Event job, the type of event described above will not satisfy the monitoring conditions.
- Event details of a JP1 event are monitored only when the details in the JP1 event are in text format. If the details contain binary data, the details in the JP1 event are ignored and do not satisfy the monitoring condition. If the JP1 events to be monitored contain binary data, do not specify event details as a monitoring condition.
- The name of the OS user who issued the JP1 event is set in the JP1 event as the name of the event issuer. If you specify a JP1 user name as the event issuer name in the monitoring condition of a Receive JP1 Event job, the Receive JP1 Event job cannot correctly monitor the reception of JP1 events. User names of event issuers specified in the monitoring condition of a Receive JP1 Event job are case sensitive. Specify the correct user names of event issuers in JP1 events.
- Host names of event issuers specified as the monitoring condition of a Receive JP1 Event job are case sensitive when JP1 event reception monitoring is performed. Specify the correct host names in JP1 events (issuing event server names).
- Partial-string matching is used for the items you specify using a regular expression. If you want to use whole-string matching, you must explicitly specify the whole string by using a regular expression.
- The JP1 event information monitored by the Receive JP1 Event job conforms to the specifications of the JP1/Base event service. For details about the JP1 events you can monitor with the Receive JP1 Event job, see the *JP1/Base User's Guide* and the manual for the product that issued the JP1 event.
- In Windows, for the `users` parameter in the event server settings file (`conf`) of the JP1/Base event service, set the user name of the account from which the JP1/AJS3 service is started. By default, this parameter is set to allow anyone to receive JP1 events.
- IP addresses in IPv6 format are not output as the IP addresses of event issuer hosts for JP1 events. For this reason, you cannot specify IPv6 addresses as the IP addresses of event issuers.

## 7.6.2 Notes on the Monitoring Files job

The following provides precautions (items you should know in advance) for using the Monitoring Files job.

Examples of jobnets that use the Monitoring Files job are as follows:

- Monitor the file write time and execute the succeeding job when the file is updated.
- Monitor the files output by applications and the files transferred from other hosts and execute the succeeding job when the files have been created.

The following paragraphs describe the events that are monitored by the Monitoring Files job, how to specify file names, and the options of the Monitoring Files job.

### (1) Events monitored by the Monitoring Files job

The following table lists the events that can be monitored.

Table 7–2: Events monitored by the Monitoring Files job

| Monitored event          | Details   |
|--------------------------|---|
| Create <sup>#1, #2</sup> | Monitors whether a file with the specified name has been created. |

| Monitored event                    | Details  |
|------------------------------------|--|
| Delete <sup>#3</sup>               | Monitors whether a file with the specified name has been deleted.  |
| Change size <sup>#2, #4</sup>      | Monitors whether the size of a file with the specified name has been changed.  |
| Final time write <sup>#2, #4</sup> | Monitors whether a file with the specified name has been updated. The start condition is satisfied when the update time changes. |

#1

If a file with the specified name already exists when monitoring starts, the condition is satisfied when the old file is deleted and a new file is created.

You can use the start monitoring option to specify whether the condition is satisfied if a file with the specified name already exists when monitoring starts. For details about the start monitoring option, see (3) *Options of the Monitoring Files job*.

#2

If the monitoring condition is satisfied, the system performs a *close check* to make sure the monitored file is not being accessed by any process other than the Monitoring files job.

If another process is accessing the file, the condition is judged to be unsatisfied, and another close check is performed when the next monitoring interval occurs. If the file is not being accessed by a process other than the Monitoring files job, the condition is judged to have been satisfied. The close check prevents the job from assuming that the condition is satisfied before the transmission (for example, copying) of a monitored file has been completed.

#3

If a file with the specified name does not exist when monitoring starts, the condition is satisfied when a new file with the specified name is created and then deleted.

#4

If the file with the specified name does not exist when monitoring starts, the condition is satisfied when a new file with the specified name is created and the size of the file or the last write time is changed. Simply creating a file does not satisfy the condition.

You can specify multiple conditions simultaneously. For example, specify **Delete** and **Final time write** if you want to execute the succeeding job when the file is deleted or updated. However, you cannot specify **Change size** and **Final time write** simultaneously.

The following figures show the basic operation of the Monitoring Files job.

### (a) Example with "Create" specified as a monitoring condition

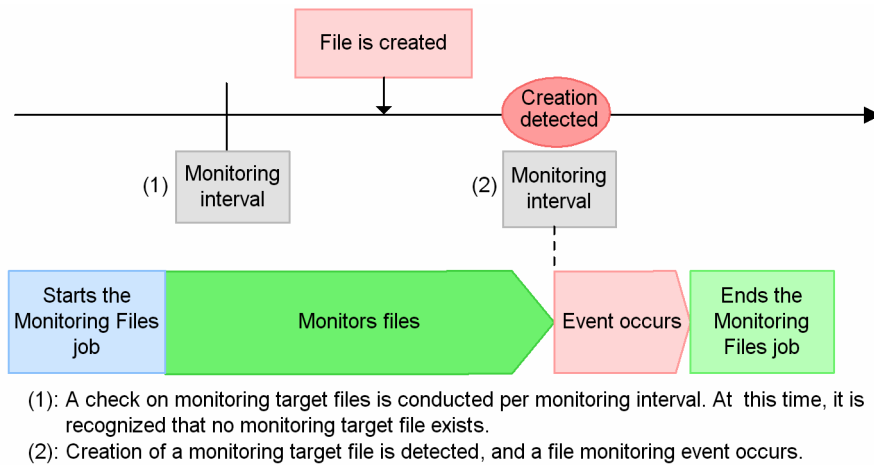
The following figure shows the basic operation of the Monitoring Files job when **Create** is specified as a monitoring condition.

The term *File* in the figure refers to the files being monitored by the Monitoring Files job.

#### ■ When the monitoring target file does not exist at the start of the job

The Monitoring Files job behaves as follows when the monitoring target file does not exist when the job starts.

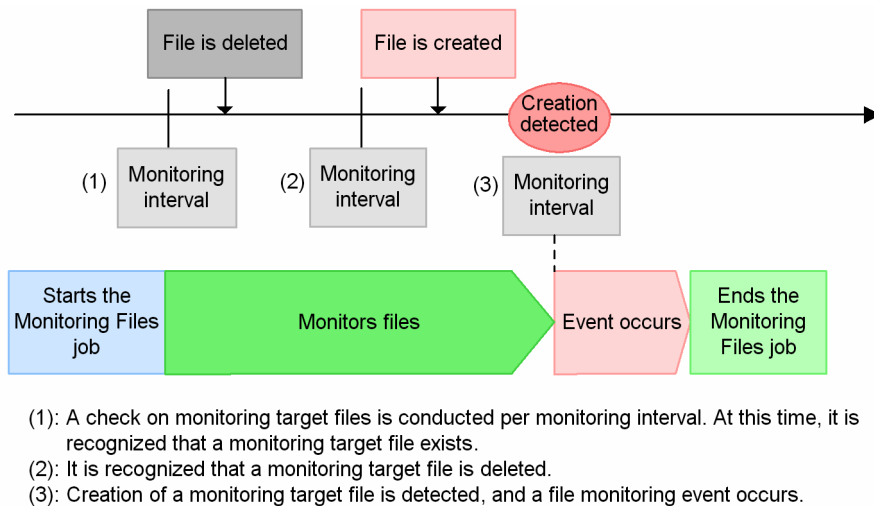
Figure 7–3: When the monitoring target file does not exist at the start of the job



■ **When the monitoring target file exists at the start of the job**

The Monitoring Files job behaves as follows when the monitoring target file already exists when the job starts.

Figure 7–4: When the monitoring target file exists at the start of the job



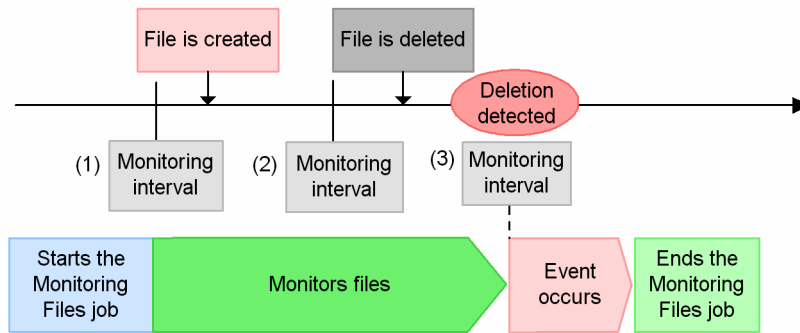
**(b) Example with "Delete", "Change size", or "Final time write" specified as a monitoring condition**

The following figure shows the basic operation of the Monitoring Files job when **Delete**, **Change size**, or **Final time write** is specified as a monitoring condition. This particular example uses **Delete** as the start condition.

■ **When the monitoring target file does not exist at the start of the job**

The Monitoring Files job behaves as follows when the monitoring target file does not exist when the job starts.

Figure 7–5: When the monitoring target file does not exist at the start of the job



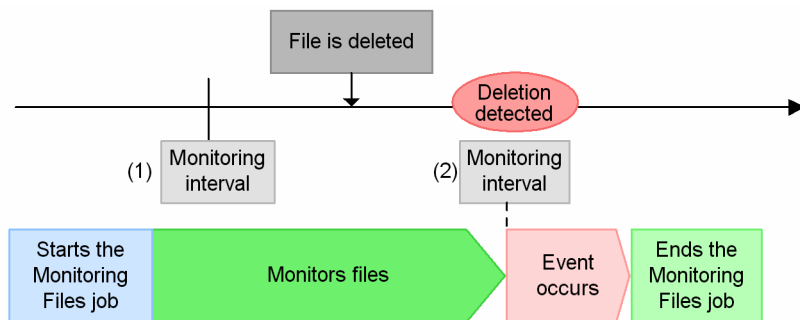
- (1): A check on monitoring target files is conducted per monitoring interval. At this time, it is recognized that a monitoring target file exists.
- (2): It is recognized that a monitoring target file is created.
- (3): Creation of a monitoring target file is detected, and a file monitoring event occurs.

Note:  
 If the monitoring condition is **Change Size** or **Final time write**, read "File is deleted" above as "Change file size" or "Update file rewriting (Change last entry time)".

■ **When the monitoring target file exists at the start of the job**

The Monitoring Files job behaves as follows when the monitoring target file already exists when the job starts.

Figure 7–6: When the monitoring target file exists at the start of the job



- (1): A check on monitoring target files is conducted per monitoring interval. At this time, it is recognized that no monitoring target file exists.
- (2): Deletion of the monitoring target file is detected, and a file monitoring event occurs.

Note:  
 If the monitoring condition is **Change Size** or **Final time write**, read "File is deleted" above as "Change file size" or "Update file rewriting (Change last entry time)".

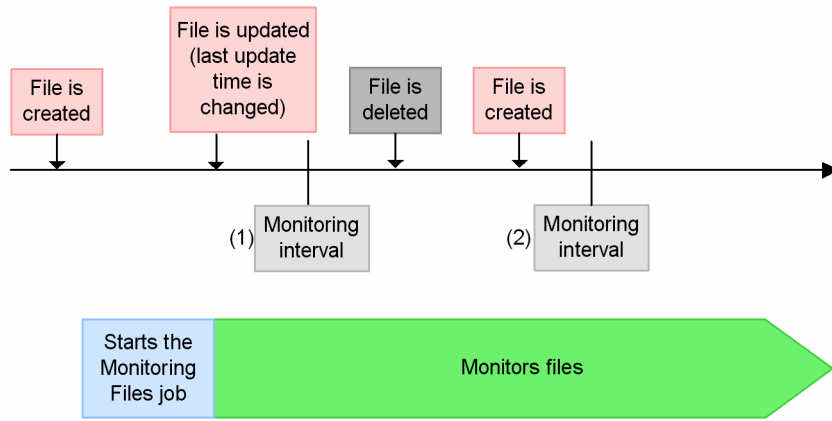
**(c) Example when multiple events occur during a monitoring interval (when the monitoring condition is "Create")**

The following figure shows the basic operation of a Monitoring Files job that uses the **Create** start condition, in a situation where a file is updated multiple times in one monitoring interval.

■ **Example when file creation is not detected**

The Monitoring Files job behaves as follows when file creation is not detected.

Figure 7–7: Example when file creation is not detected

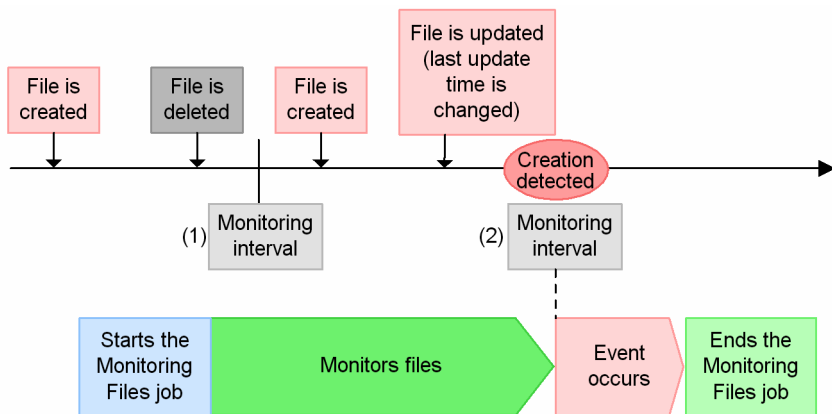


- (1): The monitoring target file is created before starting the Monitoring Files job, so that it does not detect the creation of the monitoring target file at this point. However, it is recognized that the last entry time is changed.
- (2): Although the monitoring target file is created after the file was deleted, the Monitoring Files job compares it to the status of (1). Therefore it does not detect that the monitoring target file is created, and no file monitoring event occurs.

■ Example when file creation is detected

The Monitoring Files job behaves as follows when file creation is detected.

Figure 7–8: Example when file creation is detected



- (1): It is recognized that the monitoring target file created before starting the Monitoring Files job is deleted.
- (2): Although the last update time of the monitoring target file is changed after the file was created, the Monitoring Files job compares it to the status of (1). Therefore it detects that the monitoring target file is created, then the file monitoring event occurs.

## (2) Specifying file names

To specify a file name, you can use an absolute path or wildcard characters consisting of an absolute path and a wildcard (\*). The following table lists examples of using wildcards to specify file names.

Table 7–3: Examples of using wildcards to specify the names of files to be monitored

| Example | Files specified   | Examples of monitored files |
|---------|---|-----------------------------|
| /jpl/*  | All the files in /jpl are monitored. However, in UNIX, a file whose name begins with a period (.) is not monitored. | /jpl/aaa<br>/jpl/aaa.sh     |

| Example      | Files specified   | Examples of monitored files                               |
|--------------|---|---|
| /jpl/.*      | All the files in /jpl whose name begins with a period (.) are monitored.  | /jpl/.aaa   |
| /jpl/aaa*    | All the files in /jpl whose name begins with the character string aaa are monitored.  | /jpl/aaa<br>/jpl/aaabbb<br>/jpl/aaa.sh                    |
| /jpl/*aaa    | All the files in /jpl whose name ends with the character string aaa are monitored.  | /jpl/aaa<br>/jpl/bbbaaa<br>/jpl/bbb.aaa                   |
| /jpl/aaa*bbb | All the files in /jpl whose name begins with the character string aaa and ends with the character string bbb are monitored. | /jpl/aaabbb<br>/jpl/aaaccbbb<br>/jpl/aaa.bbb              |
| /jpl/*aaa*   | All the files in /jpl whose name contains the character string aaa are monitored.   | /jpl/aaa<br>/jpl/bbbaaa<br>/jpl/bbbaacc<br>/jpl/bbbaaa.sh |

#### Note

The file name examples in the above table are for UNIX. When you specify a file name for Windows, it appears in the format `c:\jpl\*`. An error occurs at execution of the Monitoring Files job in the following cases:

- A wildcard is specified in a directory name.
- A relative path is specified.
- The specified file name already exists as a directory (for example, when you specify file name /jpl/aaa, but directory /jpl/aaa already exists).
- The parent directory of the file name including the wildcard does not exist (for example, when you specify file name /jpl/\*, but directory /jpl/ does not exist).

### (3) Options of the Monitoring Files job

You can specify the following two options for the Monitoring Files job:

- Start monitoring option
- Monitoring Files job status passing option

Details of each option are given below.

#### (a) Start monitoring option

You can use the *start monitoring option* to specify whether the monitoring condition of the Monitoring Files job is satisfied if the target file already exists when the job starts executing.

The start monitoring option takes effect when you select **Create** as the monitoring condition. If you do not specify this option, the monitoring condition is not satisfied even if the target file exists.

The following examples illustrate how the Monitoring Files job behaves depending on the setting of the start monitoring option.

#### ■ Operation when the start monitoring option is disabled

The figure below shows the operation of the Monitoring Files job when the start monitoring option is disabled.

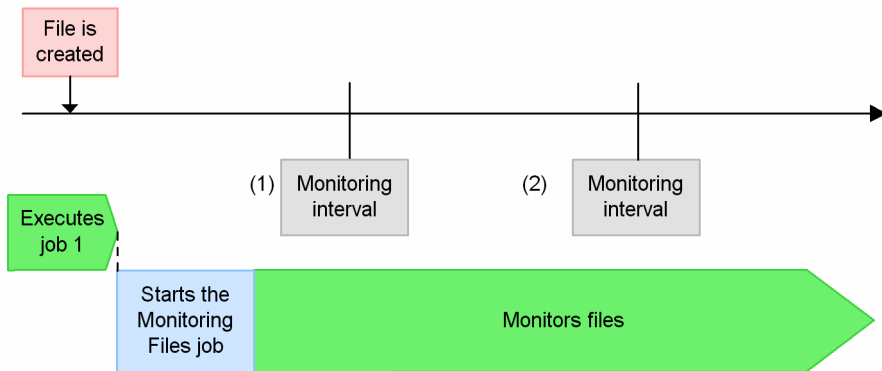


The term *File* in the figure refers to the files being monitored by the Monitoring Files job.

If the monitoring target file is created before the Monitoring Files job is executed

The Monitoring Files job behaves as follows when the monitoring target file is created before the job is executed.

Figure 7–9: Operation when the monitoring target file is created before the Monitoring Files job executes

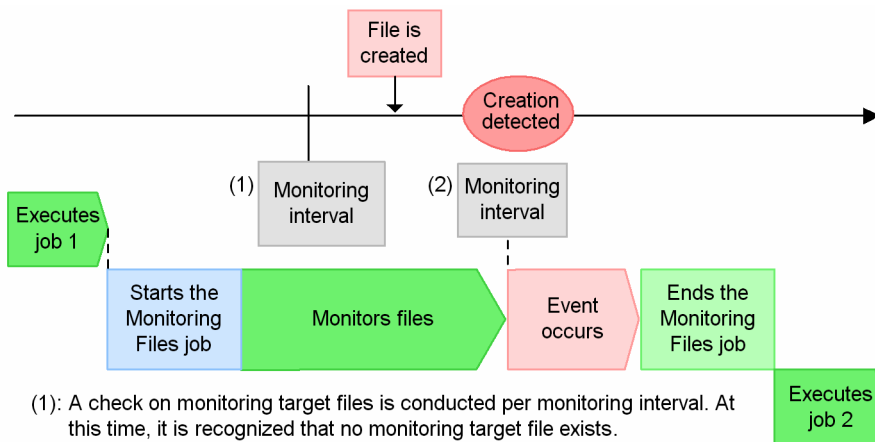


- (1): A check on monitoring target files is conducted per monitoring interval. At this time, it is recognized that a monitoring target file exists. Therefore, no file monitoring event occurs.
- (2): A check on monitoring target files is conducted per monitoring interval. At this time, it is recognized that a monitoring target file exists. Therefore, no file monitoring event occurs. The file is deleted and monitoring remains until the file is recreated.

If the monitoring target file is created after the Monitoring Files job is executed

The Monitoring Files job behaves as follows when the monitoring target file is created after the job is executed.

Figure 7–10: Operation when the monitoring target file is created after the Monitoring Files job executes



- (1): A check on monitoring target files is conducted per monitoring interval. At this time, it is recognized that no monitoring target file exists.
- (2): Creation of a monitoring target file is detected, and a file monitoring event occurs.

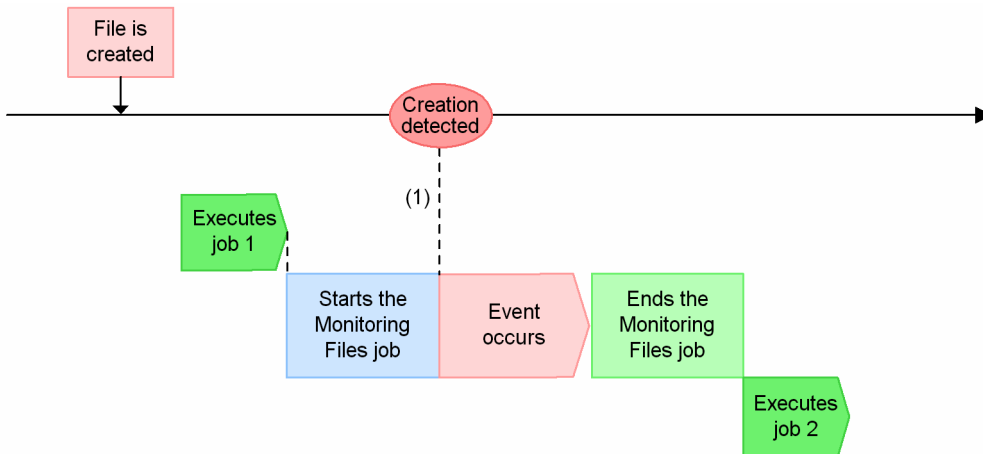
### ■ Operation when the start monitoring option is enabled

The figure below shows the operation of the Monitoring Files job when the start monitoring option is enabled.

If the monitoring target file is created before the Monitoring Files job is executed

The Monitoring Files job behaves as follows when the monitoring target file is created before the job is executed.

Figure 7–11: Operation when the monitoring target file is created before the Monitoring Files job executes

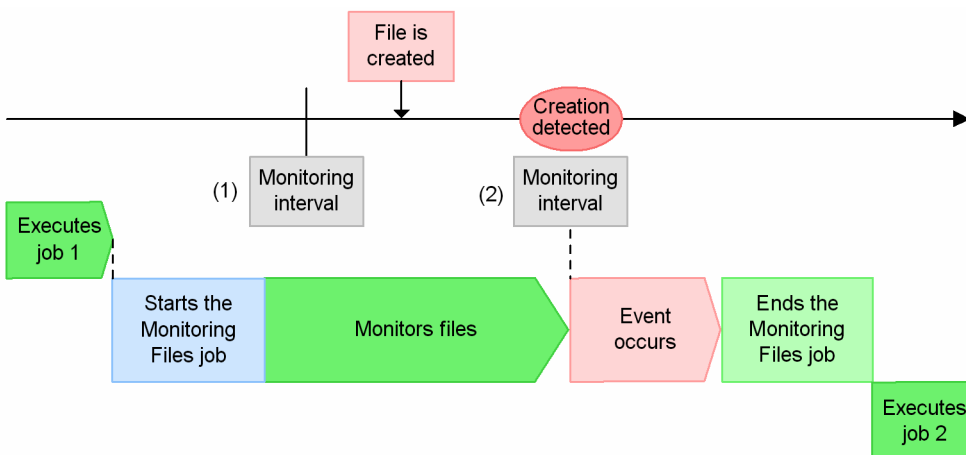


(1): At the start of the Monitoring Files job, a monitoring target file exists. Therefore, the file monitoring event occurs.

If the monitoring target file is created after the Monitoring Files job is executed

The Monitoring Files job behaves as follows when the monitoring target file is created after the job is executed.

Figure 7–12: Operation when the monitoring target file is created after the Monitoring Files job executes



(1): At the start of the Monitoring Files job, no monitoring target file exists. Therefore, check monitoring file per monitoring interval. At this point, it is recognized that no monitoring target file exists.

(2): Creation of a monitoring target file is detected, and a file monitoring event occurs.

See the following for details about how the Monitoring Files job behaves depending on the specification of the start monitoring option.

#### When **Establish for existing files** is specified

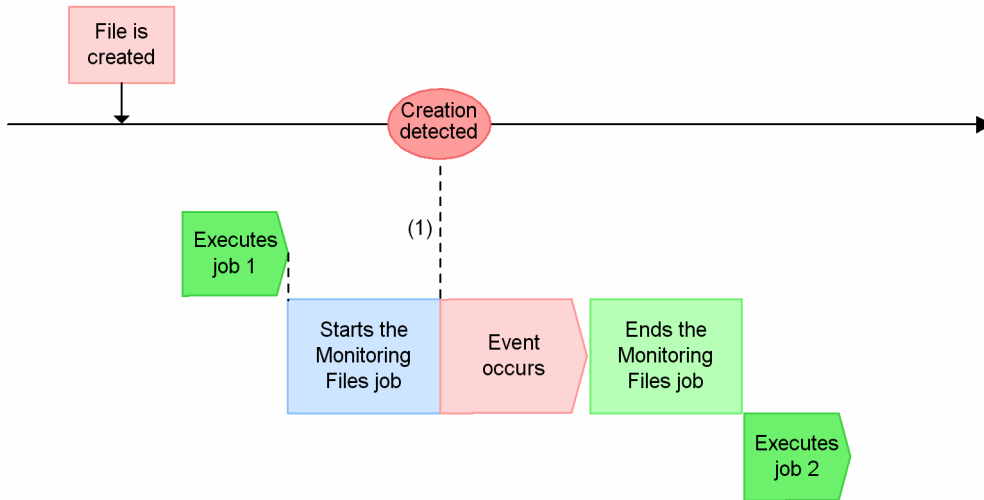
When the Monitoring Files job is executed with **Create** specified as the monitoring condition, if the target file already exists, the monitoring condition is satisfied and the Monitoring Files job terminates normally. A close check ensures that if the file is in use when the Monitoring Files job is executed, the job remains in *Now monitoring* status until the target file is no longer being used.

When the start monitoring option is specified, the monitoring condition is satisfied differently depending on the type of the Monitoring Files job, as follows:

### Monitoring Files job in a jobnet with a monitoring target file name specified

If the monitoring target file exists when the Monitoring Files job is executed, the monitoring condition is satisfied by the start monitoring option. The event occurs immediately, and the Monitoring Files job terminates normally.

Figure 7–13: Monitoring Files job in a jobnet with a monitoring target file name specified



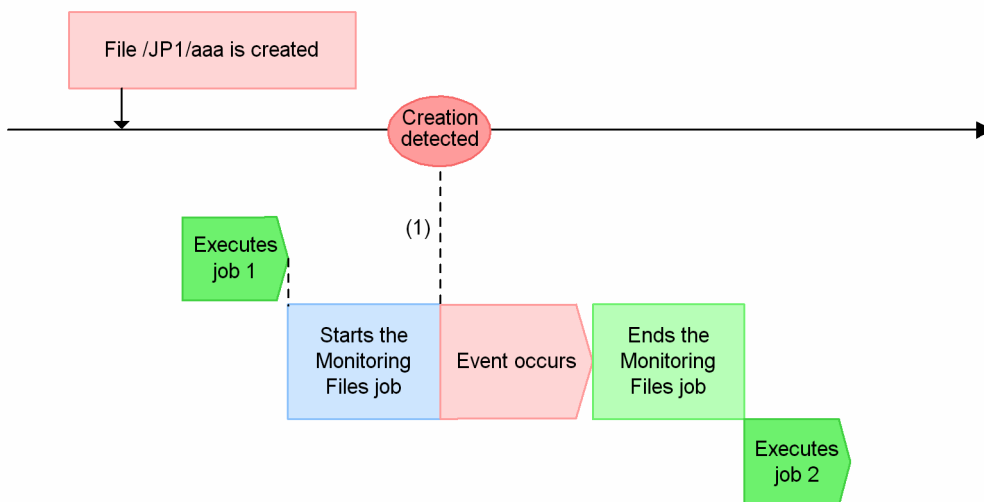
(1): At the start of the Monitoring Files job, a monitoring target file exists. Therefore, the file monitoring event occurs.

### Monitoring Files job in a jobnet with a monitoring target file name specified using a wildcard (\*):

If at least one file exists in the monitoring target directory when the Monitoring Files job is executed, the monitoring condition is satisfied by the start monitoring option. The event occurs immediately, and the Monitoring Files job terminates normally.

Figure 7–14: Monitoring Files job in a jobnet with a monitoring target file name specified using a wildcard (\*)

Monitoring target file name : /jp1/\*



(1): At the start of the Monitoring Files job, a file exists in the /JP1 directory. Therefore, the file monitoring event relating to the file /jp1/aaa occurs.

Supplementary note:

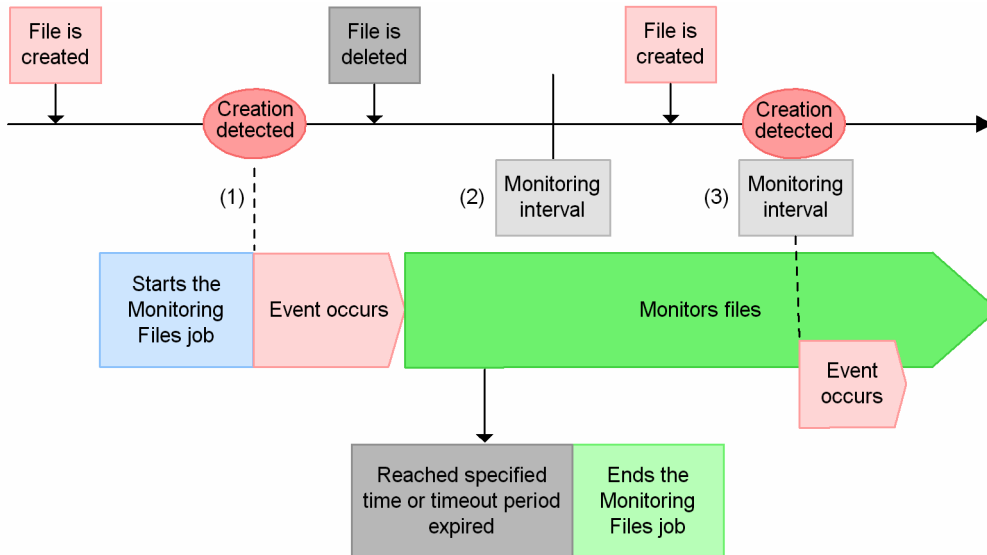
For details about wildcard usage, see (2) *Specifying file names*.

### Monitoring Files job in a start condition with a monitoring target file name specified

If the monitoring target file exists at execution of a Monitoring Files job that serves as a start condition, the monitoring condition is satisfied by the start monitoring option, and the event occurs immediately.

The start monitoring option is disabled once an event occurs. After the event occurs, the Monitoring Files job continues monitoring in the *Now monitoring* status. However, after the option is disabled, a condition is satisfied when a new file is created.

Figure 7–15: Monitoring Files job in a start condition with a monitoring target file name specified



- (1): At the start of the Monitoring Files job, the monitoring target file exists. Therefore, the condition for the event occurrence is satisfied and the file monitoring event occurs.
- (2): It is recognized that the monitoring target file is deleted.
- (3): Creation of a monitoring target file is detected, and a file monitoring event occurs.

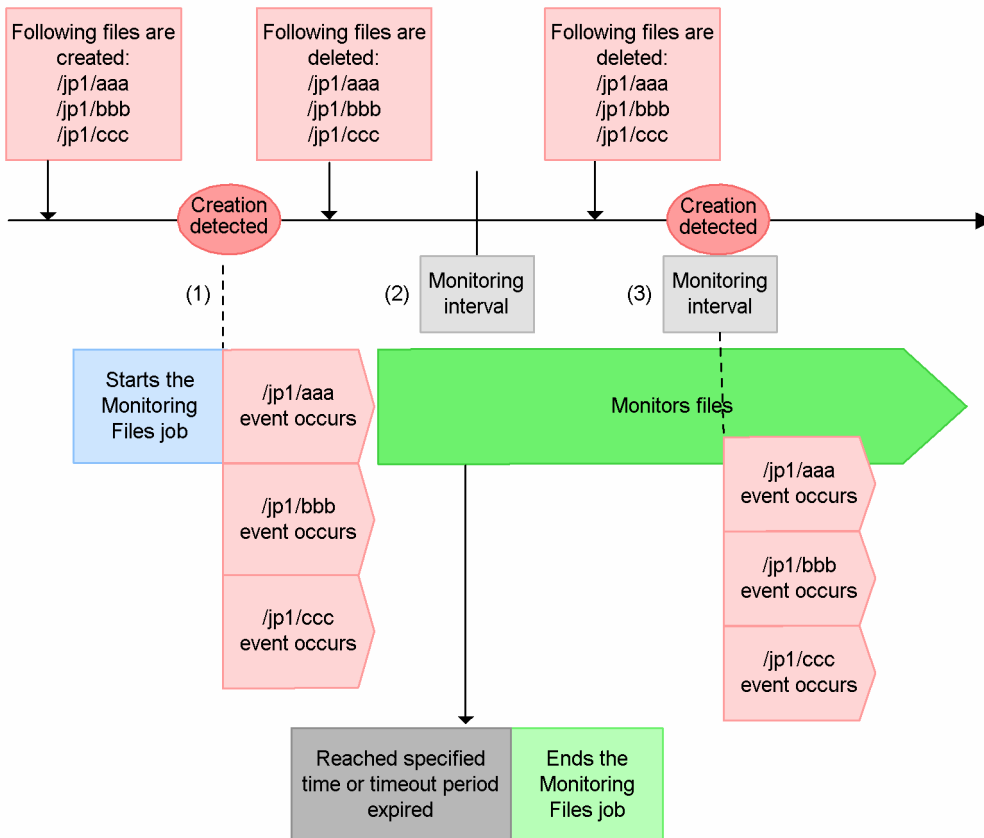
### Monitoring Files job in a start condition with a monitoring target file name specified by a wildcard (\*)

While the Monitoring Files job in the start condition is executed, a monitoring condition is satisfied for any file in the monitoring target directory by the start monitoring option, and an event occurs immediately.

The start monitoring option is disabled after events occur for each of the files. After events occur, the Monitoring Files job continues monitoring in the *Now monitoring* status. However, after the option is disabled, a condition is satisfied when a new file is created.

Figure 7–16: Monitoring Files job in a start condition with a monitoring target file name specified by wildcard (\*)

Monitoring target file name : /jp1/\*



- (1): At the start of the Monitoring Files job, the monitoring target file exists. Therefore, the condition for the event occurrence is satisfied and the file monitoring events occur. All events, /jp1/aaa, /jp1/bbb, and /jp1/ccc, that meet the condition of \*(wild card) occur.
- (2): It is recognized that the monitoring target file is deleted.
- (3): Creation of a monitoring target file is detected, and a file monitoring event occurs. All events, /jp1/aaa, /jp1/bbb, and /jp1/ccc, that meet the condition of \*(wild card) occur.

Supplementary note:

For details about wildcard usage, see (2) *Specifying file names*.

When **Establish at file creation** is specified (default)

When a Monitoring Files job with **Create** specified as the monitoring condition is executed, the monitoring condition is not satisfied even if the monitoring target file exists when the job enters *Now running* status. The Monitoring Files job continues monitoring.

When JP1/AJS3 is used in a cluster system, any Monitoring Files job with the start monitoring option specified will be re-executed after failover of the JP1/AJS3 service. If the Monitoring Files job status passing option is enabled, the job retains the same status as before the failover. If this option is disabled, the start monitoring option applies once again.

For details about how to specify the start monitoring option of the Monitoring Files job, see 12.4.19 *Detailed Definition - [Monitoring Files] dialog box* in the *JP1/Automatic Job Management System 3 Operator's Guide* or 5.2.10 *File monitoring job definition* in the manual *JP1/Automatic Job Management System 3 Command Reference*.

## (b) Monitoring Files job status passing option

You can save the information about the file monitoring performed by the Monitoring Files job as needed, and pass the status to a later instance of the job. If you enable the status passing option, a status passing information file is created for each Monitoring Files job.

For example, suppose that the JP1/AJS3 service stops in a cluster system while a Monitoring Files job is running. If the same Monitoring Files job is executed after the JP1/AJS3 service restarts, it inherits the previous monitoring status. A Monitoring Files job can inherit previous information even in a non-cluster system.

The monitoring status can be passed only if the Monitoring Files job continues running. Whether the monitoring status is passed depends on whether the Monitoring Files job runs continuously, or terminates.

The following table shows the conditions for passing the monitoring status.

Table 7–4: Conditions for passing the monitoring status

| Type of Monitoring Files job              | Restart of JP1/AJS3 service                          | Failover followed by a stop                          | Failover due to system going down |
|---|--|--|-----------------------------------|
| Monitoring Files job in a jobnet          | Not passed because the job terminates <sup>#</sup> . | Not passed because the job terminates <sup>#</sup> . | Passed.                           |
| Monitoring Files job in a start condition | Passed.  | Passed.  | Passed.                           |

#

If a failover that requires the JP1/AJS3 service on the execution agent to be either restarted or stopped occurs, [Table 7-4](#) applies even if the option to continue execution of active event jobs is enabled. This is because the file monitoring jobs are stopped during the failover. As a result, no events can be detected from the time the JP1/AJS service is stopped on the execution agent on which the file monitoring jobs are running until the service is restarted and event monitoring is resumed.

At this time, the messages KAVT2031-E and KAVT2034-W are output to the integrated trace log on the execution agent. If the option to continue execution of active event jobs is enabled, ignore these messages and continue operation. For details about the option to continue execution of active event jobs, see [8.2.1 Continuing the execution of event jobs and custom event jobs if the JP1/AJS3 service stops](#) in the *JP1/Automatic Job Management System 3 Administration Guide*.

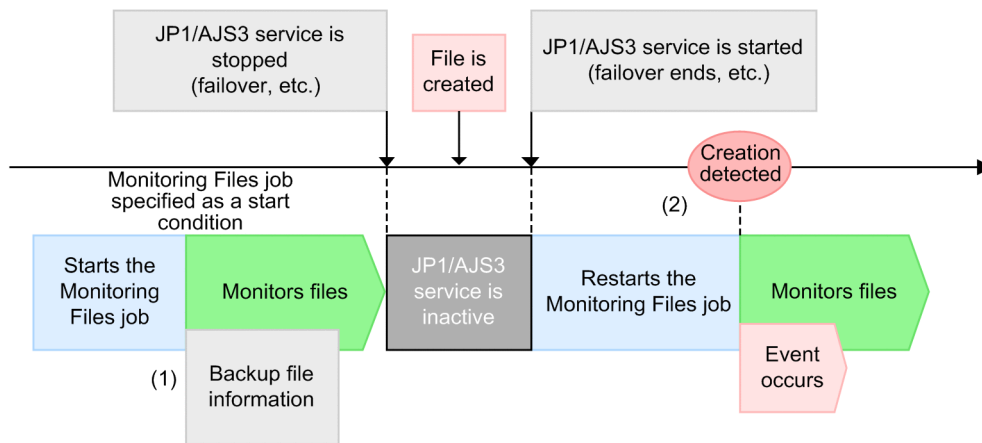
The status passing information file is deleted in the following cases:

- The status passing information file created for each Monitoring Files job is deleted when the Monitoring Files job terminates.
- If you disable status passing after status information has been passed, all the status passing information files are deleted.
- If you start the JP1/AJS3 service with the cold option, all the status passing information files are deleted.

The following figure shows an example of operation when the status passing option of the Monitoring Files job is enabled.

Figure 7–17: Example of operation when the status passing option of the Monitoring Files job is enabled

Condition: **Create** is specified



- (1): When the status passing option is enabled, the information of the monitoring target file held by the file monitoring job is backed up into the passing information storage file.
- (2): When JP1/AJS3 services start and execute the Monitoring Files job before the services end, the status passing information storage file is read. (For the condition to pass the status, see Table 7-4 Conditions for passing the monitoring status.) Therefore, it detects that the monitoring target file is created, then the file monitoring events occur.

The functionality for passing the status of the Monitoring Files job is disabled by default. To enable the functionality, perform the setting procedure for all the hosts and nodes that execute the Monitoring Files job. In a cluster system, both the primary node and the secondary node require the setting. For details about the setting procedure, see 6.3.3 *Setting the status passing option for the file monitoring job* in the *JP1/Automatic Job Management System 3 Configuration Guide* (for Windows hosts) or 15.3.3 *Setting the status passing option for the file monitoring job* in the *JP1/Automatic Job Management System 3 Configuration Guide* (for UNIX hosts).

Note that if you change the setting for monitoring files that exceed 2 gigabytes (large file support) from *yes* to *no*, and a job inherits status passing information for a file larger than 2 gigabytes, the message KAVT2038-E is output to the integrated trace log and to the execution result details, and the job ends abnormally.

#### (4) Notes on defining the Monitoring Files job

Note the following when you define the Monitoring Files job:

- When you execute a Monitoring Files job with a monitoring interval of 1 to 9 seconds, succeeding jobs are not executed immediately (in 1 to 9 seconds) when a file update (event) occurs. When you execute many Monitoring Files jobs, it could take some time before succeeding jobs are executed. In such a case, set a monitoring interval that provides sufficient leeway.  
For details about the formula for estimating the length of a monitoring interval for the Monitoring Files job, see 3.1.6 *Monitoring interval set when using the Monitoring Files job* in the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*.
- A Monitoring Files job might slow down or the CPU usage of the file monitoring process might go up if a generic name, containing a wildcard (\*), is specified for that job as the monitoring-target file name, and if many files exist which match that name. When you use a wildcard character to specify the names of monitoring target files, consider ways to minimize the number of files that match the specified monitoring target file names. We also recommend that you delete unnecessary files to reduce the number of files. Alternatively, use full names to specify the names of monitoring target files.

- As the monitoring targets of the Monitoring Files job, do not specify files in an environment where disks are mounted or unmounted when jobs are executed. If you specify such files as monitoring targets, monitoring might not operate normally or might detect the creation or deletion of files at unexpected times.
- In Windows, the Monitoring Files job can monitor files that do not exceed 2 gigabytes (2,147,483,647 bytes). To monitor larger files (large file), you must set up the environment accordingly. For details about how to perform the settings, see *6.3.16 Enabling monitoring of a large file* in the *JP1/Automatic Job Management System 3 Configuration Guide*. Note that in the UNIX version of JP1/AJS3 11-10 or later, large files can be monitored regardless of whether monitoring of large files is enabled.

If the monitoring of large files is enabled by specifying the relevant settings in Windows, or by upgrading JP1/AJS3 from a version earlier than 11-10 to version 11-10 or later in UNIX, the following notes apply:

- When a user program such as a Unix job references or uses the status passing information attribute `FLSIZE` from a Monitoring Files job, a value that exceeds 2 gigabytes might be passed to the user program. This might cause an overflow in the process that handles file sizes in the user program, and the program will need to be amended or other remedies found.
- The name of a file monitored by the Monitoring Files job must conform to the `LANG` environment variable used for running JP1/AJS3. If you monitor a file whose name uses any other character set, operation is unpredictable.
- If the file being monitored by the Monitoring Files job is also the subject of another event job or log file monitoring by the JP1/Base event service functionality, the target file is considered to be open and the monitoring condition is not satisfied. Therefore, do not set the files as the monitoring target of the Monitoring Files job if they are the target of another event job or subject to log file monitoring by the JP1/Base event service.
- Do not specify as monitoring targets the syslog, device special files, and other system files of the OS. The Monitoring Files job might be unable to identify each of the many unspecified processes accessing the files, processes that access the file frequently, and processes that use the file. Also, do not specify a RAW file as a monitoring target because the Monitoring Files job cannot identify a process that is updating or using such a file.
- If you specify files whose names include network paths<sup>#</sup> as the monitoring targets of a file monitoring job, set `Y` for the `NetworkFilewatch` environment setting parameter. For details about the `NetworkFilewatch` environment setting parameter, see *20.6.2(32) NetworkFilewatch* in the *JP1/Automatic Job Management System 3 Configuration Guide*.

If the names of monitoring-target files include network paths when `Y` is not set for the `NetworkFilewatch` environment setting parameter, the file monitoring job might perform unintended operations, depending on the network conditions.

If the file name you want to specify refers to a file on the local host, specify a file name that does not include a network path.

If the file name you want to specify refers to a file on another host, monitor the file in either of the following ways:

- Install JP1/AJS3 on the host on which the file to be monitored exists, and then monitor the file on that host locally.
- Transfer the monitoring-target file by using FTP or another protocol to a host on which JP1/AJS3 is installed, and then monitor the file.

#

Examples of files whose names include network paths are as follows:

Windows: Files that can be viewed on UNC

UNIX: Files that can be viewed when mounting an NFS

- To specify a network file as a monitoring target in a Windows environment, use a UNC path. You cannot specify a path beginning with a network drive name.
- While you are monitoring a network file, if you need to temporarily stop the server where the monitored file is located, stop the file monitoring job for the monitored file beforehand. If you do not stop the job, it might terminate abnormally when the server is stopped or restarted.



- If `Y` is not set for the `NetworkFilewatch` environment setting parameter when file monitoring jobs are run on UNIX hosts, do not monitor files for which operations (create, update, and view) might be performed by other hosts over the network. If a file is accessed from another host over a network, a condition related to the file might be met while the file is being accessed or might be met more than once by performing only one operation.
- The Monitoring Files job cannot correctly monitor a file that is frequently opened and closed to append data. If such a file is closed when the job performs monitoring, the job might assume that file updating has finished and an event might occur. If you want to monitor such a file, do not use the Monitoring Files job directly. Instead, prepare another file that indicates when file updating has finished, and use the job to monitor the file you prepared.
- Note the following when you are using names containing wildcards (\*) to specify the names of files to be monitored:
  - Monitoring Files events occurring within a given monitoring interval cannot be used to monitor files based on the order in which they were created, updated, or deleted. The *event order option* prevents the order of Monitoring Files events from being altered by factors affecting the communication status. It does not change how the Monitoring Files job behaves.
- Note the following regarding satisfying the monitoring condition for the Monitoring Files job:
  - When you specify **Delete** as a monitoring condition and the monitoring target file does not exist when monitoring starts, the monitoring condition is satisfied after the file is created and then deleted.
  - When **Change size** or **Final time write**, is specified as a monitoring condition and the monitoring target file does not exist when monitoring starts, the monitoring condition is satisfied after the file is created, and then the file size is changed or the last write time is changed.  
**Change size** and **Final time write** monitoring conditions are satisfied after a write is actually made to the target disk. Simply editing a file by using a text editor does not satisfy the monitoring condition.
- If you select the state Do not inherit for the Monitoring Files job status passing option, the information kept by the Monitoring Files job will be lost if the JP1/AJS3 service stops while the Monitoring Files job is active. When you restart the JP1/AJS3 service and begin file monitoring again, the previous file monitoring information will not be inherited.
- The Monitoring Files job checks files at a specified monitoring interval. When an asterisk (\*) is specified, the job checks all the monitored files. If a file update that matches the monitoring condition takes place more than once for one of the monitored files between monitoring times, the Monitoring Files job detects only the last update. Also, if a single file update is detected by a large number of jobs, the next monitoring time might be delayed because events will be issued for all the jobs. If a file update that matches the monitoring condition takes place more than once before the delayed monitoring time arrives, the Monitoring Files job detects only the last update.  
 If you specify a short monitoring interval and there are a large number of jobs to monitor, monitoring might take longer than the monitoring interval.
- When the Monitoring Files job detects that the status of a monitoring target file has changed, the job performs a close check to see if any process has the monitoring target file open. If the Monitoring Files job determines that the file is not open, the job issues an event. If the monitoring target file is open, the job does not issue an event, and enters *Now monitoring* status again. The Monitoring Files job, after the monitoring interval, checks if any process has the monitoring target file open again. In short, as long as the monitoring target file is open, the Monitoring Files job does not issue an event even if it detects that the status of the monitoring target file has changed. Note that, if you monitor files shared by using protocol other than SMB (for example, NFS) in a Windows environment or monitor files via a network in a UNIX environment, it is impossible to check whether a file has been opened by a process on another host (a host other than the agent host running the applicable job). Therefore, an event is issued even when a file being monitored is opened by a process on another host.
- If you select the state Inherit for the Monitoring Files job status passing option, and execute a saved JP1/AJS3 unit or use the environment under the installation directory for a backup copy of JP1/AJS3, you need to have cold-started the JP1/AJS3 service. If you execute a Monitoring Files job of the backup JP1/AJS3 without cold-starting the JP1/AJS3 service, monitoring starts from the monitoring status in effect before the unit was saved.

- If an information file or folder is affected by some error that prevents files that store passing statuses from being created or written to, the message KAVT2034-W is output to the integrated trace log, and job execution continues. In this case, the file for inheriting the monitoring status is not created.
- If you select the state Inherit for the Monitoring Files job status passing option, execute a Monitoring Files job in a start condition, and then kill the JP1/AJS3 service, the status of the Monitoring Files job will be passed. However, if the JP1/AJS3 service is subjected to planned termination rather than killed, the status of the Monitoring Files job will not be passed.
- The following directories might contain a file that could be read or written by the file monitoring process at any time while the process is running. Therefore, do not specify a file in the following directories as a monitoring target:  
When running on the physical host

For Windows, if the installation folder is the default installation folder or is in a folder protected by the system:

`%ALLUSERSPROFILE%\Hitachi\JP1\JP1_DEFAULT\JP1AJS2\jplajs2\sys`

The default location of `%ALLUSERSPROFILE%` is `system-drive\ProgramData`.

A *folder protected by the system* is the path to a folder in any of the following:

- `system-drive\Windows`
- `system-drive\Program Files`
- `system-drive\Program Files (x86)`

For Windows, if the installation folder is other than the above:

`JP1/AJS3-installation-folder\jplajs2\sys`

For UNIX:

`/var/opt/jplajs2/sys`

When running on a logical host

For Windows:

`shared-folder\jplajs2\sys`

For UNIX:

`shared-directory/jplajs2/sys`

- A Monitoring Files job executed under Windows might detect updates to files other than the expected target files when both of the following conditions are met:
  1. The monitoring target files are specified using a 3-character extension, with a wildcard for the rest of the file name.  
Example:  
Monitoring target files specified as `C:\Temp\*.txt`.
  2. The monitoring target folder contains files with extensions that are four characters or longer, and the first three characters match the extension as specified in 1 above.  
Example:  
Suppose that the monitoring target files are specified as `*.txt` (to match the beginning of a string by using a wildcard), and the target folder contains the three files `aaa.txt`, `aaa.txta`, and `aaa.txtab`. When any one of these files is updated, the Monitoring Files job will assume that the monitoring conditions are satisfied. The table below shows examples of how different monitoring target files are selected according to how you specify the file extension.

Table 7–5: Detection of monitoring target files by file extension

| Monitoring target file name | Updated files |         |          |           |
|-----------------------------|---------------|---------|----------|-----------|
|                             | aaa.tx        | aaa.txt | aaa.txta | aaa.txtab |
| C:\Temp\*.tx                | Y             | N       | N        | N         |
| C:\Temp\*.txt               | N             | Y       | Y        | Y         |
| C:\Temp\*.txta              | N             | N       | Y        | N         |
| C:\Temp\*.txtab             | N             | N       | N        | Y         |

Legend:

Y: An event is issued when this file is updated.

N: No event is issued when this file is updated.

- When you use a file transfer tool to perform an operation on a monitoring target file such as updating the file, the Monitoring Files job might assume that the **Delete** or **Create** monitoring condition is satisfied, even if the file was only overwritten. That is, if the tool performs file deletion processing internally, and the deletion timing happens to coincide with the monitoring interval, the Monitoring Files job assumes that the file has been deleted. To avoid such problems, we recommend that you use the **Create** and **Final time write** monitoring conditions even if the monitoring target files are always overwritten in your particular setup.
- If you change the Monitoring Files job status passing option from the state Do not inherit to Inherit during monitoring by a Monitoring Files job specified as a start condition, and then restart JP1/AJS3, the messages KAVT2031-E and KAVT2034-W are output to the integrated trace log. This indicates that the status of the job before the restart could not be inherited because the Monitoring Files job status passing option for the job was set to the state Do not inherit before you restarted JP1/AJS3. For this reason, do not change the option from the state Do not inherit to Inherit while a Monitoring Files job is active, as you will be unable to inherit the status from before JP1/AJS3 restarted.
- When you use a Monitoring Files job in Windows and specify \*. \* as the name of the monitoring target file, the job will also detect files without an extension.  
For example, if you specify C:\temp\\*. \* as the monitoring target file, an event will be issued when any of the following files under C:\temp is updated:
  - abc
  - abc.txt
  - abc.txt.txt
- You cannot specify a symbolic link as a monitoring target file. However, provided that the file itself is not a symbolic link, you can include a symbolically linked directory in the name of the target file.
- Each time the monitoring interval arrives, a Monitoring Files job checks whether the status of the monitoring target file has changed. When a change is detected, a close check is performed regardless of the monitoring condition (**Create**, **Change size**, or **Final time write**) specified in the Monitoring Files job. In Windows, the check process attempts to open the monitoring target file in write mode. If the attempt to open the file is successful, the file is immediately closed and a determination is made that no other process has the file open. No other program can access the monitoring target file while it is open for the purposes of a close check. For this reason, when you create a user program that operates a monitored file from within an event job or a jobnet with a start condition, take measures to deal with potential file access problems by incorporating retry processing into the user program.
- In Windows, if the monitored files are read-only, the files cannot be opened in write mode, a close check cannot be performed, and no event will be issued even when a file change that matches the monitoring conditions is detected. If a monitored file is read-only, the message KAVT2036-W is output to the integrated trace log. Remove the read-only attribute from the monitored files so that the files can be opened in write mode.
- The Windows file system uses the following two file name types:

- Long file names specified arbitrarily by users
- Corresponding short file names generated automatically by Windows (8.3 filenames)

The Monitoring Files job in JP1/AJS3 monitors both file name types, and therefore the Monitoring Files job will detect an event for a file whose short file name matches the monitoring target file. In this case, an event might appear to have been detected for the wrong file.

If the Monitoring Files job appears to have detected an event for the wrong file, run the following command at the command prompt to output the short file names of files in the folder that contains the monitoring target file. Check the command output to see whether the file name specified as the monitoring target file matches the short file name of another file in the folder.

```
dir full-path-of-folder-containing-monitoring-target-file /x
```

To prevent the Monitoring Files job from identifying a short file name as its monitoring target, specify the monitoring target file name as follows:

- When using wildcard characters to specify the monitoring target file, make sure that the base file name is at least 8 characters.

The following table shows examples of defining the monitoring target file name to include and exclude short file names as monitoring targets.

**Table 7–6: Examples of defining the name of the monitoring target file to include and exclude short file names as monitoring targets**

| Monitoring target file group |                      | Wildcard characters specified as monitoring target file name  |
|------------------------------|----------------------|---|
| Long file name               | Short file name      |   |
| C:\temp\ABCDEFGH001.log      | C:\temp\ABCDEF~1.log | To exclude short file names as monitoring targets:<br>C:\temp\ABCDEFGH*.log<br>The base file name ABCDEFGH is 8 or more characters. |
| C:\temp\ABCDEFGH002.log      | C:\temp\ABCDEF~2.log |   |
| C:\temp\ABCDEFGH003.log      | C:\temp\ABCDEF~3.log |   |
| C:\temp\ABCDXXXXXXX.log      | C:\temp\ABCDEF~4.log |   |
|                              |                      | To include short file names as monitoring targets:<br>C:\temp\ABCDEF*.log<br>The base file name ABCDEF is fewer than 8 characters.  |

- When using a full name to specify the monitoring target file, do not specify a file name in short file name format. Here, a file name in short file name format refers to a long file name in the 8.3 format used by short file names.

**Supplementary note**

If you specify the name of a monitoring target file in short file-name format, make sure that the files in the same location as the monitoring target file are all named in short file-name format.

- In a Linux environment, if a network connection between a host that is running a file monitoring job and a file system (such as NFS) is closed, the job might sometimes remain running. If this occurs, you might become unable to monitor the states of files correctly.

One solution to this problem is to set a timeout for the close check of the file monitoring job. If a timeout is set, a message can be output when the file monitoring job remains running after a disconnection. To output such a message, set the `CloseCheckTimeout` and `CloseCheckWarnLogInterval` environment setting parameters. For details about the `CloseCheckTimeout` environment setting parameter, see 20.6.2(34) *CloseCheckTimeout* in the *JP1/Automatic Job Management System 3 Configuration Guide*, for details about the `CloseCheckWarnLogInterval` environment setting parameter, see 20.6.2(35) *CloseCheckWarnLogInterval* in the *JP1/Automatic Job Management System 3 Configuration Guide*.

## 7.6.3 Notes on the Receive Mail job

The following provides precautions (items you should know in advance) for using the Receive Mail job.

Examples of jobnets that use the Receive Mail job are as follows:

- Execute the succeeding job only when an email is received from the system administrator.
- Execute the succeeding job only when the received email is titled Error.

In email reception monitoring, a condition is satisfied when an email is received. For details about email reception, see [2.6 Defining email reception monitoring job](#) in the *JP1/Automatic Job Management System 3 Linkage Guide*.

Cautionary note (UNIX only)

To change from an environment linked with a mail system to one that is not, change the parameter `ExecMode` under the `[JP1_DEFAULT#\JP1AOMAGENT\mail_link]` key from U to N by using the `jajs_config` command, and then restart JP1/AJS3.

#

If JP1/AJS3 is running on a logical host, this will be the logical host name.

## 7.6.4 Notes on the Monitoring Log Files job

The following provides precautions (items you should know in advance) for using the Monitoring log files job.

For an overview of the operation of the Monitoring Log Files job, see [2.4.4\(3\) Executing a process at log file update \(Monitoring log files job\)](#).

Cautionary notes

- The Monitoring Log Files job requires the log file trapping function of JP1/Base.
- The Monitoring Log Files job is executed using the log file trap functionality of JP1/Base. Before you execute the Monitoring Log Files job, start the log file trap management service of JP1/Base and the event service of JP1/Base.

If the log file trap functionality and event service of JP1/Base are not running, the Monitoring Log Files job enters *Now running* status but does not actually monitor its target. The Monitoring Log Files job starts monitoring after the event service and the log file trap management service of JP1/Base have started.

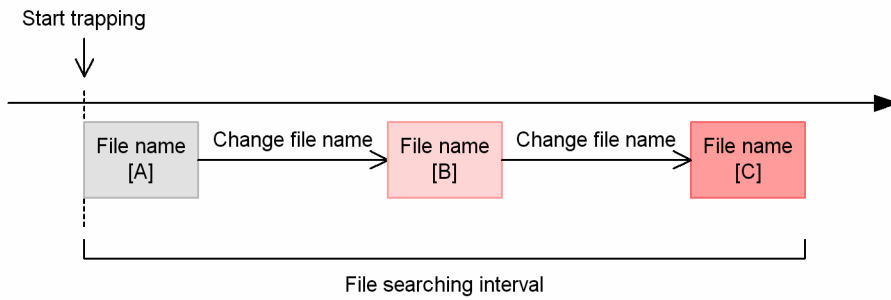
For details about the log file trap functionality, see the *JP1/Base User's Guide*.

- Do not stop the log file trap management service or event service of JP1/Base while the Monitoring Log Files job is running. Also, do not execute the JP1/Base `jevlogstop` command to terminate the log file trap run by the Monitoring Log Files job. If you do, events might not be detected even if the log data of monitoring targets is written.
- Each Monitoring Log Files job requires a share of memory, disk space, and system resources to make use of the log file trap functionality of JP1/Base.  
For details about estimating the resource requirements for JP1/Base, see the *JP1/Base User's Guide*.
- Do not monitor log files that might be mounted or unmounted while a job is executed. If you monitor such files, monitoring might not operate normally, or the system might incorrectly assume that a new log file was created and read the file from the beginning.
- When **SEQ2** is selected as the output format for log files, if a file is renamed more than once during a file search interval, the acquired messages will be lost. For this reason, take care when specifying the file search interval.

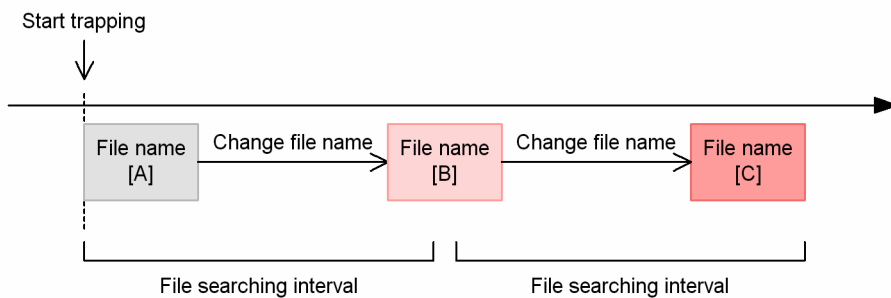
If you want to edit a file renamed during monitoring with **SEQ2** set, edit it after a new file is created and the specified file search interval elapses.

Figure 7–18: Examples when acquired messages are lost and not lost with SEQ2 selected

- When all data written into File B is lost:



- When data written into File B is not lost:



- The Monitoring Log Files job cannot monitor the output formats of the following log files:
  - **HTRACE**
  - **SEQ3**
  - **UPD**
- Make sure that the log files to be monitored by the Monitoring Log Files job are encoded in the character encoding that was specified for the system at the time when JP1/AJS3 started.
- If you use version 06-00 to 06-71 of JP1/AJS2 - View to view the detailed definition of a job whose file output format is **SEQ2**, the file output format appears blank. At this time, if you click **OK** in the Detailed Definition dialog box, all the items defined as **SEQ2** are re-defined as **SEQ**.
- If you specify multiple Monitoring Log Files jobs that reference the same log file (including the syslog), the load on the system is increased by the increased number of accesses to the log file. In this case, we recommend that you use the log file trap management service of JP1/Base to define the operation of a single log file trap. Then use the JP1/Base JP1 event conversion facility to define a log file update as a JP1 event, and use the Receive JP1 Event job to monitor the created JP1 event.
- As monitoring targets, do not specify log files that are accessed on network drives (for Windows) or the log files that are accessed by mounting a file system over a network such as NFS (for UNIX). JP1/AJS3 cannot monitor these files in the event of a network disconnection, and cannot determine whether such files are in use on another system.
- Do not attempt to monitor the following files (if you do, monitoring might not work properly):
  - Special files and device files
  - Log files containing records with binary data except at the end character of one line
  - Files whose exact names are not known beforehand
- If you are monitoring the scheduler log and execute `ajsalter -c COPY` during monitoring, monitoring will no longer work properly.

- If the target log file is not found when log file monitoring starts, the job continues to search for the target log file until it is found. Therefore, log data that is written to the log file after monitoring starts but before the file is found could fall outside the scope of the Monitoring Log Files job.
- When you execute a Monitoring Log Files job, you can specify an option that abnormally ends the job if the target file is not found. In this case, if the file is not found when monitoring starts, the Monitoring Log Files job ends without searching for the target file.
- If you want to use extended regular expressions for items specified by regular expressions, specify the JP1/Base settings.
- An item specified using a regular expression matches the condition if part of the specified character string matches. To require a full match, use a regular expression that explicitly specifies the full name. For information about the use of regular expressions in Windows, see the *JP1/Base User's Guide*. For information about regular expressions in UNIX, see your UNIX documentation.

In Linux, which is a prerequisite for JP1/Base, Japanese characters cannot be processed in regular expressions used by the log file trapping function. Therefore, the JP1/AJS3 Monitoring Log Files job also handles two periods (..) as one Japanese character. For details about regular expressions, see the *JP1/Base User's Guide*.

- You might want to execute the Monitoring Log Files job as a start condition for a job that uses an OR condition to trap multiple log data items. In this case, if log data written to the log file contains the specified log data items, the condition is satisfied multiple times for the same log entry. You can avoid this problem by using the AND, OR, and NOT operators as shown in the following example, so that the second and subsequent items are combined with all the prior items.

To monitor Error, Warning, Information, and Notice, specify:

```
lftpd="Error";
lftpd="Warning":!"Error";
lftpd="Information":!"Error":!"Warning";
lftpd="Notice":!"Error":!"Warning":!"Information";
```

- When you use a relative path to specify a log file, the current directory is assumed as follows.

In Windows:

```
system-folder\system32
```

In UNIX:

Directory where the `jajs_spmd` command is executed

- In some cases, when several log file monitoring jobs are executed concurrently in Windows, a JP1/Base error message might be output and the jobs may end abnormally. If this occurs, refer to the *JP1/Base User's Guide* and take the appropriate corrective action for the JP1/Base error message.
- If you run a Monitoring Log Files job on a host that runs an event server whose name is in FQDN format, enable the Event Server Name option (environment setting parameter `EventServerName`) in the configuration of JP1/AJS3 on the execution agent, and specify the event server name (FQDN format) of the event server used by the Monitoring Log Files job. For details about this procedure, see *6.3.17 Setting the event server name in the system using DNS* in the *JP1/Automatic Job Management System 3 Configuration Guide* (for Windows) or *15.3.16 Setting the event server name in the system using DNS* in the *JP1/Automatic Job Management System 3 Configuration Guide* (for UNIX).

## 7.6.5 Notes on the Monitoring Event Log job

The following provides precautions (items you should know in advance) for using the Monitoring Event Log job.

For an overview of the operation of the Monitoring Event Log job, see [2.4.4\(4\) Executing a process on receipt of a Windows event log record \(Monitoring event log job\)](#).

- Before you can use the Monitoring Event Log job, the event log trapping function must be configured in the action definition file of JP1/Base.
- Start the event log trapping service and event service of JP1/Base before running a Monitoring Event Log job. If the event log trapping service and event service of JP1/Base are not running, the Monitoring Event Log job will remain in *Now running* status and take no action if a Windows event log is received.
- The Windows events monitored by the Monitoring Event Log job conform to the specifications of the JP1/Base event log trapping function. For details about the operating definition of the JP1/Base event log trapping function, see the *JP1/Base User's Guide*.
- When a Windows event cannot be placed in a category, "Others" appears in the Windows event viewer. However, if the event is converted to a JP1 event by using the JP1/Base event log trapping function, the category of the event will be "None". For this reason, when you define a Monitoring Event Log job, specify "None" rather than "Others" for **Category**. If you specify the string "Others", the monitoring condition will not be satisfied.
- In its default state, the JP1/Base event log trapping function monitors **Error** and **Warning** events only. If you want to monitor **Verbose**, **Information**, **Critical**, **Failure audit**, and **Success audit** events, add the necessary definition of the target events in the action definition file of the event log trapping function of JP1/Base.
- The monitoring condition is satisfied when the monitoring condition specified in **Category** of the Monitoring Event Log job completely matches the category of the Windows event. The maximum size of a monitoring condition is 255 bytes. However, the Windows event log might contain entries longer than 255 bytes. In this case, the system compares the character string specified in **Category** of the Monitoring Event Log job against the first 255 bytes of the category information of the Windows event, and the condition is satisfied if they match.
- An item specified using a regular expression matches the condition if part of the specified character string matches. To require a full match, use a regular expression that explicitly specifies the full name. For information about the use of regular expressions in Windows, see the *JP1/Base User's Guide*.
- Do not specify a character string that includes a line break for the **Explanation** definition item of the Monitoring Event Log job. If you do so, the job might not be able to detect events. If the explanations of Windows events that you want to monitor include a line break, use `.*\n` as a regular expression that represents a line break. For information about the use of regular expressions in Windows, see the *JP1/Base User's Guide*.

Example when the following character strings are monitored:

```
Starting TEST1...
Starting TEST2...
```

Specify the following:

```
Starting TEST1.*\nStarting TEST2
```

- The following errors might occur depending on the OS on the job execution host or the JP1/AJS3 version.

When the OS is UNIX on the job execution host

The job enters the *Failed to start* status and the KAVT0567-E message appears.

When the OS is Windows Server 2003 on the job execution host

If a monitoring condition that is not supported by Windows Server 2003 is specified, an error occurs.

The monitoring conditions that are not supported by Windows Server 2003 are as follows:

Log type: **Any log type**

Event type: **Verbose** and **Critical**

- When the JP1/AJS3 version is 10-00 or later on the job execution host



When only **Verbose** or **Critical**, or only both, are specified, the job enters the *Failed to start* status and the KAVT0591-E message appears.

When **Verbose** or **Critical** is specified and other event types are also specified, the job is executed and no error occurs.

- When the JP1/AJS3 version is earlier than 10-00 on the job execution host

The job ends abnormally and the KAVT1013-E message appears.

## 7.6.6 Notes on the Interval Control job

The following provides precautions (items you should know in advance) for using the Interval Control job.

Example of jobnets that use the Interval Control job are as follows:

- Register a jobnet for execution, and execute the succeeding job 30 minutes later.

For details about how to register a jobnet for execution, see 4. *Executing an Application* in the manual *JP1/Automatic Job Management System 3 Overview*.

As the wait time for the Interval Control job, you can specify a value from 0 to 1,440 (in minutes).

Cautionary notes

- The wait time you specify in the Interval Control job is the wait time for the interval control process, not the time from the start to end of the job execution. Depending on such factors as the status of the network at the time, the wait time might not exactly match the time you specify.
- When the Interval Control job is running and the JP1/AJS3 service is restarted on the host corresponding to the agent running the Interval Control job, measurement (monitoring) of the preset wait time begins after the restart. When the wait time expires, the monitoring condition is satisfied.

Suppose 60 (minutes) is defined as the wait time for the Interval Control job. If the JP1/AJS3 service restarts while the Interval Control job is in the *Now running* status, the monitoring condition is satisfied 60 minutes after the restart regardless of the time that has elapsed up to the restart.

For the following cases, the job status is the *Now running* even after the JP1/AJS3 service restarts. Note that if you restart the JP1/AJS3 service during monitoring, the amount of time between execution of the Interval Control job and when the monitoring condition is satisfied is longer than the wait time specified for the job.

- When the Interval Control job defined in a start condition is executed.

- When the Interval Control job is executed from the manager host on which the option to continue execution of active event jobs is enabled.

For details about the option to continue execution of active event jobs, see 8.2.1 *Continuing the execution of event jobs and custom event jobs if the JP1/AJS3 service stops* in the *JP1/Automatic Job Management System 3 Administration Guide*.

- If you click **Yes** in the **Expire right after starting** section in the Detailed Definition - [Interval Control] dialog box, define the Interval Control job in a start condition, and then register the root jobnet with that start condition for execution while either of the following is true, it might take some time before the start condition is satisfied.
  - Many root jobnets with start conditions or jobnets defined with event jobs are registered for execution at the same time.
  - Many events are detected.

## 7.6.7 Notes on defining passing information

Note the following when defining passing information:

- Passing information can be passed even if the event job and the succeeding job are executed on different agent hosts. If the hosts use different character codes, the passing information is converted to the character codes used on the succeeding host. Note, however, that if the passing information contains unsupported characters, JP1/AJS3 might not operate correctly.
- For details about the characters you can use with JP1/AJS3, see *2.4.2 Language type and character encoding of the system* in the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*. Characters that are not supported by JP1/AJS3 cannot be used even in a stand-alone configuration. Make sure that passing information does not include such characters.
- If multiple event jobs are related to a single succeeding job, the succeeding job can inherit the information received by all the event jobs. However, if multiple event jobs define the same macro variable name, the older received information will be overwritten.
- If the same macro variable name is defined more than once in the passing information of a single event job, the information that is defined first is passed.

Example for the Receive JP1 Event job:

Suppose that the following two macro variables are defined in passing information:

- `?AJS2111?:EVID` (specifies that macro variable `?AJS2111?` inherit an event ID)
- `?AJS2111?:EVMSG` (specifies that macro variable `?AJS2111?` inherit message information)

In this case, `?AJS2111?` inherits an event ID.

- Do not pass data that contains an escape sequence to the command line. If you pass data that contains a space character, unexpected behavior might result. To prevent this, when you define a macro variable, enclose it in double quotation marks ("").
- When you specify a macro variable in the command line of the succeeding job, the succeeding job cannot correctly inherit passing information if the information contains space characters or single quotation marks (''). Note that macro variables are executed on the agent host that executes the succeeding job of the event job by replacing the macro variable in the command line with its value. When defining passing information, define only information that can be treated as command arguments at execution.
- If there is no information to be inherited from the event job or if the event job is not executed, the macro variables defined in the succeeding job will not inherit any information. In this case, if you define `?AJS2111?` as the macro variable name when you execute the job, the character string `?AJS2111?` will be passed.
- When you pass the information received in an event job to the parameters of a standard job or an action job, if the data to be passed contains a double quotation mark ("), you must add a backslash (\) before the double quotation mark. If you do not add a \ before the double quotation mark, the double quotation mark will be ignored when it is passed to a standard job or an action job.  
To prevent this, enable the option for handling data containing double quotation marks (as is) in passing information. For details about how to set this option, see *6.3.4 Passing event data containing double quotation marks* in the *JP1/Automatic Job Management System 3 Configuration Guide* (for Windows systems) or *15.3.4 Passing event data containing double quotation marks* in the *JP1/Automatic Job Management System 3 Configuration Guide* (for UNIX systems).
- When passing information is used in a command line of a job, even if the passing information contains a double quotation mark ("), the information is passed to the succeeding job without being converted. Therefore, the job might not be executed correctly depending on the command line restrictions of the OS in question. If you want to use passing information that contains a special character, do not use it directly in a command line. Such information must be passed to an environment variable.

- If you define macro variables in an event job defined in a start condition, the information is passed to the entire jobnet that starts when the start condition is satisfied.
- Make sure that the macro variable names and passing information together do not exceed 4,096 bytes. In particular, if you use the AND condition to define a start condition, the system merges the macro variable names and passing information for all of the event jobs in the start condition. For this reason, take care that the macro variable names and passing information do not exceed 4,096 bytes in total.
- Note the following when using a job that specifies a macro variable in a definition item that permits the specification of macro variables from JP1/AJS3 version 09-50:
  - If you copy the job to an environment running a version of JP1/AJS3 - Manager earlier than 09-50, the macro variable will not be expanded.
  - If you copy a passing information setting job to an environment running a version of JP1/AJS3 - Manager earlier than 09-50, executing the passing information setting job will cause an error.

## 7.6.8 Notes on restarting the JP1/AJS3 service while event jobs are running

If you perform any of the operations listed below while event jobs (including those with a start condition) are running, communication and correlation processing take place between the schedule control and event/action control functions to ensure consistency in job statuses between the functions. If a large number of event jobs are in *Now running* status, the amount of data to be processed places a considerable load on the system.

Operations that result in high system load

- Stopping the scheduler service, and then performing a warm or hot start.
- Stopping the JP1/AJS3 service on the manager host, and then performing a warm or hot start.
- Stopping and then restarting the JP1/AJS3 service on the agent host.
- Executing the `jajs_maintain` command.

Use the following workarounds to avoid placing a high load on the system:

Workarounds

- Before you perform any of the above operations, forcibly terminate any jobnets that use event jobs. Register them for execution again when the operation is complete.
- Before you perform any of the above operations, forcibly terminate all event jobs. Re-execute them when the operation is complete.
- Do not run any jobs until 30 minutes to an hour has elapsed after performing the operation.
- Do not trigger any monitored events until 30 minutes to an hour has elapsed after performing the operation.

The following problems can occur when a high load is placed on the system.

Problems resulting from a high system load

1. Event jobs (including those with a start condition) registered for execution immediately after you performed one of the above operations take a long time to enter *Now running* status.<sup>#1</sup>
2. When you forcibly terminate an event job in *Now running* status, or a jobnet with a start condition in *Now monitoring* status, it takes a long time to terminate.<sup>#1</sup>
3. When you change the status of an event job in *Now running* status, the change takes a long time to take effect.<sup>#1</sup>

4. It takes a long time for an event to be detected when a monitoring condition is satisfied.<sup>#1</sup>
5. Event jobs (including those with a start condition) registered for execution immediately after you performed one of the above operations remain in *Queuing* status.<sup>#2</sup>
6. You attempt to forcibly terminate an event job in *Now running* status, or a jobnet with a start condition in *Now monitoring* status, but it does not terminate.<sup>#2</sup>
7. A monitoring condition is satisfied but no event is detected.<sup>#2</sup>

#1

Problems 1 through 4 can occur when you perform an operation that results in a high system load while the event/action control manager is near its resource limits. For details about the resource limits that apply to event/action control, see *B.8 Limits for the event/action control* in the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*.

#2

Problems 5 through 7 can occur when you perform an operation that results in a high system load after the event/action control manager has exceeded its resource limits. For details about the resource limits that apply to event/action control, see *B.8 Limits for the event/action control* in the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*.

If you perform an operation that results in a high system load when a large number of event jobs are running, a large volume of communication takes place between the scheduler control and event/action control functions. This results in an increased number of unreported items of information that are generated and managed in the event of a communication error. JP1/AJS3 imposes a limit on the number of unreported item that can be kept, to prevent the high system load that results from processing this data from monopolizing system resources and delaying job execution and event detection. When the number of unreported items reaches the limit, the information is deleted starting with the oldest item. Any of problems 5 through 7 can occur as a result, depending on the content of the deleted data.

The upper limit of the number of unreported items of information that can be kept is not disclosed to JP1/AJS3 users. Instead, the limit for event/action control is calculated from the number of unreported items generated by an operation that results in a high system load. When using JP1/AJS3, be careful not to exceed this limit.

If any of the problems mentioned above occurs, take the following remedial action:

#### Recovery procedure

For problems 1 to 4

Wait until the processing finishes. This can take 30 minutes to an hour, depending on the number of jobs affected.

For problem 5

Forcibly terminate the event job or jobnet in question, and then re-register it for execution.

For problem 6

For an event job, change the status of the job and terminate it. For a jobnet with a start condition, forcibly terminate the jobnet again.

For problem 7

Forcibly terminate the event job or jobnet in question, and then re-register it for execution. Then, generate the event again.

## 7.6.9 Monitoring events or messages issued by JP1/AJS3

When monitoring JP1/AJS3 messages, monitor the message ID and the first part of the message, rather than the entire message text. If you monitor the entire message text, JP1/AJS3 could fail to detect the target message if it has extra information added at the end.

You might want to set a JP1/AJS3 event job to monitor JP1 events or Windows events issued by JP1/AJS3, or to monitor the messages output to log files. However, such an event job might be unable to detect its monitoring targets as events depending on the status of JP1/AJS3, or might be executed repeatedly depending on the monitoring condition settings. For example, the following can occur:

- Like a normal job, an event job is executed as a job in a jobnet. Monitoring of events does not commence until both JP1/AJS3 and the jobnet have started. Because the jobnet-initiating event is issued after the first job starts, an event job cannot monitor a jobnet-initiating event of JP1/AJS3 which is executing the event job.
- If a problem occurs in JP1/AJS3 and the jobnet in which that event job is defined ends abnormally, the event job itself also ends abnormally. If this situation occurs, events cannot be monitored.
- An event job also issues JP1 events or Windows events, or outputs messages to log files, as part of its job processing. Because the event job monitors the JP1 events, Windows events, and messages that the event job itself outputs to the log files, monitoring continues in an infinite loop.

When you want to execute actions by using events from JP1/AJS3 or messages in log files as triggers, you can avoid these problems by using the automatic action functionality of JP1/IM. Alternatively, you could use a JP1/AJS3 manager that is different from the one executing the event job. The preventive measures when you want to execute commands by using events from JP1/AJS3 or messages in log files as triggers are described below for two cases: when there are multiple JP1/AJS3 managers (including a configuration where multiple instances of JP1/AJS3 are concurrently running on multiple logical hosts) and when there is only one JP1/AJS3 manager (only one instance of JP1/AJS3). The following also gives notes on monitoring events and messages.

Note that the description in this section applies to monitoring targets such as JP1 events issued by JP1/AJS3 itself, Windows events, and messages output to log files or the syslog. The description does not apply when the Send JP1 Event job is used or when JP1 events are sent as a user job.

### **(1) When there are multiple JP1/AJS3 managers (including a configuration where multiple instances of JP1/AJS3 are concurrently running on multiple logical hosts)**

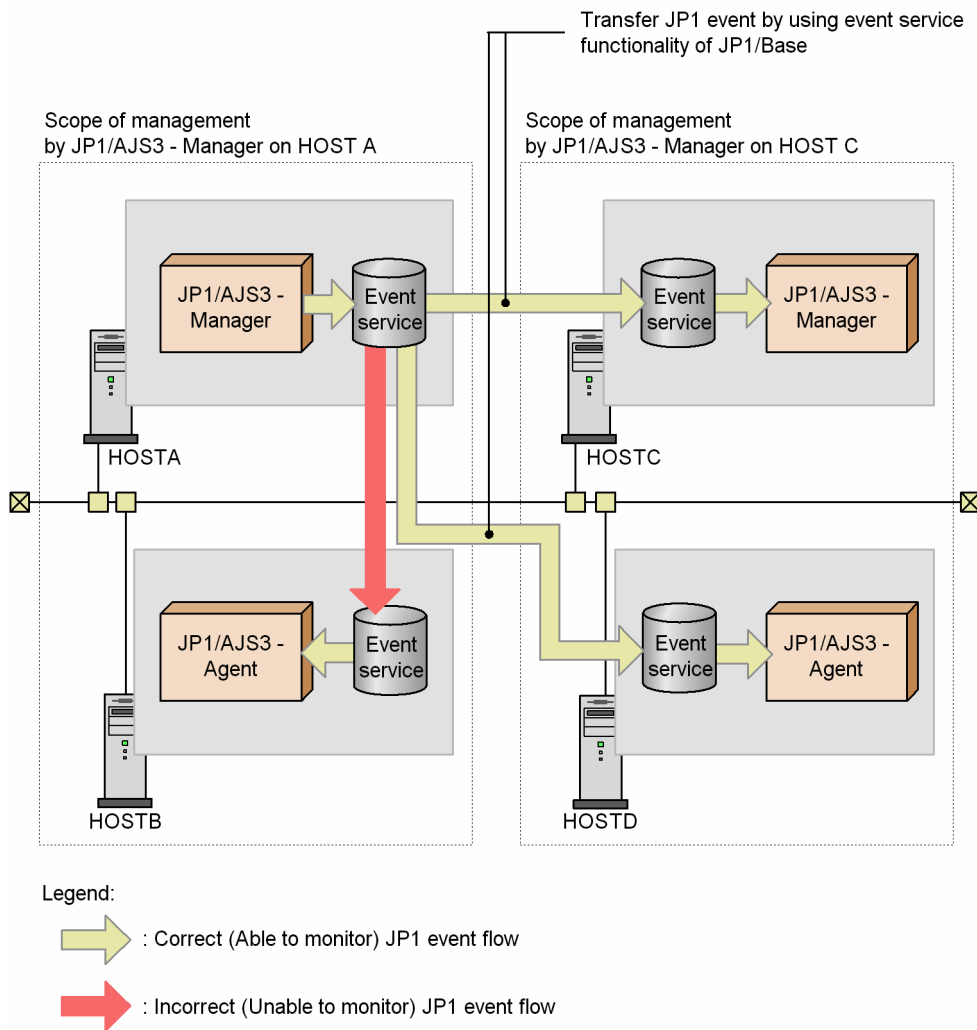
The following describes the measures to take to monitor events issued by JP1/AJS3 on another logical host or messages in log files when there are multiple JP1/AJS3 managers (or when multiple instances of JP1/AJS3 are concurrently running on multiple logical hosts). In this case, JP1/IM is not needed.

#### **(a) Monitoring JP1 events issued by JP1/AJS3 on another logical host, or when there are multiple JP1/AJS3 managers**

Under normal circumstances, you cannot monitor JP1 events issued by an instance of JP1/AJS3 running on another host (logical host). However, you can monitor such events if you transfer them using the event service functionality of JP1/Base. To do this, transfer JP1 events to an event service on another host (logical host) that is not managed by the JP1/AJS3 that issued the JP1 events. You cannot monitor the events if you transfer them to the event service running on the host managed by the JP1/AJS3 that registered the JP1 events.

The following figure shows an example flow of JP1 events.

Figure 7–19: Flow of JP1 events



In the above example, if you want to monitor the JP1 events issued by JP1/AJS3 on HOSTA, transfer them to the event service on HOSTC or HOSTD. You cannot monitor the events if you transfer them to the event service on HOSTB.

### (b) Monitoring Windows events issued by JP1/AJS3 on another logical host

Windows events can only be monitored on logical hosts that use the JP1/Base event log trapping function.

When you want to monitor Windows events issued by JP1/AJS3, set up a separate logical host and monitor the events from a different manager. In this case, you cannot use the Monitoring Event Log job from the monitored JP1/AJS3.

### (c) Monitoring log file messages output by JP1/AJS3 on another logical host

Reference the monitored log files from JP1/AJS3 on another logical host.

## (2) When there is only one JP1/AJS3 manager (only one instance of JP1/AJS3)

The following describes the measures to take to monitor events issued by JP1/AJS3 or messages in log files when there is only one JP1/AJS3 manager. In this case, JP1/IM is needed.

### (a) Monitoring JP1 events issued by JP1/AJS3

Use the automatic action functionality of JP1/IM.

### (b) Monitoring Windows events issued by JP1/AJS3

Use the Windows event trapping functionality of JP1/Base and the automatic action functionality of JP1/IM.

### (c) Monitoring log file messages output by JP1/AJS3

Use the log file trapping functionality of JP1/Base and the automatic action functionality of JP1/IM.

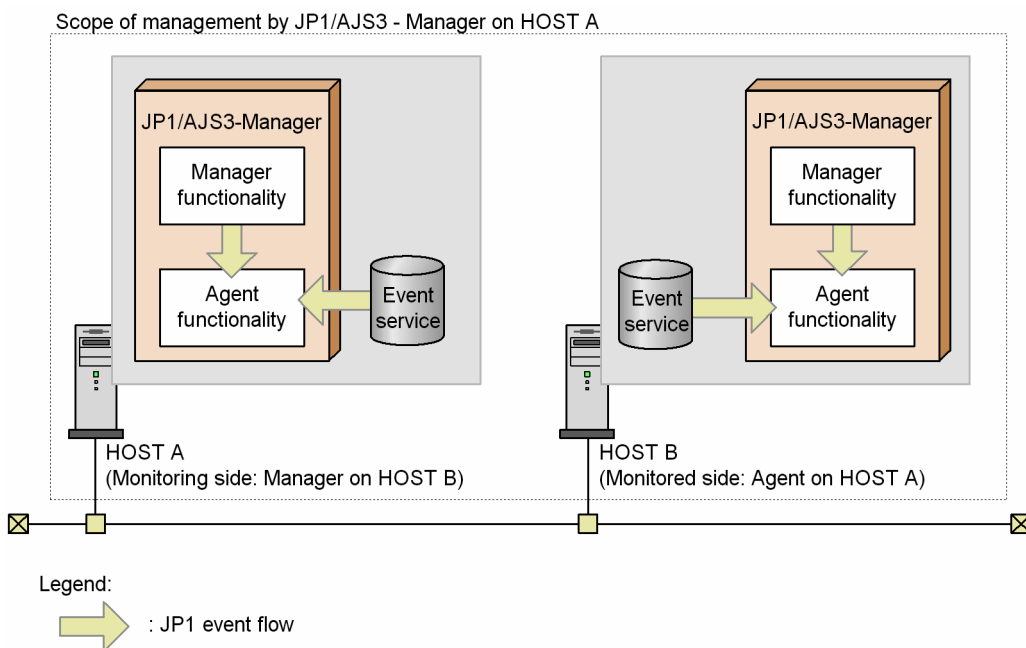
## (3) Notes on monitoring events and messages

The following provides precautions for monitoring events or messages.

### (a) Restrictions when using an event job to monitor a JP1/AJS3 - Manager host from another JP1/AJS3 - Manager host

The following describes the restrictions that apply when HOSTB is monitored from HOSTA by using an event job as shown in the figure below. In this case, JP1/AJS - Manager host (HOSTA) is set as the manager of HOSTB (monitoring host), and another JP1/AJS3 -Manager host (HOSTB) is set as the agent of HOSTA (monitored host).

Figure 7–20: Monitoring a JP1/AJS3 - Manager host from another JP1/AJS3 - Manager host by using an event job (1)

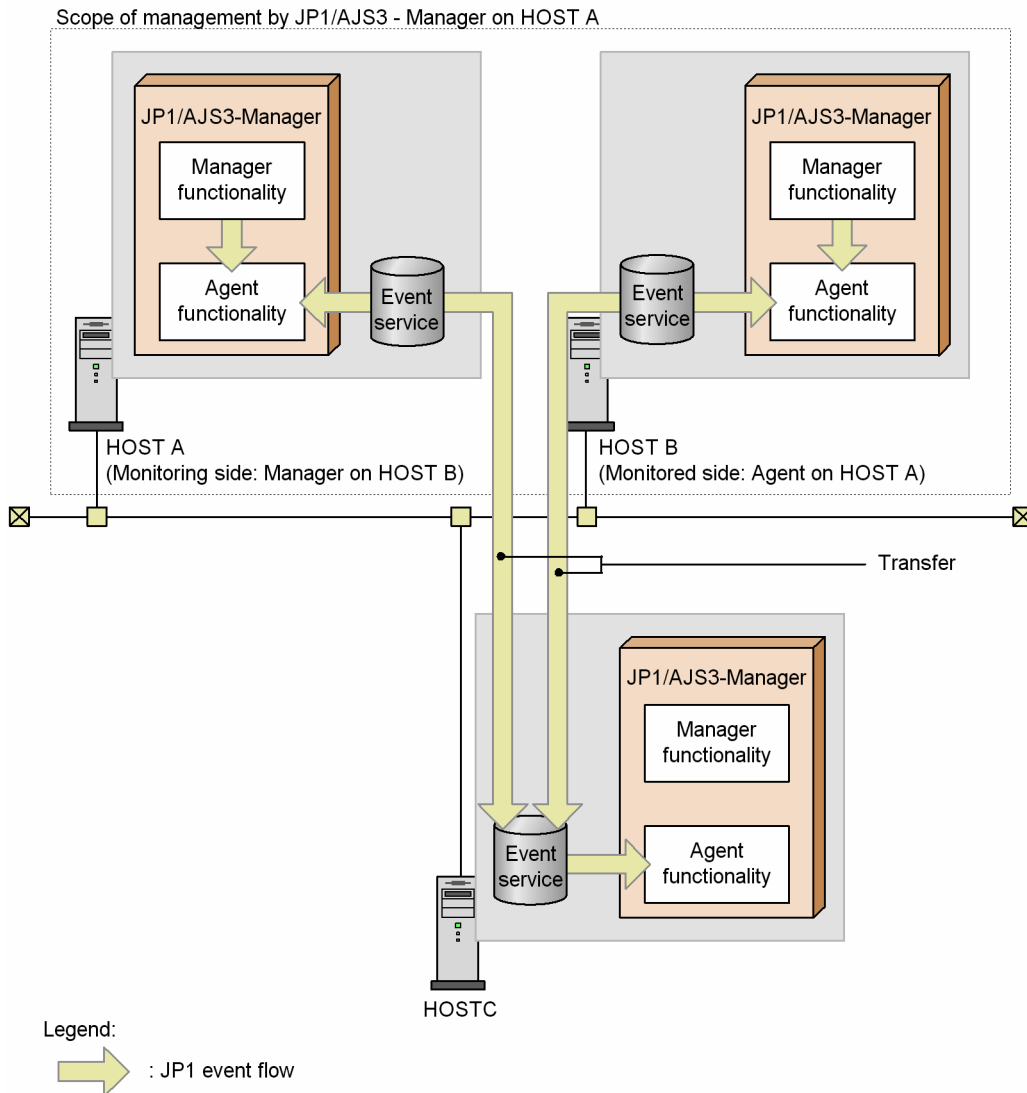


#### Restrictions

- When JP1/AJS3 is not running on HOSTB, you cannot use an event job to monitor HOSTB.  
The manager provides the agent functionality. Therefore, the agent of HOSTB monitored by HOSTA is started or stopped in tandem with the manager of HOSTB starting or stopping.
- If JP1/AJS3 on HOSTB terminates abnormally, it cannot provide information to HOSTA (you cannot monitor HOSTB from HOSTA).  
The manager provides the agent functionality. Therefore, the agent of HOSTB monitored by HOSTA stops in tandem with the manager of HOSTB ending abnormally.

To monitor a host in this configuration, transfer events to a JP1/AJS3 - Manager host that is neither the monitoring JP1/AJS3 - Manager host nor the monitored host.

Figure 7–21: Monitoring a JP1/AJS3 - Manager host from another JP1/AJS3 - Manager host by using an event job (2)



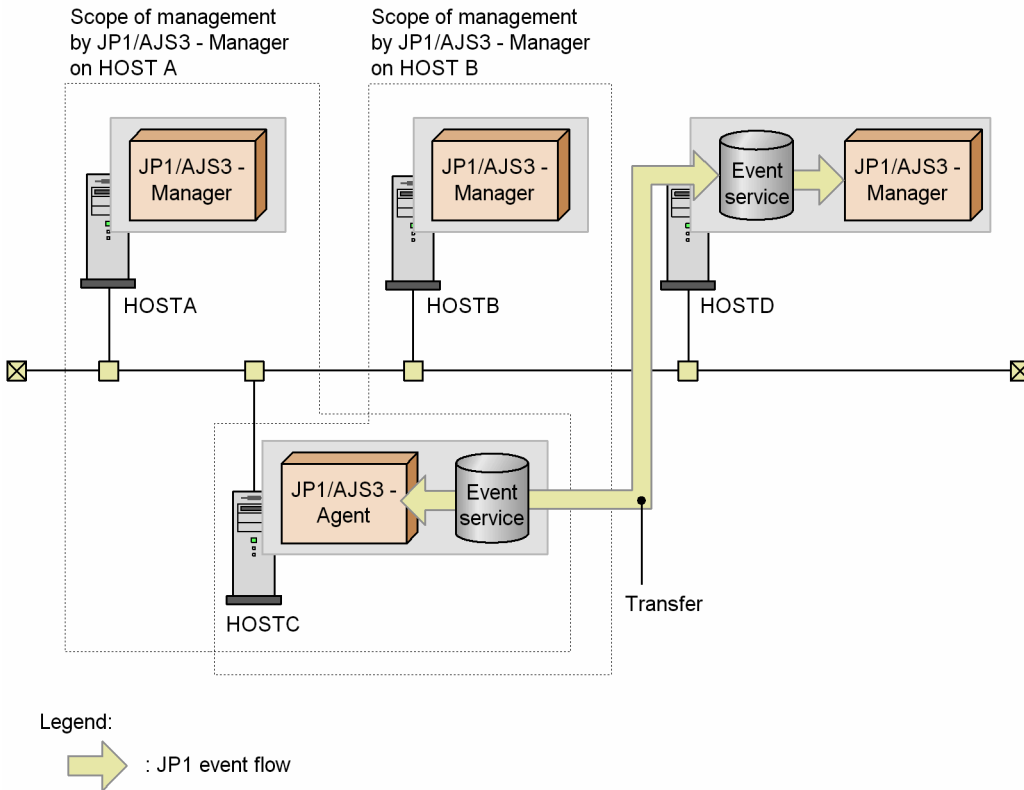
**(b) Monitoring events and messages when multiple JP1/AJS3 - Manager hosts use the same JP1/AJS3 - Agent host as an agent**

When the same JP1/AJS3 - Agent host is configured as an agent for multiple JP1/AJS3 - Manager hosts, JP1/AJS3 - Agent monitors the same log files (JP1/AJS3 log files to which the instance of JP1/AJS3 on the Agent host outputs information) for events and messages. Therefore, you cannot monitor the log files correctly even from a Manager on a different host.

If you want to monitor events and messages in this configuration, transfer the events and messages to a different JP1/AJS3 - Manager host from the one managing the JP1/AJS3 - Agent.



Figure 7–22: Monitoring events and messages when multiple JP1/AJS3 - Manager hosts use the same JP1/AJS3 - Agent host as an agent



## 7.7 Notes on using action jobs

---

The following provides precautions (items you should know in advance) for using action jobs.

### Cautionary notes

- You cannot specify a **User name** in an action job. For details, see the cautionary notes for each type of action job.
- Action jobs are affected by the concurrently executable job limit that JP1/AJS3 imposes. If you attempt to execute more jobs on the agent host than the limit allows, the jobs are placed in *Waiting to execute* status. For details about the maximum number of concurrently executable jobs, see 2.5.4 *Maximum number of concurrently executable jobs* in the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*.

### 7.7.1 Notes on the Send JP1 Event job

The following provides precautions (items you should know in advance) for using the Send JP1 Event job. When defining the JP1 event you want to send, you specify the event level, event ID, and event destination host. After sending a JP1 event, you can check whether the JP1 event has reached the event destination host. JP1/Base must be installed in the destination host.

#### Important

Before executing the Send JP1 Event job, you must start the JP1/Base event service on the event-originating host and event destination hosts.

Examples of jobnets that use the Send JP1 Event job are as follows:

- Define a jobnet so that a JP1 event is sent if a specific job ends abnormally. In another jobnet, define the start condition so that the jobnet starts if the jobnet receives the JP1 event that is sent from the other jobnet. With these settings, the second jobnet starts when a specific job ends abnormally.
- Define a jobnet to send a JP1 event to the server on which JP1/IM - Manager is installed when a job ends, and check the sent JP1 event by using JP1/IM - View.

### Cautionary notes

- The arrival confirmation facility for JP1 events merely allows you to wait until you can confirm that the event has arrived at the destination event server. The feature does not perform retries even if transmission fails. Therefore, when you execute a Send JP1 Event job while the event service is inactive at the destination, the job ends immediately regardless of whether **Check interval** or **Check count** is specified.
- If you do not check whether the JP1 event you sent has arrived at the destination, an error does not occur even if you specify the following event servers as event destination host names:
  - An event server that is not defined as the JP1/Base event server
  - An inactive event server
  - An event server that cannot receive JP1 events due to a network error or other problem
- The event server of the local host cannot acquire a sent JP1 event that specifies a destination host name other than the local host.
- When you send a JP1 event to the event server of another host with a destination host name specified, the JP1 event is not subject to retry processing according to the `forward-limit` setting in the event server settings file (`conf`) of JP1/Base.

- When you specify a macro variable as an event destination host name, message, or extended attribute, a double quotation mark (") in the passing information might result in the information being passed incorrectly or cause an error. Specify macro variables only if you are sure that they will not cause any problems with the passing information.
- If you do not specify a destination host name, JP1 events are sent to the event server of the host that executes the Send JP1 Event job.
- You cannot specify a **User name** in an action job. The sender of the JP1 event will be the primary OS user mapped to the **User who registered** or **User who owns** of the Send JP1 event job.
- When a logical host name that does not exist in the local host is specified in the JP1\_HOSTNAME environment variable, even if the execution host of the Send JP1 Event job is a logical host, the name of the host that sends the event will be the name of the physical host.
- A Send JP1 Event job uses the JP1/Base event server functionality. For this reason, if you want to set up a transmission path for a JP1 event that traverses multiple LAN environments, you must make the settings on the JP1/Base side (in the `conf` file). For details about how to do so, see the *JP1/Base User's Guide*.
- In the API settings file for the JP1/Base event service, set `keep-alive` for the communication type of the `server` parameter. Setting `close` as the communication type can result in operational problems, including JP1/AJS3 being unable to issue JP1 events at startup, or Send JP1 Event jobs ending abnormally with message KAVT1040-E output to the integrated trace log. For details about the API settings file and how to set parameters, see the *JP1/Base User's Guide*.

## 7.7.2 Notes on the Send Mail job

The following provides precautions (items you should know in advance) for using the Send Mail job. This job sends email messages when JP1/AJS3 is linked to a mail system. This action job can send email to both Windows and UNIX hosts.

Examples of jobnets that use the Send Mail job are as follows:

- The jobnet sends an email if a job ends with a warning.
- The jobnet sends an email when all the jobs in the jobnet end.

For details about the functionality that can be implemented by the Send Mail job, see *2.7 Defining email sending job* in the *JP1/Automatic Job Management System 3 Linkage Guide*.

### Cautionary note

You cannot specify a **User name** in an action job. When you execute a Send Mail job on a UNIX host, log in as a JP1 user mapped to the primary OS user who will be the mail sender and then register the job for execution.

## 7.7.3 Notes on the OpenView Status Report job

The following provides precautions (items you should know in advance) for using the OpenView Status Report job. This job reports designated statuses to HP NNM. For details about linkage with HP NNM, see *A. Monitoring Jobnets Using HP NNM* in the *JP1/Automatic Job Management System 3 Linkage Guide*.

### Cautionary note

You must have superuser privileges to execute an OpenView Status Report job on a UNIX host. Because you cannot specify a **User name** in an action job, log in as a JP1 user that mapped to the root user as the primary OS user, and then register the job for execution.

## 7.7.4 Notes on the Local Power Control job and Remote Power Control job

The following provides precautions (items you should know in advance) for using the Local Power Control job and Remote Power Control job.

- The Local Power Control job links with JP1/Power Monitor and shuts down the manager host or agent host.
- The Remote Power Control job links with JP1/Power Monitor and starts and shuts down the agent host of JP1/Power Monitor on the network. This job can control hosts in the manager/agent configuration of JP1/Power Monitor. It does not depend on the manager/agent configuration of JP1/AJS3.
- When the Remote Power Control job is executed, the job sends a request to JP1/Power Monitor on the agent (job-executing host) to perform remote power control. JP1/Power Monitor then starts remote power control. The agent's next power-on time is determined by the time zone setting of the agent.
- The host that executes the Remote Power Control job must be set as the manager host for the remote power linkage function of JP1/Power Monitor.

### Supplementary note

If you use the option to continue execution of active event jobs, the event job and the custom event job will remain in *Now running* status on the manager host if JP1/AJS3 terminates on the agent host. This rules out planned termination and other modes of operation that depend on waiting for jobs to terminate on the manager host. If you must use a Local Power Control job or Remote Power Control job in an environment where this option is enabled, you can work around this problem by waiting for active event jobs and custom event jobs to end or forcibly terminating them, or waiting for the scheduling function to end the event jobs and the custom event jobs before performing power control.

### (1) Example of a jobnet that uses the Local Power Control job

Example jobnets defined with the Local Power Control job are as follows:

- When the preceding job ends, the jobnet shuts down the manager host or agent host specified as the execution host.

Note that executing a Local Power Control job from JP1/AJS3 is the same as using the JP1/Power Monitor calendar to perform a planned stop.

### Supplementary note

For details about the OS users who can execute a Local Power Control job, see [\(3\) Notes on OS users capable of executing power control jobs](#). Because you cannot specify a **User name** in an action job, to execute a Local Power Control job, log in as a JP1 user that mapped to the root user as the primary OS user, and then register the job for execution.

### (2) Example of a jobnet that uses the Remote Power Control job

The following shows examples of jobnets that use the Remote Power Control job:

- The jobnet starts the agent host (agent host of JP1/Power Monitor) before executing a job on the host. When the job ends, the host is shut down.
- If the preceding job ends abnormally, the jobnet restarts the host (agent host of JP1/Power Monitor) that executed the job, and executes the job again.

### Cautionary note

For details about the OS users who can execute a Remote Power Control job, see [\(3\) Notes on OS users capable of executing power control jobs](#). Because you cannot specify a **User name** in an action job, to execute a Remote Power Control job, log in as a JP1 user that mapped to the root user as the primary OS user, and then register the job for execution.

### (3) Notes on OS users capable of executing power control jobs

A power control job (Local Power Control job or Remote Power Control job) that links with JP1/Power Monitor can only be executed by the OS users listed in the following table.

Table 7–7: OS users capable of executing power control jobs

| OS      | OS user           |   |
|---------|-------------------|---|
| Windows | With UAC enabled  | Administrator user (built-in Administrator account) <sup>#</sup>                    |
|         | With UAC disabled | Administrator user (built-in Administrator account) or user in Administrators group |
| UNIX    | Superuser         |   |

#

You can have an OS user with administrator permissions execute power control jobs by enabling the setting that gives OS users with administrator permissions to execute power control jobs. For details, see *6.2.22 Settings for executing jobs as a user with administrator permissions when the UAC is enabled* in the *JP1/Automatic Job Management System 3 Configuration Guide*.

## 7.8 Notes on using flexible jobs

---

The following provides notes on using flexible jobs for each agent host OS:

Notes common to all OSs:

- *OSs that can be used for relay, broadcast, and destination agents*  
Only Windows or Linux hosts can be specified as relay, broadcast, and destination agents that are used to execute flexible jobs. If you define a flexible job in an OS other than Windows or Linux, you must specify a relay agent.
- *Forced termination of a flexible job*  
If you forcibly terminate a flexible job, the job itself is terminated but the programs running on the destination agent are not forcibly terminated.
- *Notes on physical and logical hosts*  
You can execute flexible jobs on physical hosts only. If you specify the name of a logical host as the name of the destination agent, the flexible job will be executed on the physical host.
- *Notes on user mapping*  
User mapping must be possible on both the relay and destination agents. If you choose to use a relay agent, set either the host name of the relay agent or \* (asterisk) for **Server host** in the user mapping definition on the destination agent.
- *Notes on the FXJOB\_MONITOR\_TIMEOUT environment setting parameter*
  - The status of flexible jobs changes to *Ended abnormally* when the time specified in the FXJOB\_MONITOR\_TIMEOUT environment setting parameter elapses. The default value of this parameter is 60 minutes. If you have flexible jobs that require 60 or more minutes for execution, explicitly set a proper value for the FXJOB\_MONITOR\_TIMEOUT environment setting parameter.
  - If an error occurs during execution of a flexible job, it might take the time specified for the FXJOB\_MONITOR\_TIMEOUT environment setting parameter (default: 60 minutes) before the job status changes to *Ended abnormally*.
- *Note applicable if a communication error occurs*  
If a communication error occurs, the status of a flexible job changes to *Ended abnormally*. In this case, however, the user program might still be running normally on the destination agent. If the status of a flexible job changes to *Ended abnormally*, take action according to the execution result of the user program.
- *Note on commands that do not support data input from the standard input file*  
You cannot execute commands that do not support data input from the standard input file. These commands include the Windows `timeout` command.
- *Note on stopping services*  
Do not stop the JP1/AJS3 Autonomous Agent service or JP1/AJS3 Autonomous AgentMessenger service on relay and destination agents while a flexible job is being executed. If you do so, the status of the flexible job will change to *Ended abnormally*. In this case, the job might remain in *Now running* status until the time specified for the FXJOB\_MONITOR\_TIMEOUT environment setting parameter elapses (default: 60 minutes).
- *Note on the working path for flexible job execution*  
You cannot change the working path for flexible job execution. The working path for flexible job execution is as follows:
  - In Windows, if the JP1/AJS3 installation folder is the default installation folder or is in a folder protected by the system:  
`%ALLUSERSPROFILE%\Hitachi\JP1\JP1_DEFAULT\JP1AJS2\tmp\AJSEFX_WORK_00000001`
  - In Windows, if the JP1/AJS3 installation folder is other than the above:  
`JP1/AJS3-installation-folder\tmp\AJSEFX_WORK_00000001`

- In UNIX:

```
/var/opt/jplajs2/tmp/AJSFX_WORK_00000001
```

- *Note on re-executing a flexible job*

If you re-execute or retry a flexible job when a load balancer is used, the execution hosts to which the job will be distributed are re-determined.

If you re-execute a flexible job when broadcast execution is used, the job is re-executed on all destination agents.

- *About use with a remote jobnet*

Using a flexible job with a remote jobnet is not recommended. If you use a flexible job with a remote jobnet, both the execution request of the flexible job and the definition of the remote jobnet are forwarded as shown below. This might make it difficult to identify the cause of any problem that occurs.

1. The definition of a remote jobnet on a manager host is forwarded to another manager host.
2. The destination manager host requests a relay agent to execute a flexible job.
3. The relay agent requests a load balancer to execute a program.
4. The load balancer requests the destination agent to execute the program.

- *Precaution applying if a passing information setting job is specified as the succeeding job of a flexible job*

The maximum size of a standard output file that a passing information setting job succeeding a flexible job can inherit is 4,096 bytes. If the size of the output data exceeds 4,096 bytes, only the first 4,096 bytes are available (data beyond this is ignored). The 4,096-byte limitation is applied on both the relay agent and the destination agent. Therefore, if the relay agent and the destination agent use different character encodings, the output data must not exceed 4,096 bytes in either character encoding. Note that if a multibyte character is split at the 4,096th byte, the multibyte character is replaced by a question mark (?).

- *Do not include a command that shuts down the OS in a flexible job*

Do not execute a flexible job that includes a command for shutting down the operating system of the local host, relay agent, or destination agent. Because Windows does not wait for JP1/AJS3 to stop before it shuts down, there might be damage to the JP1/AJS3 data files if Windows shuts down while JP1/AJS3 is running. If you want to shut down the OS from a job, use a Local Power Control job in conjunction with the JP1/Power Monitor.

To shut down the system manually, use the power control command (`aompwcon` command) of JP1/Power Monitor. If you want to shut down the system manually but do not have JP1/Power Monitor installed, stop the JP1/AJS3 service manually before shutting down the OS.

- *About checking of the job status on destination agents when broadcast execution is used*

If a flexible job is executed by broadcast execution, you cannot use JP1/AJS3 - View to check the execution status of the job on the destination agents. To manage the job status on the destination agents, use a standard job rather than a flexible job.

- *About execution results of broadcast execution*

When a flexible job is executed by broadcast execution in sync mode, the destination agents output the execution results to their respective standard error outputs. If you use JP1/AJS3 - View to check the execution result of the flexible job, the contents of those standard error outputs are merged, which might result in a large amount of data. Therefore, to prevent large data amounts, the size of each standard error output is limited to a maximum of 384 bytes. If the data that is output to a standard error output exceeds 384 bytes, only the first 384 bytes are output.

- *About the IP addresses used for broadcast execution*

To perform broadcast execution of a flexible job, the broadcast agent and each broadcast-execution destination agent must have IP addresses that can mutually communicate.

- *Notes on the job execution time when the broadcast execution function is used*

If broadcast execution is performed while some broadcast-destination agents cannot communicate for a length of time exceeding the value set by the environment setting parameter `FXBC_MANAGEDAGT_REMOVEDTIME`,

the job might take a long time to end normally (even if the job was executed properly and has finished) because of the following reason.

It takes time to judge whether a connection from a broadcast agent to a broadcast-destination agent has timed out. However, processing continues by deleting the broadcast-destination agents that are unable to communicate and treating these agents as if they do not exist.

- *Use the connection source restriction function*

If you use the connection source restriction function, connections from the manager host to the relay agent can be restricted by the agent connection permission configuration file on the relay agent. However, connections from the manager host or the relay agent to the destination agent cannot be restricted by the connection source restriction function.

For details about connection source restrictions, see 2.3.9 *Restricting hosts that can access JP1/AJS3* in the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*.

- *Note on length of a host name*

To use a flexible job, make sure that the following hosts have a host name that is not longer than 128 bytes:

- Manager host
- Relay agent
- Broadcast agent
- Destination agent

Notes applicable if the OS of the agent host is Windows:

- *About the UAC function*

If the Windows UAC function is enabled, all OS users, except the built-in user Administrator, will lose administrator permissions. Therefore, if the UAC function is enabled, you cannot use flexible jobs to execute any operations that require administrator permissions.

- *Avoiding a system resource shortage*

If you execute more than a specific number of jobs concurrently, you might encounter a shortage of a system resource (desktop heap) and an error might occur depending on the system environment. In this case, consider taking the following action:

- Change the JP1/AJS3 service account to a user account. By setting the JP1/AJS3 service account to a user account that differs from the accounts of other services and the user account of the logon user, you can use the system without sharing the desktop heap.
- You can decrease consumption of the desktop heap by setting the account of the OS user frequently used for flexible job execution to the same account as the JP1/AJS3 service account.

- *Using space characters in an application file name*

If an application file name with an extension associated with a file type contains a space character, check whether the file type is registered correctly in the File Types page in Explorer. Then, enclose the application file name with double quotation marks ("").

- *Use file names of 254 bytes or less in flexible jobs*

A job might enter *Failed to start* or *Ended abnormally* status if any of the following file names specified in the job is 255 bytes or longer:

- The executable file of the job
- The environment variable file for the job
- The end judgment file for the job

Make sure that the above file names have no more than 254 characters.

- *Note on a job that requires a user profile*

Because no user profile is loaded when a flexible job is run, a task that requires a user profile cannot be performed by a flexible job.



Notes applying if the OS of the agent host is Linux:

- *Return codes that can be set by flexible jobs*

Flexible jobs can set return code values from 0 to 255. If a job fails to start, or if acquisition of standard output data or standard error output data fails, the return code is set to -1.

- *Preventing flexible jobs from entering Failed-to-start status*

Check whether the user who starts the JP1/AJS3 service and the OS user who executes flexible jobs have read and right permissions on the following files and directories. If those users do not have those permissions, the status of flexible jobs might change to *Failed to start*. To prevent this, make sure that the user who starts the JP1/AJS3 service has read and write permissions on the following files and directories:

- Home directory of the OS user who executed the job
- Log files used for job execution control<sup>#</sup>

#

For details about the log files used in job execution control, see *1.2.5 List of log files and directories* in the manual *JP1/Automatic Job Management System 3 Troubleshooting*.

- *Resource limits when flexible jobs are executed*

In JP1/AJS3 for UNIX, the resource limits in effect at JP1/AJS3 startup apply when a job is executed. To apply resource limits, set them for the root user who starts JP1/AJS3. However, if you specify a limit in the flexible job to be executed, the value specified in the job takes effect. For details, see *20.5 Setting up the job execution environment* in the *JP1/Automatic Job Management System 3 Configuration Guide*.

The following is an example of changing file size limits:

1. In the login profile for the root user (usually `[/.profile]` (`$HOME/.profile`)), include the setting below:  
For *fsize*, specify the required file size. If you do not want to impose a limit, specify *unlimited*.  
`ulimit -f fsize`
2. Log in as the root user.
3. From the root account, start the JP1/AJS3 service.  
The *fsize* value takes effect.

#### Cautionary note

The resource limits you define in the resource settings file for the OS (`/etc/security/limits.conf`) apply only to processes started by users using the `login` command over a telnet connection or similar. Because flexible jobs started by JP1/AJS3 take the form of processes started by a service, the settings in the file do not apply.

- *Notes on the LANG environment variable when a flexible job is executed*

Set the same type of character encoding for the following LANG environment variables:

- The LANG environment variable when the JP1/AJS3 Autonomous Agent service starts
- The LANG environment variable when the JP1/AJS3 Autonomous Agent Messenger service starts
- The LANG environment variable for the login profile of the root user

- *Precautions applying when the JP1/AJS3 service starts automatically*

In a system where the JP1/AJS3 service starts automatically, the login profile of the root user is not loaded. For this reason, even if you change the resource limits in the login profile of the root user, different limits might apply when you log in manually and start the JP1/AJS3 service. In this case, set resource limits in the environment setting parameters of the job execution environment. For details about these parameters, see *20.5 Setting up the job execution environment* in the *JP1/Automatic Job Management System 3 Configuration Guide*.

You can also write resource limits into the automatic start script (`/etc/opt/jp1ajs2/jajs_start`) of JP1/AJS3. If you do so, test your settings thoroughly before using the system.

Note that flexible jobs might be executed under a group ID that differs from the group ID set for the root user at login. For details, see *5.4.12 Group ID for job execution (UNIX only)* in the manual *JP1/Automatic Job Management System 3 Overview*.

- *Executing a user program that requires a terminal as a job*

In the UNIX version of JP1/AJS3, if a user program that requires a terminal is executed as a job, the user program might not operate correctly (the flexible job might end abnormally).

- *Registering or changing an OS user*

While a flexible job is running, do not use the `passwd` command with administrator privileges to register or update the OS user associated with the job. Register or update the OS user information before you run the job.

Note applying if the OS of the manager host is Linux, and the relay agent is not specified or the relay agent host is Linux:

- *Note on login shells*

For the login shell of an OS user that is mapped to a JP1 user, `bash` or `sh` needs to be specified.

## 7.9 Notes on using HTTP connection jobs

---

This section provides notes on using HTTP connection jobs for each agent host OS.

Read this section in conjunction with *2.6.2 Troubleshooting problems related to standard jobs, HTTP connection jobs, action jobs, and custom jobs* in the manual *JP1/Automatic Job Management System 3 Troubleshooting*, which provides information about what might cause an HTTP connection job to fail to start or end abnormally, and cautionary notes about using HTTP connection jobs.

Notes applicable if the OS of the agent host is Windows:

- *Avoiding a system resource shortage*

If you execute more than a specific number of jobs concurrently, you might encounter a shortage of a system resource (desktop heap) and an error might occur depending on the system environment. In this case, consider taking the following action:

- Change the JP1/AJS3 service account to a user account. By setting the JP1/AJS3 service account to a user account that differs from the accounts of other services and the user account of the logon user, you can use the system without sharing the desktop heap.
- You can decrease consumption of the desktop heap by setting the account of the OS user frequently used for HTTP connection job execution to the same account as the JP1/AJS3 service account.

- *Preventing HTTP connection jobs from entering Failed-to-start status*

If the user who starts the JP1/AJS3 service does not have appropriate permissions, the status of HTTP connection jobs might change to *Failed to start* status. Make sure that the user who starts the JP1/AJS3 service is granted the following permissions:

- For the standard output file for the HTTP connection job: Read and write permissions
- For the standard error output file for the HTTP connection job: Read and write permissions

- *Use file names of 254 bytes or less in HTTP connection jobs*

A job might enter *Failed to start* or *Ended abnormally* status if any of the following file names specified in the job is 255 bytes or longer:

- The standard output file for the job
- The standard error output file for the job
- The end judgment file for the job

Make sure that the above file names have no more than 254 characters.

Notes applying if the OS of the agent host is Linux:

- *Preventing HTTP connection jobs from entering Failed-to-start status*

Check whether the user who starts the JP1/AJS3 service and the OS user who executes HTTP connection jobs have read and right permissions on the following files and directories. If those users do not have those permissions, the status of HTTP connection jobs might change to *Failed to start*. To prevent this, make sure that the user who starts the JP1/AJS3 service has read and write permissions on the following files and directories:

- Standard output file for the HTTP connection job
- Standard error output file for the HTTP connection job
- Home directory of the OS user who executed the job
- Log files used for job execution control<sup>#</sup>

#

For details about the log files used in job execution control, see *1.2.5 List of log files and directories* in the manual *JP1/Automatic Job Management System 3 Troubleshooting*.

- *Resource limits when HTTP connection jobs are executed*

In JP1/AJS3 for UNIX, the resource limits in effect at JP1/AJS3 startup apply when a job is executed. To apply resource limits, set them for the root user who starts JP1/AJS3. However, if you specify a limit in the HTTP connection job to be executed, the value specified in the job takes effect. For details, see *20.5 Setting up the job execution environment* in the *JP1/Automatic Job Management System 3 Configuration Guide*.

The following is an example of changing file size limits:

1. In the login profile for the root user (usually `[/.profile]` (`$HOME/.profile`)), include the setting below:  
For *fsize*, specify the required file size. If you do not want to impose a limit, specify `unlimited`.  

```
ulimit -f fsize
```
2. Log in as the root user.
3. From the root account, start the JP1/AJS3 service.  
The *fsize* value takes effect.

#### Cautionary note

The resource limits you define in the resource settings file for the OS (`/etc/security/limits.conf`) apply only to processes started by users using the `login` command over a telnet connection or similar. Because HTTP connection jobs started by JP1/AJS3 take the form of processes started by a service, the settings in the file do not apply.

- *Precautions applying when the JP1/AJS3 service starts automatically*

In a system where the JP1/AJS3 service starts automatically, the login profile of the root user is not loaded. For this reason, even if you change the resource limits in the login profile of the root user, different limits might apply when you log in manually and start the JP1/AJS3 service. In this case, set resource limits in the environment setting parameters of the job execution environment. For details about these parameters, see *20.5 Setting up the job execution environment* in the *JP1/Automatic Job Management System 3 Configuration Guide*.

You can also write resource limits into the automatic start script (`/etc/opt/jplajs2/jajs_start`) of JP1/AJS3. If you do so, test your settings thoroughly before using the system.

Note that HTTP connection jobs might be executed under a group ID that differs from the group ID set for the root user at login. For details, see *5.4.12 Group ID for job execution (UNIX only)* in the manual *JP1/Automatic Job Management System 3 Overview*.

- *Registering or changing an OS user*

While an HTTP connection job is running, do not use the `passwd` command with administrator privileges to register or update the OS user associated with the job. Register or update the OS user information before you run the job.

## 7.10 Notes on using custom event jobs

---

This section gives notes on using custom event jobs.

- Custom event jobs do not support operations that use execution agent groups. If a custom event job without a specified execution agent is included in a root jobnet or nested jobnet for which an execution agent group is specified, JP1/AJS3 will attempt to use the agent group specified for the jobnet as the execution agent for the custom event job. If an execution agent with the same name as the agent group exists, the custom event job will be executed by that execution agent. If there is no execution agent with the same name as the agent group, an error occurs and the following message is output to the integrated trace log: `KAVT0403-E The specified agent is not defined in the job execution environment. (host=exec-agent, maintenance-information)`. Therefore, if you want to specify an execution agent group for a root jobnet or nested jobnet, make sure that an execution agent is explicitly specified for the custom event job in the jobnet.
- You cannot specify the name of an execution agent group in the **Exec-agent** field of the detailed definition of a custom event job.
- For a custom event job waiting for a start condition to be satisfied, if JP1/AJS3 - Manager stops during start condition monitoring, you can resume event monitoring after JP1/AJS3 - Manager restarts. If multiple events were being monitored for a start condition, you can hold the reception information of the events that satisfy the condition even after JP1/AJS3 - Manager restarts.
- When a custom event job is defined at the beginning of a jobnet, the jobnet is executed after the condition is satisfied, in the same way as a jobnet with a start condition. A jobnet with a start condition stays in *Wait for start cond.* status while event reception is being monitored. A jobnet with a custom event job defined at the beginning is kept in *Now running* status while event reception is being monitored. When you define a custom event job at the beginning of a jobnet, we recommend that you use the custom event job to wait for events you expect to occur.
- If a timeout period is specified for a custom event job, the counting is done by the execution host. Therefore, when event monitoring resumes after the execution host is restarted due to a power failure or similar condition, counting restarts upon the restarting of the execution host.

Note that you can verify the restart of counting and the restart time in the message `KAVT0603-W Elapsed time since restart-time is used for time-out due to temporary interruption of monitoring.` that is output in the custom event job execution result details.

If you want to stop monitoring at an absolute time regardless of the status of the agent host, define a custom event job as a jobnet start condition, and then specify the effective range of the start condition by using an absolute time. For details about a start condition, see *3.4 Defining a start condition* in the manual *JP1/Automatic Job Management System 3 Overview*.

- When using custom event jobs, system load or temporary network failure might cause a time lag between the job execution time and the time that the execution agent actually starts monitoring the events. The system detects only events that occur after event monitoring begins. Therefore, you must provide some leeway when executing a custom event job, to allow the system to be ready and monitoring when a target event occurs.
- When you execute custom event jobs (including those with a start condition), definition data for the executed custom event jobs and information about events that satisfy monitoring conditions is transmitted between processes such as the event/action control manager and the event/action control agent. If transmission fails due to some problem such as a temporary network error or because the remote process is busy, the information is saved to a file and transmission is attempted again after a predetermined interval. In JP1/AJS3, this is referred to as an *unreported information file*. At successful re-transmission, the information is deleted.
- In a manager/agent configuration, if the event/action control manager and the event/action control agent are unable to communicate due to a network error or other problem, inconsistencies in custom event job monitoring (including custom event jobs defined as start conditions) can occur if you perform any of the operations listed below. In this case, the custom event job will continue monitoring on the agent despite having ended on the manager.
  - Killing a custom event job

- Killing a jobnet that has a start condition
- Changing the status of a custom event job to *Ended*

These operations can result in problems such as failure to restart the custom event job where the inconsistency occurred, and delays to processing of other normal custom event jobs.

For this reason, if you perform one of the above operations in a system affected by a network error or similar problem, execute the `jpomanjobshow` command on the manager host, and the `jpogtjobshow` command on the agent host. Compare the results of the commands to see whether any custom event jobs that have ended on the manager are still monitoring on the agent.

If you discover a custom event job that is in *Now monitoring* status only on the agent host, restart the JP1/AJS3 service on the agent host, and then terminate the custom event job that is still monitoring on the agent.

## 7.10.1 Notes on defining passing information

Note the following when defining passing information:

- Passing information can be passed even if the custom event job and the succeeding job are executed on different agent hosts. If the hosts use different character codes, the passing information is converted to the character codes used on the succeeding host. Note, however, that if the passing information contains unsupported characters, JP1/AJS3 might not operate correctly.
- For details about the characters you can use with JP1/AJS3, see *2.4.2 Language type and character encoding of the system* in the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*. Characters that are not supported by JP1/AJS3 cannot be used even in a stand-alone configuration. Make sure that passing information does not include such characters.
- If multiple custom event jobs are related to a single succeeding job, the succeeding job can inherit the information received by all the custom event jobs. However, if multiple custom event jobs define the same macro variable name, the older received information will be overwritten.
- If the same macro variable name is defined more than once in the passing information of a single custom event job, the information that is defined first is passed.

Example:

Suppose that the following two macro variables are defined in passing information:

- `?AJS2111?:OBJECT01` (This specifies that the information defined in OBJECT01 is passed to the `?AJS2111?` macro variable.)
- `?AJS2111?:OBJECT02` (This specifies that the information defined in OBJECT02 is passed to the `?AJS2111?` macro variable.)

In this case, `?AJS2111?` inherits OBJECT01.

- Do not pass data that contains an escape sequence to the command line. If you pass data that contains a space character, unexpected behavior might result. To prevent this, when you define a macro variable, enclose it in double quotation marks ("").
- When you specify a macro variable in the command line of the succeeding job, the succeeding job cannot correctly inherit passing information if the information contains space characters or single quotation marks ('). Note that macro variables are executed on the agent host that executes the succeeding job of the custom event job by replacing the macro variable in the command line with its value. When defining passing information, define only information that can be treated as command arguments at execution.
- If there is no information to be inherited from the custom event job or if the custom event job is not executed, the macro variables defined in the succeeding job will not inherit any information. In this case, if you define `?AJS2111?` as the macro variable name when you execute the job, the character string `?AJS2111?` will be passed.

- When you pass the information received in a custom event job to the parameters of a standard job or an action job, if the data to be passed contains a double quotation mark ("), you must add a backslash (\) before the double quotation mark. If you do not add a \ before the double quotation mark, the double quotation mark will be ignored when it is passed to a standard job or an action job.

To prevent this, enable the option for handling data containing double quotation marks (as is) in passing information. For details about how to set this option, see *6.3.4 Passing event data containing double quotation marks* in the *JP1/Automatic Job Management System 3 Configuration Guide* (for Windows systems) or *15.3.4 Passing event data containing double quotation marks* in the *JP1/Automatic Job Management System 3 Configuration Guide* (for UNIX systems).

- When passing information is used in a command line of a job, even if the passing information contains a double quotation mark ("), the information is passed to the succeeding job without being converted. Therefore, the job might not be executed correctly depending on the command line restrictions of the OS in question. If you want to use passing information that contains a special character, do not use it directly in a command line. Such information must be passed to an environment variable.
- If you define macro variables in a custom event job defined in a start condition, the information is passed to the entire jobnet that starts when the start condition is satisfied.
- Make sure that the macro variable names and passing information together do not exceed 4,096 bytes. In particular, if you use the AND condition to define a start condition, the system merges the macro variable names and passing information for all of the custom event jobs in the start condition. For this reason, take care that the macro variable names and passing information do not exceed 4,096 bytes in total.

## 7.10.2 Notes on restarting the JP1/AJS3 service while custom event jobs are running

If you perform any of the operations listed below while custom event jobs (including those with a start condition) are running, communication and correlation processing take place between the schedule control and event/action control functions to ensure consistency in job statuses between the functions. If a large number of custom event jobs are in *Now running* status, the amount of data to be processed places a considerable load on the system.

Operations that result in high system load

- Stopping the scheduler service, and then performing a warm or hot start.
- Stopping the JP1/AJS3 service on the manager host, and then performing a warm or hot start.
- Stopping and then restarting the JP1/AJS3 service on the agent host.
- Executing the `jajs_maintain` command.

Use the following workarounds to avoid placing a high load on the system:

Workarounds

- Before you perform any of the above operations, forcibly terminate any jobnets that use custom event jobs. Register them for execution again when the operation is complete.
- Before you perform any of the above operations, forcibly terminate all custom event jobs. Re-execute them when the operation is complete.
- Do not run any jobs until 30 minutes to an hour has elapsed after performing the operation.
- Do not trigger any monitored events until 30 minutes to an hour has elapsed after performing the operation.

The following problems can occur when a high load is placed on the system.

## Problems resulting from a high system load

1. Custom event jobs (including those with a start condition) registered for execution immediately after you performed one of the above operations take a long time to enter *Now running* status.<sup>#1</sup>
2. When you forcibly terminate a custom event job in *Now running* status, or a jobnet with a start condition in *Now monitoring* status, it takes a long time to terminate.<sup>#1</sup>
3. When you change the status of a custom event job in *Now running* status, the change takes a long time to take effect.<sup>#1</sup>
4. It takes a long time for an event to be detected when a monitoring condition is satisfied.<sup>#1</sup>
5. Custom Event jobs (including those with a start condition) registered for execution immediately after you performed one of the above operations remain in *Queuing* status.<sup>#2</sup>
6. You attempt to forcibly terminate a custom event job in *Now running* status, or a jobnet with a start condition in *Now monitoring* status, but it does not terminate.<sup>#2</sup>
7. A monitoring condition is satisfied but no event is detected.<sup>#2</sup>

### #1

Problems 1 through 4 can occur when you perform an operation that results in a high system load while the event/action control manager is near its resource limits. For details about the resource limits that apply to event/action control, see *B.8 Limits for the event/action control* in the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*.

### #2

Problems 5 through 7 can occur when you perform an operation that results in a high system load after the event/action control manager has exceeded its resource limits. For details about the resource limits that apply to event/action control, see *B.8 Limits for the event/action control* in the *JP1/Automatic Job Management System 3 System Design (Configuration) Guide*.

If you perform an operation that results in a high system load when a large number of custom event jobs are running, a large volume of communication takes place between the scheduler control and event/action control functions. This results in an increased number of unreported items of information that are generated and managed in the event of a communication error. JP1/AJS3 imposes a limit on the number of unreported item that can be kept, to prevent the high system load that results from processing this data from monopolizing system resources and delaying job execution and event detection. When the number of unreported items reaches the limit, the information is deleted starting with the oldest item. Any of problems 5 through 7 can occur as a result, depending on the content of the deleted data.

The upper limit of the number of unreported items of information that can be kept is not disclosed to JP1/AJS3 users. Instead, the limit for event/action control is calculated from the number of unreported items generated by an operation that results in a high system load. When using JP1/AJS3, be careful not to exceed this limit.

If any of the problems mentioned above occurs, take the following remedial action:

### Recovery procedure

#### For problems 1 to 4

Wait until the processing finishes. This can take 30 minutes to an hour, depending on the number of jobs affected.

#### For problem 5

Forcibly terminate the custom event job or jobnet in question, and then re-register it for execution.

#### For problem 6

For a custom event job, change the status of the job and terminate it. For a jobnet with a start condition, forcibly terminate the jobnet again.



For problem 7

Forcibly terminate the custom event job or jobnet in question, and then re-register it for execution. Then, generate the event again.

## 7.11 Notes on defining jobs

---

This section gives cautionary notes on jobnet definition.

### 7.11.1 Notes on the standard output file and standard error output file

The following cautionary notes relate to the standard output file and standard error output file.

#### (1) When defining the standard output file and standard error output file

Note the following when defining the standard output files and standard error output file output at job execution (of PC and Unix jobs including queueless jobs, flexible jobs and HTTP connection jobs):

1. By default, jobs executed from jobnets acquire the content of the standard error output to serve as their execution result. Their execution results are not acquired from the standard output. You can view the contents of the standard error output in the Execution Result Details dialog box of the JP1/AJS3 - View window. This window also displays the error messages output to the standard error output by JP1/AJS3.
2. To output the contents of the standard output and standard error output to files of your choosing, specify the file names in the **Standard output** and **Standard error** fields when defining the job.
3. To also have the contents of the standard output displayed in the Execution Result Details dialog box of the JP1/AJS3 - View window, specify the same file name in the **Standard output** and **Standard error** fields when defining the job.
4. If you specify the same file name in the **Standard output** and **Standard error** fields, use the same **Append** setting for both files. If you specify the option to create a new file for one file and the option to add data to an existing file for the other, a parameter error occurs and the job fails to register for execution.
5. If you specify the same name as the script file name or the execution file name as the environment variable file name, the standard input file name, the standard output file name, or the standard error output file name in the Define Details - [unit-name] window, the job might terminate abnormally. Do not use the same file name as the script file or the execution file for any of these files.
6. If you specify the same standard output file name or standard error output file name for jobs that are executed concurrently, the older output results in the standard output or the standard error output will be overwritten by the new output results. Also, when multiple jobs that use the standard output or standard error output run concurrently, the older output results will be overwritten by the new output results.  
Specify a different standard output file name or standard error output file name for each of the jobs to be executed concurrently.
7. If you specify an inaccessible network file name as the standard output file name or the standard error output file name, the job might fail to start or end abnormally. Specify a correct network file name.
8. To specify the name of a network file as the standard output file name or standard error output file name, you must have the following permission for the network file:

#### In Windows

The account for the JP1/AJS3 service (or the JP1/AJS3 Queueless Agent service for queueless jobs) must have Create, Read, and Modify permissions for the file.

#### In UNIX

The account specified as the OS user who executes the job must have Create, Read, and Modify permissions for the file.

9. Specifying a file name in the **Standard output** field might make the jobnet run slower than it would if no file name were specified.

## Supplementary note

By default, submit jobs executed using the `jpqjbsub` command do not save data to standard output or standard error output. Specify the desired file name in an option of the `jpqjbsub` command.

## (2) When outputting large quantities of data to the standard output or standard error output file

In JP1/AJS3, when a job ends (meaning a standard job, action job, or custom job, but not queueless jobs), the standard output file and standard error output files are transferred from the agent host to the manager host. If you execute a job that outputs large quantities of data (several megabytes or more) to the standard output or standard error output file, transferring and analyzing the data can place a high demand on system resources. The additional CPU and memory resources such operations require can delay job execution, and even slow down the entire system. This problem affects the manager host and the agent host.

If you specify the option to append data, output data accumulates with each executed job. This markedly increases the size of the transferred file. In this situation as well, transferring and analyzing the data can place a high demand on system resources, which can cause jobs to terminate abnormally or delay the transfer of data from the agent host to the manager host. In this case, we recommend that you disable the append option. Alternatively, you can periodically back up and delete the standard output and standard error output data files.

In JP1/AJS3, you can set an upper limit on the sizes of the standard output file and standard error output file so that excess data is discarded or a warning message is output when the upper limit has been reached. In this way, you can control the file sizes so that the processing of only a few jobs will have little effect on overall system performance. For details about how to set a maximum file size for the standard output or standard error output file, see *6.2.7 Placing restrictions on file reception* in the *JP1/Automatic Job Management System 3 Configuration Guide* (for Windows systems) or *15.2.7 Placing restrictions on file reception* in the *JP1/Automatic Job Management System 3 Configuration Guide* (for UNIX systems).

In addition, you can also specify the settings on the agent host so that excess data is discarded or a warning message is output when the upper limit has been reached. In this way, you can also control the file sizes so that the processing of only a few jobs will have little effect on overall system performance. For details about how to specify the settings in Windows, see *6.2.27 Placing restrictions on file transmission* in the *JP1/Automatic Job Management System 3 Configuration Guide*. For details about how to specify the settings in UNIX, see *15.2.24 Placing restrictions on file transmission* in the *JP1/Automatic Job Management System 3 Configuration Guide*.

If you have not imposed a limit on the sizes of the standard output file and standard error output file, and a job is not operating correctly, take appropriate action as follows:

- When a large amount of data is output to the standard output  
For a PC job or Unix job, the standard output file is transferred only when the file name is explicitly specified in the job definition. If a large amount of data is transferred as the standard output file, the job might end abnormally. If the job ends abnormally, do not specify anything as the standard output file name, and redirect the standard output from a batch file or shell script within the job.  
If you do not specify a standard output file name in the job definition, the standard output file is not transferred to the manager host.
- When a large amount of data is output to the standard error output  
Specify a NULL device as the standard error output file name in the job definition. If the job is to be executed in UNIX, specify `/dev/null`; if the job is to be executed in Windows, specify `NUL`. If a standard error output file name is explicitly specified in the job definition, use a batch file or shell script to redirect the standard error output. When you specify these settings, the standard error output file is not transferred to the manager host, and you will be unable to view the contents of the file in the Execution Result Details dialog box of JP1/AJS3 - View.

## Supplementary notes

1. If the problems described above occur, a file named `A_JPQ*_job-number` might remain in the job execution environment work directory of agent host. The agent host job execution environment work directory is a folder specified for the `WorkPath` environment setting parameter of `[ {JP1_DEFAULT | logical-host-name } \JP1NBQAGENT\Process ]`. For details about the `WorkPath` environment setting parameter, see 20.5.2(37) *WorkPath* (for agent process) in the *JPI/Automatic Job Management System 3 Configuration Guide*.

You can safely delete files whose file name contains the job number of a job that ended abnormally.

2. When you use submit jobs that output large amounts of data to the standard output or standard error output, the amount of data in the temporary files created in the work directory of the job execution environment of the manager host (`M_JPQSTDE_job-number` or `M_JPQSTDE_job-number`) also increases. Usually, when the setting determining how many days to retain job information is exceeded for a job, temporary files for that job are deleted when job information deletion processing is performed. However, if the disk becomes full due to the temporary files, you can delete them manually after the job terminates. If the disk often becomes full, consider reducing the number of days job information is retained, or redirecting the standard output or standard error output from a script file in the job.

Note that once you delete a temporary file, you can no longer use the `jpqjobget` command to view the standard output and standard error output files. The manager host job execution environment work directory is a folder specified for the `WorkPath` environment setting parameter of `[ {JP1_DEFAULT | logical-host-name } \JP1NBQMANAGER\Process ]`. For details about the `WorkPath` environment setting parameter, see 20.5.2(1) *WorkPath* (for manager process) in the *JPI/Automatic Job Management System 3 Configuration Guide*.

The following tables show examples of specifying redirection as **Parameter** in the detailed definition of a job.

**Table 7–8: Examples of specifying redirection as Parameter in the detailed definition of a job (in Windows)**

| Job definition  | Items to be specified  | Specification   |
|---|--|---|
| The names of the standard output file and standard error output file are different, and the executable file is an exe file. | Executable file: <code>test.exe</code><br>Parameters: None<br>Standard output file: <code>out.txt</code><br>Standard error output file: <code>err.txt</code> | Executable file: <code>cmd.exe</code><br>Parameters:<br><code>/C test.exe &gt;out.txt 2&gt;err.txt</code><br>Standard output file: None<br>Standard error output file: None |
| The names of the standard output file and standard error output file are different, and the executable file is an bat file. | Executable file: <code>test.bat</code><br>Parameters: None<br>Standard output file: <code>out.txt</code><br>Standard error output file: <code>err.txt</code> | Executable file: <code>test.bat</code><br>Parameters: <code>&gt;out.txt 2&gt;err.txt</code><br>Standard output file: None<br>Standard error output file: None               |
| The names of the standard output file and standard error output file are the same.  | Executable file: <code>test.bat</code><br>Parameters: None<br>Standard output file: <code>out.txt</code><br>Standard error output file: <code>out.txt</code> | Executable file: <code>test.bat</code><br>Parameters: <code>&gt;out.txt 2&gt;&amp;1</code><br>Standard output file: None<br>Standard error output file: None                |

**Table 7–9: Examples of specifying redirection as Parameter in the detailed definition of a job (in UNIX)**

| Job definition  | Items to be specified   | Specification  |
|---|---|--|
| The names of the standard output file and standard error output file are different. | Script file: <code>test.sh</code><br>Parameters: None<br>Standard output file: <code>out.txt</code><br>Standard error output file: <code>err.txt</code> | Script file: <code>test.sh</code><br>Parameters: <code>&gt;out.txt 2&gt;err.txt</code><br>Standard output file: None<br>Standard error output file: None |

| Job definition   | Items to be specified   | Specification   |
|--|---|---|
| The names of the standard output file and standard error output file are the same. | Script file: <code>test.sh</code><br>Parameters: None<br>Standard output file: <code>out.txt</code><br>Standard error output file: <code>out.txt</code> | Script file: <code>test.sh</code><br>Parameters: <code>&gt;out.txt 2&gt;&amp;1</code><br>Standard output file: None<br>Standard error output file: None |

### (3) About restrictions placed on reception and transmission of the standard output file and standard error output file

If you execute a job that outputs a large amount of data (for example, a few megabytes or more) to the standard output file or standard error output file, file data analysis or transfer will result in a heavy load. As a result, the CPU usage or the amount of memory used by JP1/AJS3 increases. This might cause job execution to delay, adversely affecting the overall system performance. To prevent this, in JP1/AJS3 11-00 or later, by default, the manager host places restrictions on file reception, and the agent host places restrictions on file transmission.

File reception and transmission restrictions in JP1/AJS3 work by truncating sent or received data at a certain file size, and outputs a message indicating that the upper limit was reached. The maximum size of a file received on the manager host is 5 MB (5,242,880 bytes), and the maximum size of a file sent from the agent is 3 MB (3,145,728 bytes).

You can change the action taken by the system when the maximum file size is exceeded, and the maximum file size, on the manager host. For details about how to change the settings in Window, see *6.2.7 Placing restrictions on file reception* in the *JP1/Automatic Job Management System 3 Configuration Guide*. For details about how to change the settings in UNIX, see *15.2.7 Placing restrictions on file reception* in the *JP1/Automatic Job Management System 3 Configuration Guide*. You can change the action taken by the system when a maximum file size is exceeded, and the maximum file sizes, on the agent host. For details about how to change the settings in Windows, see *6.2.27 Placing restrictions on file transmission* in the *JP1/Automatic Job Management System 3 Configuration Guide*. For details about how to change the settings in UNIX, see *15.2.24 Placing restrictions on file transmission* in the *JP1/Automatic Job Management System 3 Configuration Guide*.

#### Supplementary note

If you upgrade-install JP1/AJS3 on the manager host, the new installation will inherit the same restrictions on file reception that were set in the environment setting parameters of the previous installation. If you upgrade-install JP1/AJS3 on the agent host, JP1/AJS3 outputs a message indicating that the maximum file size is reached but does not truncate large data at the maximum file size.

### (4) Other notes

1. You can specify only text files as the output destination for standard output and standard error output data.
2. Make sure that machine-dependent characters are not output in the standard output and standard error output data. JP1/AJS3 - View cannot display these characters correctly.
3. Standard output data and standard error output data is saved to temporary files on the agent host, which are transferred to the manager host when the job has terminated. These files are created with the following names under the work directories in the job execution environment on both the manager and agent hosts.

Manager host (submit jobs only)

`M_JPQSTDE_<job-number>`

`M_JPQSTDO_<job-number>`

Agent host

`A_JPQSTDE_*_<job-number>`

`A_JPQSTDO_*_<job-number>`

The following describes when the files are deleted from each host.

Manager host:

Temporary files for QUEUE jobs and submit jobs are deleted when job information deletion processing is performed. For other jobs, the temporary files are deleted when the job ends.

Agent host:

The temporary files are deleted automatically when the job ends.

For queueless jobs, only the standard error output data is stored in a temporary file on the agent host. The data in this file is transferred to the manager host when the queueless job terminates. The file is deleted automatically after the queueless job terminates or when the queueless agent service starts.

The standard output data and standard error output data for user programs executed by flexible jobs are always forwarded to the manager host. You can check the names of forwarded files by executing the `ajsshows` command by specifying `%s0` for the `-i` option.

4. When you display the Execution Result Details dialog box of JP1/AJS3 - View after executing a job for which a standard error output file name is specified and **Append** is selected, the displayed information differs depending on the destination service of the job.
  - When **Standard** is specified as the **Exec. Service**:  
The content of the file specified in **Standard error** is displayed.
  - When **Queueless Agent** is specified as the **Exec. Service**:  
The content output to standard error output at job execution is displayed.

## 7.11.2 Notes on the execution priority of jobs

JP1/AJS3 specifies the execution priority for jobs (PC jobs including queueless jobs, Unix jobs, HTTP connection jobs, QUEUE jobs executed in JP1/AJS3, and custom jobs). For flexible jobs and action jobs, the execution priority is specified by setting a priority for the upper-level jobnet.

You can set **None**, **1**, **2**, **3**, **4**, or **5** as the execution priority. If the execution priority of a job and of all upper-level jobnets is set to **None**, the execution priority of that job is assumed to be the value specified for the `DEFAULTPRIORITY` environment setting parameter. If the execution priority of a job executed from JP1/AJS3 and of all upper-level jobnets is set to **None**, and if no value is specified for the `DEFAULTPRIORITY` environment setting parameter, then the execution priority of that job is assumed to be 1. For details about the `DEFAULTPRIORITY` environment setting parameter, see 20.4.2(108) `DEFAULTPRIORITY` in the *JP1/Automatic Job Management System 3 Configuration Guide*. See the following notes to determine whether you need to change the job and the jobnet execution priority for your system environment and operating conditions.

The following subsections describe the values that JP1/AJS3 sets for jobs according to the execution priority of jobs in Windows and UNIX.

### (1) Job execution priorities and the priority values that JP1/AJS3 assigns to jobs (in UNIX)

In UNIX, JP1/AJS3 assigns a nice value that corresponds to the execution priority of the job. The base value from which a job's execution priority is determined differs depending on which of **Standard** or **Queueless Agent** is specified as the execution service.

## (a) When Standard is specified as the execution service

When **Standard** is specified as the execution service for the job, the nice value of a job executed by JP1/AJS3 is based on the nice value<sup>#</sup> of the JP1/AJS3 service when it starts (when you executed the `jajs_spmd` command). A job's nice value is assigned by adding to or subtracting from this base value, according to the execution priority specified for the job.

#

The nice value at startup of the JP1/AJS3 service is the default value assigned by the OS when it started the process. The maximum and minimum nice values that can be assigned depend on your operating system. For details about the default nice value assigned by the OS, and the maximum and minimum assignable values, see the documentation for your operating system and the UNIX reference documentation.

The following table shows an example of the correspondence between job execution priorities specified in JP1/AJS3 and the nice values assigned to jobs. In this example, the operating system is HP-UX, **Standard** is specified as the execution service for the job, and the default nice value to the JP1/AJS3 service at startup is 20.

Table 7–10: Job execution priorities and the nice values assigned to jobs (when Standard is specified as the execution service)

| Execution priority | nice value assigned to the job | When the nice value of JP1/AJS3 service is 20 |
|--------------------|--------------------------------|---|
| 1                  | nice value + 20                | 39  |
| 2                  | nice value + 10                | 30  |
| 3                  | nice value                     | 20  |
| 4                  | nice value - 10                | 10  |
| 5                  | nice value - 20                | 0   |

If no value is specified for the execution priority of a job, the execution priority of that job is assumed to be the value specified for the `DEFAULTPRIORITY` environment setting parameter. In this case, the job is assigned a nice value of 39. This value is determined by adding 20 to the nice value of the JP1/AJS3 service (20 by default). System restrictions in this case mean that nice values are capped at 39, and cannot be lower than 0.

## (b) When Queueless Agent is specified as the execution service

When **Queueless Agent** is specified as the execution service, the job's nice value is not based on that of the JP1/AJS3 service. In this case, JP1/AJS3 sets 39, 30, 20, 10, or 0 (in the Linux, JP1/AJS3 sets 19, 10, 0, -10, or -20) as the nice value according to the execution priority, where 0 corresponds to the highest priority. To change a job's nice value, specify the job execution priority that corresponds to the nice value that you want to assign.

The following table shows the correspondence between job execution priorities specified in JP1/AJS3 and the nice values assigned to jobs, when **Queueless Agent** is specified as the execution service.

Table 7–11: Job execution priorities and the nice values assigned to jobs (when Queueless Agent is specified as the execution service)

| Execution priority | nice value assigned to job <sup>#</sup> |
|--------------------|---|
| 1                  | 39                                      |
| 2                  | 30                                      |
| 3                  | 20                                      |
| 4                  | 10                                      |

| Execution priority | nice value assigned to job# |
|--------------------|-----------------------------|
| 5                  | 0                           |

#

In Linux, nice values in the range from -20 to 19 can be assigned. JP1/AJS3 sets 19, 10, 0, -10, or -20 according to the execution priority, where 19 corresponds to the lowest priority.

## (2) Job execution priorities and the values that JP1/AJS3 assigns to jobs (in Windows)

In Windows, JP1/AJS3 sets one of three job execution priorities. The following table shows the correspondence between job execution priorities and the base priorities of Windows. You can use Task Manager to check the base priority.

Table 7–12: Job execution priorities and Windows base priorities

| Job execution priority | Base priority assigned to job |
|------------------------|-------------------------------|
| 1                      | Low                           |
| 2                      |                               |
| 3                      | Normal                        |
| 4                      | High                          |
| 5                      |                               |

The following describes the priority class that JP1/AJS3 sets based on the job execution priority when it starts the job process.

- If you set **1** or **2** as the execution priority value for a job, the job is executed when the system is idle (the Windows `IDLE_PRIORITY_CLASS` is assigned). This priority is lower than the priority of interactive processing.
- If you set **3** as the execution priority value for a job, the job is executed as a general process (the Windows `NORMAL_PRIORITY_CLASS` is assigned). This priority is the same as the priority of interactive processing.
- If you set **4** or **5** as the execution priority value for a job, the job is executed before the threads of the processes assigned the above priority classes (the Windows `HIGH_PRIORITY_CLASS` is assigned). This priority is higher than the priority of interactive processing.

## (3) Changing the job execution priority

The execution priority of a process started directly from a service or console and not through JP1/AJS3 depends on the operating system. In Windows, the process is assigned the *Normal* base priority, which is equivalent to the priority of interactive processing. In UNIX, the process is executed with a nice value of 20 (in Linux, with a nice value of 0). If you want jobs executed from JP1/AJS3 to have the same execution priority as these processes, you must set the job execution priority to 3. Use either of the following methods to change a job's execution priority.

In these examples, the job execution priority is set to **3**:

### (a) For PC jobs including queueless jobs, Unix jobs, HTTP connection jobs, QUEUE jobs executed in JP1/AJS3, and custom jobs.

You can change the execution priority by using either of the following methods:

1. In the Define Details - [*unit-name*] dialog box for the job in question, on the **Definition** page, specify **3** in **Priority**.



2. In the Define Details - [unit-name] dialog box for the job in question, on the **Definition** page, specify **None** in **Priority**. Then, in the Define Details - [jobnet] dialog box for the upper-level jobnet, such as the root jobnet, on the **Definition** page, specify **3** in **Priority**.

### (b) For flexible jobs and action jobs

You can change the execution priority by using the following method:

1. In the Define Details - [jobnet] dialog box for the upper-level jobnet (for example, the root jobnet), on the **Definition** page, specify **3** in **Priority**.

### (c) If the execution priority of a job and of all upper-level jobnets is set to None

You can change the default execution priority level of a job by using the following method:

1. Specify **3** in the DEFAULTPRIORITY environment setting parameter.  
For details about procedure for setting the environment setting parameter, see *4.2 Environment setting parameter settings* in the *JP1/Automatic Job Management System 3 Configuration Guide* (for Windows) or *14.2 Environment setting parameter settings* in the *JP1/Automatic Job Management System 3 Configuration Guide* (for UNIX).
2. In the Define Details - [unit-name] dialog box for the job in question and in the Define Details - [jobnet] dialog box for the upper-level jobnet (for example, the root jobnet), on the **Definition** page, specify **None** in **Priority**.

### (d) Cautionary note

In UNIX, if a user without superuser permission tries to execute a job with a job execution priority of **4** or **5**, a permission error occurs.

Also, if you assign jobs a higher execution priority than other non-job processes, some jobs might monopolize system resources and slow down the entire JP1/AJS3 system. This is the opposite situation to that described in *2.1.4(2) Execution priority*. Carefully consider the system environment and operating conditions before you assign jobs a higher priority than other processes.

## 7.11.3 Checking the return code of a job

You can check the return code of a job in the Monitor Details - [unit-name] dialog box of JP1/AJS3 - View, or from the value of EXITCODE in the job information output by the jpqjobget command. Under normal circumstances, 0 is set as the return code when a job terminates normally. If the job terminates abnormally, the return code of the job process is set. However, in the following cases, the return code is set by JP1/AJS3.

Table 7–13: Return codes set by JP1/AJS3 and associated conditions

| Condition  | Return code set by JP1/AJS3 |
|--|-----------------------------|
| If one of the following statuses occurs in a QUEUE job for which JP1/AJS3 is specified as a host, a PC job, a Unix job, a flexible job, an HTTP connection job, an action job, or a custom job: <ul style="list-style-type: none"> <li>Failed to start</li> <li>Killed (including jobs killed due to their timeout period elapsing)</li> <li>Ended abnormally<sup>#</sup></li> </ul> | -1                          |
| If an unexpected JP1/AJS3 error occurs during execution of a Unix job.   | 255                         |

#

If a problem occurs during post-processing by JP1/AJS3 after a job process has terminated (for example, if the result file could not be transferred to the manager host), the return code of the job process is overwritten with -1 by JP1/AJS3 and the job process terminates abnormally.

#### Cautionary notes

- For flexible jobs executed on Windows hosts and PC jobs, the job processes started by these jobs themselves can return -1 as a return code. For flexible jobs executed on Linux hosts and Unix jobs, if job processes end with a negative value, the return code is calculated by adding 256 to the negative value. For example, if a job process ends with -1, the return code is  $256 - 1 = 255$ . In a user application, if you need to determine whether the return code was set by the job process or JP1/AJS3, do not use a return code of -1 or 255 in your user applications.
- For flexible jobs executed on Linux hosts and Unix jobs, a login script is executed prior to job execution. Therefore, if processing terminates during execution of the login script, the return code for the login script is set. Make sure that processing does not terminate abnormally during execution of the login script. For details about how to change the login script, see *13.4.2 Changing the login scripts* in the *JP1/Automatic Job Management System 3 Configuration Guide*.
- For a flexible job, if either of the following options is selected for the end judgment method and the selected end condition is not satisfied, the return code is set to 1:
  - Normal if specified file exists
  - Normal if file was updated

In some cases the return code of a job is set by the OS, rather than by a user application or JP1/AJS3. The following table gives examples of typical return codes. Because upgrading your OS might change these return codes, check the technical information for your OS.

Table 7–14: Examples of return codes set by the OS

| OS      | Return code | Cause   | Action to take   |
|---------|-------------|---|--|
| Windows | 259 or -1   | The process might have failed to open the results file. In this case, one of the following occurs: <ul style="list-style-type: none"><li>• Message KAVU3284-W is output: A system call error occurred in the internal process (<i>logical-host-name</i>) (reason module: <i>reason-module</i>, reason code 1: 0x2013000a, system call name: CreateFile, reason code 2: <i>reason-code-2</i>)</li><li>• The following message is output to the standard error output: The process cannot access the file. The file is being used by another process.</li></ul> | As the standard output file or standard error output file when you register a job, do not specify a file opened from within a program executed as a job or opened by redirection from a batch file. However, where the file is opened from within the program by a function call, you can work around the problem by opening the file with a setting that permits shared read and write modes. |
|         | 128         | There might be a desktop heap shortage. If so, one of the following messages is output: <ul style="list-style-type: none"><li>• XXXX.XXX - application error: application not initialized correctly;</li></ul>  | To reduce use of the desktop heap, use the same user account as the JP1/AJS3 service to execute jobs.  |

| OS      | Return code         | Cause  | Action to take  |
|---------|---------------------|--|---|
| Windows | 128                 | <ul style="list-style-type: none"> <li>Failed to initialize <i>XXXX.XXX</i></li> </ul> | To reduce use of the desktop heap, use the same user account as the JP1/AJS3 service to execute jobs.   |
| UNIX    | Signal number + 128 | A program started from a job process might have received a signal.#                    | Take one of the following actions: <ul style="list-style-type: none"> <li>Set up the program started from a job process so that it does not receive signals.</li> <li>Set up a signal handler or similar for the program to provide additional processing that sets signal-dependent return codes.</li> </ul> |

#

If the job process itself receives a signal, jobs with **Standard** specified as the execution service will change to the *Killed* status with the return code -1. Jobs with **Queueless Agent** specified as the execution service will change to *Ended abnormally* status with the return code -1.

## (1) Return codes of flexible jobs

The following table describes the return codes of flexible jobs.

Note that if a flexible job is executed by broadcast execution, the largest of the return codes for the executions on the destination agents is used for the return code of the flexible job. However, if at least one of the return codes is -1, the return code of the flexible job is -1.

Table 7–15: Return codes of flexible jobs

| Return code   | Description  | Corresponding message    |
|---|--|--------------------------|
| Return code set by the user application (executable file) | <i>Always normal</i> , <i>Always abnormal</i> , or <i>Judgment by threshold</i> is selected as the end judgment method, and JP1/AJS3 processing terminated normally. | KAVS8138-I<br>KAVS8140-I |
| 0   | <i>Normal if specified file exists</i> or <i>Normal if file was updated</i> is selected as the end judgment method, and the condition for being normal is met.       |                          |
| 1   | <i>Normal if specified file exists</i> or <i>Normal if file was updated</i> is selected as the end judgment method, and the condition for being normal is not met.   |                          |
| -1  | The monitoring termination time was exceeded.  | KAVS8101-E               |
|   | Resources are insufficient.  | KAVS8102-E               |
|   | Communication with the JP1/AJS3 Autonomous Agent Messenger service is impossible.  | KAVS8100-E               |
|   | A system error occurred.   | KAVV8106-E               |
|   | No executable file exists.   | KAVS8108-E               |
|   | The permission level is too low to execute the executable file.  | KAVS8109-E               |

## (2) Return codes of HTTP connection jobs

The following table describes the return codes of HTTP connection jobs.

Table 7–16: List of return codes of HTTP connection jobs

| Return code | Description   | Corresponding message    |
|-------------|---|--------------------------|
| 0           | If return codes are explicitly specified, this return code indicates that processing terminated normally (the HTTP status code specified for 0).<br>If return codes are not explicitly specified, this return code indicates that processing terminated normally (HTTP status code 200).  | KAVS8050-I               |
| 1 to 9      | If return codes are explicitly specified, this return code corresponds to the HTTP status code that is specified for normal termination.  | KAVS8050-I               |
| 10          | Ended normally (an HTTP status code that is not 200 and not defined for any return code).   | KAVS8050-I               |
| 20          | A timeout occurred.   | KAVS8061-E               |
| 30          | The name of the connection destination host could not be resolved.  | KAVS8054-E               |
| 40          | Another error occurred during file access.  | KAVS8053-E               |
| 60          | A parameter error occurred.<br>The job definition items are specified incorrectly (for example, the length or value is invalid).  | KAVS8051-E               |
| 64          | The connection configuration file contains syntax errors.   | KAVS8052-E               |
| 70          | The URL is specified incorrectly.<br>The resource referenced at the URL does not exist.<br>The transmission information file used when the GET method is called contains line breaks.   | KAVS8060-E               |
| 80          | No file exists.<br>A directory was specified.   | KAVS8053-E               |
| 81          | A permission error occurred when a file was accessed.<br>A directory was specified.   | KAVS8053-E               |
| 90          | An attempt to load environment setting parameters failed.   | KAVS8063-E               |
| 101         | Connection to the connection destination host failed.   | KAVS8056-E               |
| 102         | An error occurred during communication with the connection destination host.  | KAVS8057-E               |
| 106         | The size of response data exceeds the maximum file size specified in the connection configuration file.   | KAVS8059-E               |
| 110         | One of the following HTTPS authentication errors occurred: <ul style="list-style-type: none"> <li>• HTTPS authentication failed.</li> <li>• A handshake failed.</li> <li>• The certificate was invalid.</li> <li>• The certificate file could not be accessed.</li> <li>• The CRL file was invalid.</li> <li>• A CRL file access error occurred.</li> <li>• The publisher (CN (Common Name) or SAN (subjectAltName)) could not be checked.</li> <li>• The validation period of the certificate has expired.</li> <li>• The certificate is invalid.</li> </ul> | KAVS8062-E               |
| 120         | A memory shortage occurred.   | KAVS0902-E               |
| 124         | An unexpected error occurred.<br>The libraries necessary for executing the HTTP connection job could not be loaded.   | KAVS8058-E<br>KAVS0990-E |

### **(3) Return codes of event jobs, custom event jobs and action jobs**

For details about the return codes that can be set while an event job, a custom event job or an action job is being executed (from the start to the end of a job process), see *A. Return Values from Event Jobs, Custom Event Jobs and Action Jobs*.

## 7.12 Startup performance of scheduler services

---

A scheduler service manages the number of logs to keep and changes the status according to the status before it stops and the start mode. As a result, the startup performance of scheduler services is mainly influenced by the root jobnets registered for execution, the number of units under jobnets, the number of generations, and the statuses of units when a scheduler service stops. To verify the startup performance of a scheduler service, register a root jobnet for execution according to your operation and wait for the number of generations to reach the number of logs to keep. Then, restart a scheduler service at a peak time when many jobs and jobnets are being executed, and check the time it takes to start the scheduler service.

To confirm that the startup of the scheduler service begins and ends normally, check the scheduler service messages KNAD3705-I and KNAD3602-I, which are output to the integrated trace log.

Note that, if the number of generations has increased since the verification time (for example, because more generations are retained in a jobnet with start conditions), the time required for starting the scheduler service is longer than the verification result.

If the CONDGENWARNINT and MULTIMONWARNNUM environment setting parameters are set, the KAVS1157-W and KAVS1156-W messages are output to inform you that the number of generations in a jobnet with start conditions is increasing. If you want to check the number of generations without having messages output, execute the `ajsshow` command as follows:

```
ajsshow -F scheduler-service-name -g a -i "%JJ %## %CC" -ERT /
```

In addition, you can improve the startup processing of scheduler services by enabling the FLSTARTPERFIMP environment setting parameter. Consider setting this environment setting parameter in addition to using messages and the `ajsshow` command for confirmation. For details about the environment setting parameters, see *20.4.2 Details of the environment setting parameters for scheduler services* in the *JPI/Automatic Job Management System 3 Configuration Guide*. For details about the `ajsshow` command, see *ajsshow* in *3. Commands Used for Normal Operations* in the manual *JPI/Automatic Job Management System 3 Command Reference*.

### Supplementary note

The processing that is performed varies depending on the start mode. With hot start, generation management processing and status change processing are performed. With warm start, schedule recalculation processing is also performed. Furthermore, with disaster recovery start, in addition to all of the previously mentioned processing, startup finishes only after all status change processing has finished. For this reason, check the amount of time required for startup for the start mode that you use.

# 8

## Definition Pre-Check

JP1/AJS3 allows you to check for errors in job definitions.

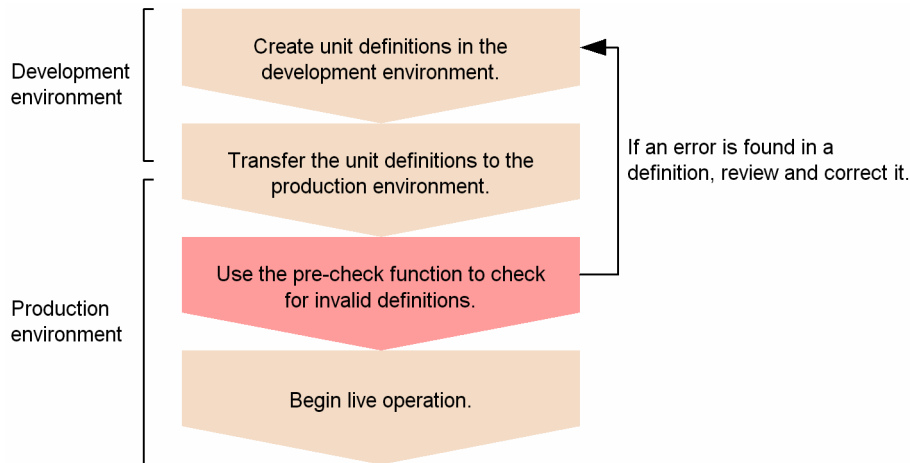
This chapter describes the procedure for the definition pre-check performed before live JP1/AJS3 operation starts. The chapter also describes check items and includes cautionary notes.

## 8.1 Checking definitions before live JP1/AJS3 operation

JP1/AJS3 allows you to check whether job definitions are valid before you deploy them, to prevent failures in a production environment.

The following figure shows the sequence of operations involved in a definition pre-check, up to the point when you begin live operation.

Figure 8–1: Flow of system design and definition pre-check up to commencing operation



Use the `ajschkdef` command to run a definition pre-check. When you execute the `ajschkdef` command, the check results are output to the pre-check result file. You can see the results of the pre-check by viewing the file. The pre-check result file is output to the following location by default:

In Windows, if the installation folder is the default installation folder or is in a folder protected by the system:

```
%ALLUSERSPROFILE%\Hitachi\JP1\JP1_DEFAULT\JP1AJS2\log\ajschkfile.txt
```

The default location of `%ALLUSERSPROFILE%` is `system-drive\ProgramData`.

A folder protected by the system is the path to a folder in any of the following:

- `system-drive\Windows`
- `system-drive\Program Files`
- `system-drive\Program Files (x86)`

In Windows, if the installation folder is other than the above:

```
JP1/AJS3 - Manager-installation-folder\log\ajschkfile.txt
```

In UNIX:

```
/var/opt/jplajs2/log/ajschkfile.txt
```

For details on the `ajschkdef` command, see *ajschkdef* in 3. *Commands Used for Normal Operations* in the manual *JP1/Automatic Job Management System 3 Command Reference*.

Before you run a definition pre-check, you must set up the definition pre-check function. For details on how to set up the function and its environment, see 6.5.1 *Setting up the JP1/AJS3 definition pre-check function* in the *JP1/Automatic Job Management System 3 Configuration Guide* (for Windows systems) or 15.5.1 *Setting up the JP1/AJS3 definition pre-check function* in the *JP1/Automatic Job Management System 3 Configuration Guide* (for UNIX systems).



## 8.1.1 Overview of the definition pre-check function and cautionary notes

This subsection describes the items that are checked by the definition pre-check function, describes the sequence of pre-check operations, and provides cautionary notes on using the function.

### (1) Checked items

The following table lists the items checked by the definition pre-check function.

Table 8–1: Items checked by the definition pre-check function

| Category                        | Description   |  |  |
|---------------------------------|---|--|--|
| Job execution order             | <ul style="list-style-type: none"> <li>• Checks whether the execution order creates a loop.</li> <li>• Checks whether a dependent unit is connected to a judgment job by a condition.</li> <li>• Checks whether relations other than conditions are defined for dependent units.</li> </ul> |  |  |
| Detailed definition of a jobnet | Jobnet connector  | Connection range <sup>#1</sup>   | Checks whether the connection range matches that specified for the connection-destination jobnet.  |
|                                 |   | Connection host <sup>#1</sup>  | Checks whether the specified host is accessible.   |
|                                 |   | Connection service   | Checks whether the specified scheduler service exists.   |
|                                 |   | Connect destination  | Performs the following: <ul style="list-style-type: none"> <li>• Checks whether a connection-destination jobnet is specified.</li> <li>• Checks whether the specified unit is a root jobnet or a planning group.</li> <li>• Checks whether the specified unit exists.</li> <li>• Checks whether the specified unit is subject to execution order control.</li> <li>• Checks whether the specified unit specifies a different host.</li> <li>• Checks whether the specified unit specifies a different scheduler service.</li> <li>• Checks whether the specified unit specifies a different jobnet connector.</li> </ul> |
|                                 | Wait condition  | Checks the following: <ul style="list-style-type: none"> <li>• Whether the specified units whose ends are being waited for exist.</li> <li>• Whether the unit whose end is being waited for is a type that can be specified as a unit whose end is being waited for<sup>#2</sup>.</li> </ul> |  |
| Execution agent restriction     | Checks whether the execution agent is permitted on the manager side. <sup>#3, #4, #5</sup>  |  |  |
| Empty job definition            | For a Unix job, checks whether a <b>Script file name</b> or <b>Command statement</b> is specified.<br>For a PC job, checks whether a <b>File name</b> is specified.<br>For a Receive JP1 event job, checks whether a <b>Event ID</b> is specified.  |  |  |
| Execution agent name            | <ul style="list-style-type: none"> <li>• Unix job</li> <li>• PC job</li> <li>• Action job</li> </ul>  | When <b>Standard</b> is specified as the <b>Exec. service</b> <sup>#6</sup>  | Performs the following: <ul style="list-style-type: none"> <li>• Checks whether the agent host is registered with the manager.</li> </ul>  |

| Category                     | Description  |   |  |
|------------------------------|--|---|--|
| Execution agent name         | <ul style="list-style-type: none"> <li>• Unix job</li> <li>• PC job</li> <li>• Action job</li> </ul>   | When <b>Standard</b> is specified as the <b>Exec. service</b> <sup>#6</sup> | <ul style="list-style-type: none"> <li>• Checks whether the manager can resolve the execution host name (agent host name) set for the execution agent.</li> <li>• Checks whether the agent can resolve the host name of the manager.<sup>#7</sup></li> </ul>   |
|                              |  | When <b>Queueless Agent</b> is specified as the <b>Exec. service</b>        | Performs the following: <ul style="list-style-type: none"> <li>• Checks whether the manager can resolve the host name of the agent.</li> <li>• Checks whether the agent can resolve the host name of the manager.<sup>#7</sup></li> </ul>  |
|                              | <ul style="list-style-type: none"> <li>• Event job</li> <li>• Custom event job</li> </ul>  |   | Performs the following: <ul style="list-style-type: none"> <li>• Checks whether the agent host is registered with the manager.</li> <li>• Checks whether the manager can resolve the execution host name (agent host name) set for the execution agent.</li> <li>• Checks whether the agent can resolve the host name of the manager.<sup>#7</sup></li> </ul>      |
|                              | <ul style="list-style-type: none"> <li>• Jobnet<sup>#6</sup></li> <li>• Custom job<sup>#6</sup></li> <li>• Flexible job<sup>#5, #6</sup></li> <li>• HTTP connection job<sup>#6</sup></li> </ul>  |   | Performs the following: <ul style="list-style-type: none"> <li>• Checks whether the execution agent is registered with the manager.</li> <li>• Checks whether the manager can resolve the execution host name (agent host name) set for the execution agent.</li> <li>• Checks whether the agent can resolve the host name of the manager.<sup>#7</sup></li> </ul> |
| User mapping                 | Checks whether user mapping can be performed correctly on the agent. This check uses the same method for JP1 users, execution source hosts, and OS users as when normal jobs are executed. Note that JP1/AJS3 does not check whether the OS user mapped to a JP1 user actually exists.<br>Note that if you use the execution-user fixing function, the definition pre-check checks the user mapping of the fixed JP1 user according to the contents of the unit-attribute profile. <sup>#3, #5</sup> |   |  |
| Detailed definition of a job | Unix job   | Script file name <sup>#9</sup>  | Checks whether the file exists on the Agent.   |
|                              |  | Environment file  | 1. Checks whether the file exists on the Agent.<br>2. Checks the access permission for the file.   |
|                              |  | Working path  | 1. Checks whether the path exists on the Agent.<br>2. Checks the access permission for the path.   |
|                              |  | Standard input  | 1. Checks whether the file exists on the Agent.  |

| Category                     | Description           |                                |   |
|------------------------------|-----------------------|--------------------------------|---|
| Detailed definition of a job | Unix job              | Standard input                 | 2. Checks the access permission for the file.   |
|                              |                       | Standard output                | 1. Checks whether the directory containing the file exists on the Agent.<br>2. If the file exists, checks the access permission for the file. |
|                              |                       | Standard error                 | 1. Checks whether the directory containing the file exists on the Agent.<br>2. If the file exists, checks the access permission for the file. |
|                              |                       | User name                      | Checks the user by user mapping.  |
|                              |                       | File to transfer <sup>#9</sup> | 1. Checks whether the file exists on the Manager.<br>2. Checks the access permission for the file.  |
|                              |                       | Destination file <sup>#9</sup> | 1. Checks whether the file exists on the Agent.<br>2. Checks the access permission for the file.  |
|                              | PC job                | File name <sup>#9</sup>        | Checks whether the file exists on the Agent.  |
|                              |                       | Environment file               | Checks whether the file exists on the Agent.  |
|                              |                       | Working path                   | Checks whether the path exists on the Agent.  |
|                              |                       | Standard input                 | Checks whether the file exists on the Agent.  |
|                              |                       | Standard output                | Checks whether the directory containing the file exists on the Agent.   |
|                              |                       | Standard error                 | Checks whether the directory containing the file exists on the Agent.   |
|                              |                       | User name                      | Checks the user by user mapping.  |
|                              |                       | File to transfer <sup>#9</sup> | Checks whether the file exists on the Manager.  |
|                              |                       | Destination file <sup>#9</sup> | Checks whether the file exists on the Agent.  |
|                              | Receive JP1 Event job | Event ID                       | Checks whether the event ID confirms to the proper format.  |
|                              |                       | Find events prior to execution | Checks whether the pre-search time is within the specified range.   |
|                              | Monitoring File job   | Monitoring file                | Checks the format of the file name.   |
|                              | Receive Mail job      | Platform                       | Checks whether the OS matches that of the execution destination host.   |

| Category   | Description   |          |   |
|--|---|----------|---|
| Detailed definition of a job                           | Common to action jobs   | Platform | Checks whether the OS matches that of the execution destination host. <sup>#8</sup> |
| Permission for executing files <sup>#9, #10, #11</sup> | <p>Checks whether the OS user who executes the job has execution and read permission for the target file. This check uses the same method as when normal jobs are executed. However, for queueless jobs executed in UNIX, the execution permission is not checked when the OS user executing a job has superuser permission (that is, the root user).</p> <p>Note that JP1/AJS3 checks execution-file permissions only for the primary group of the OS user who executes the job.</p> |          |   |

#1

When **Connection range** is set to **Other service** and the connection source restriction function is enabled on the destination host, you must configure the manager connection permission configuration file to include the IP addresses of the hosts on which jobnet connectors are defined. If you perform a definition pre-check without configuring the file, the number of checked units shown for NUMBER OF CHECKUNITS in the check results will be reduced by the number of jobnet connectors, and the message KAVS3431-I will be output to the integrated trace log.

#2

If you specify a root jobnet or a unit in the root jobnet as a unit whose end is being waited for, a check is conducted regardless of whether a start condition is used.

If you specify a nested jobnet or job in a planning group as a unit whose end is being waited for, the definition is not checked for errors.

A definition pre-check does not check whether a wait condition produces an execution sequence loop.

#3

Regardless of the profile status (enabled or disabled), the check will be performed based on the profile specified by the `ajschkdef` command.

#4

Whether the execution agent is set in the execution agent profile or not is checked. Therefore, the jobs that are set in the execution agent profile as not subject to restriction checking when executed (`JobExec=off`) will still be checked.

#5

For flexible jobs, replace *agent* with *relay agent*. The destination agent is not checked.

#6

This item is not checked if the name of an execution agent group is specified.

#7

If NAT (network address translation) is used for communication between the agent and the manager, the check in this item results in an error because on the agent host and the manager host, the manager host name is resolved to different IP addresses. You can disregard the error if there are no problems with the configuration.

#8

You cannot specify the platform for the MSMQ message sending job because the MSMQ message sending job cannot be executed in a UNIX environment. Therefore, assume that PC has been specified as the platform, and perform a definition pre-check on the MSMQ message sending job.

#9

This item is not checked if a relative path or a UNC path is specified.

#10

PC jobs (on a Windows host) are not checked.

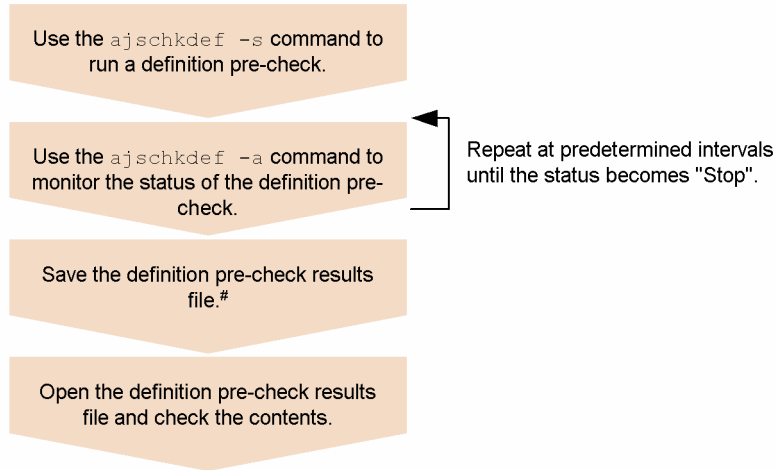
#11

Flexible jobs are not checked.

## (2) Pre-check sequence

The following figure shows the flow of a definition pre-check.

Figure 8–2: Flow of the definition pre-check process



#: Because the file may be overwritten by a later request, save the file under a different name.

## (3) Cautionary notes

Note the following when performing a definition pre-check:

1. Do not run a definition pre-check while the system is being used to perform business tasks. Doing so can lead to the following problems:
  - Access contention for the scheduler database reduces job processing performance.
  - The system load might temporarily spike, causing errors during job execution processing.
2. To run a definition pre-check, the JP1/AJS3 service must be started on the manager.
3. You cannot perform multiple definition pre-checks simultaneously.
4. If you specify a file name and parameters in the **Script file name** field for a Unix job or in the **File name** field for a PC job, the definition pre-check detects an error.
5. Information passed from events is not checked.
6. Macro variables are not checked.
7. Environment variables are not checked.
8. The job definition items that is specified with Variables in the `$variable-name$` format are not checked. If `$` is used twice or more in the definition precheck, the system assumes that variables are used in the definition pre-check.
9. In UNIX, JP1/AJS3 Check Manager service does not use the port number for the `jp1ajs2chkman` service specified in the `services` file.
10. JP1/AJS3 checks the access permissions for a script file in a different way from that used during actual operation. Therefore, the definition pre-check may detect an error for a job that would end normally.
11. Messages KAVS3400 to KAVS3431 are output only to the integrated trace log or the standard output.

12. In Windows, when you specify a folder as an environment variable file, a standard input file, a standard output file, or a standard error output file, and execute the queueless agent service, the job ends normally but the pre-check detects an error.
13. In UNIX, if you specify a directory as an environment variable file and execute the job by using the queueless agent service, the job ends normally but the pre-check detects an error.
14. When you specify the `-A` option in the `ajschkdef` command, information about the OS user who executes the job is required in order to check the permissions for the executable file. Therefore, user mapping must also be checked. However, in a Windows system, the OS does not check the account details of the user who is specified in **User name** in the Define Details - [PC Job] dialog box.
15. When both of the following conditions exist, temporary files are created before the definition pre-check:
  - In the definition of a Unix job, a non-existent file is specified for the **Standard output**, **Standard error**, or **Destination file**.
  - A definition pre-check is executed for this job with the `-D` option specified.The temporary files are created in the location specified for the **Standard output**, **Standard error**, or **Destination file** of jobs that meet these conditions. As a result, the update time for the parent directory of the specified file might change. The temporary files are deleted when the check is finished.
16. The following units and their lower-level units are not checked:
  - Remote Jobnet
  - Manager jobnet
  - Manager job group
17. If the jobnet release function is used, a pre-check is conducted only on the definition items for which the release status is *Being applied*. Because the definition pre-check cannot be performed on definitions whose release status is *Release wait*, check the jobnet that is the release source.
18. Even when the connection source restriction function is enabled, the definition pre-check function does not check whether the manager host is able to connect to the destination agent host. The definition pre-check function does not detect any error as long as the definition is correct, including when the unit is defined to execute on an agent host that cannot be connected to. However, to perform the definition pre-check for a jobnet connector whose **Connection range** is set to **Other service**, the destination host must permit connection from the host for which the jobnet connector is defined.
19. Do not use an alias for the agent host name of a target jobs. If an alias for the agent host name of a target jobs is used, the pre-check result might not be output correctly.

## (4) Notes on definition pre-checks in a cluster system

Note the following when performing a definition pre-check in a cluster system:

1. In a cluster system, when you perform a definition pre-check for units in a logical host, run the check on the primary node of the logical host. You cannot check unit definitions on the standby node.
2. If a failover occurs while you are checking units on the primary node of the logical host in a cluster system, the checking process is interrupted. Pre-checks are not resumed after a failover to the standby node.

# Appendixes

## A. Return Values from Event Jobs, Custom Event Jobs and Action Jobs

The following table lists the return values from event jobs, custom event jobs and action jobs.

Table A–1: Return values from event jobs and action jobs

| Return value | Description   | Job status after execution or time-out |
|--------------|---|--|
| 0            | The job ended normally, or the system always assumes that the job ends normally.  | Ended normally                         |
| 1            | The job ended normally with a matching message.                                   | Ended normally                         |
| 2            | The job ended normally with detailed matching information.                        | Ended normally                         |
| 3            | The job ended normally without a matching message.                                | Ended normally                         |
| 4            | The job ended normally without detailed matching information.                     | Ended normally                         |
| 5            | Event-inherited information has exceeded the limit.                               | Ended normally                         |
| 6            | An error has occurred during event-inherited information creation.                | Ended normally                         |
| 7            | The job ended normally after a time-out.  | Ended normally                         |
| 12           | The job ended with a warning after a time-out.                                    | Ended with warning                     |
| 16           | Condition is not satisfied (incomplete scheduled task).                           | Bypassed                               |
| 17           | The job was killed after a time-out.  | Killed                                 |
| 18           | The job was killed.   | Killed                                 |
| 20           | The system always assumes that the job ends abnormally.                           | Ended abnormally                       |
| 21           | The job ended abnormally with a matching message.                                 | Ended abnormally                       |
| 22           | The job ended abnormally with detailed matching information.                      | Ended abnormally                       |
| 23           | The job ended abnormally without a matching message.                              | Ended abnormally                       |
| 24           | The job ended abnormally without detailed matching information.                   | Ended abnormally                       |
| 25           | The job ended abnormally after a time-out.  | Ended abnormally                       |
| 30           | The job ended because the event action agent has stopped.                         | Ended abnormally                       |
| 50           | A communication error has occurred between the manager and the agent.             | Ended abnormally or failed to start    |
| 55           | Resources are insufficient. Alternatively, the specified file cannot be accessed. | Ended abnormally                       |
| 56           | Insufficient memory   | Ended abnormally                       |
| 64           | The file cannot be read (I/O error).  | Ended abnormally                       |
| 90           | Incorrect parameter   | Ended abnormally                       |
| 91           | Required item was not defined.  | Ended abnormally or failed to start    |
| 92           | An incorrect macro variable was specified.  | Ended abnormally                       |
| 93           | An incorrect platform was specified.  | Ended abnormally                       |
| 250          | The mail system or message queue system is not available.                         | Failed to start                        |
| 253          | An option error occurred in a prerequisite program.                               | Ended abnormally                       |
| 254          | An error occurred in a prerequisite program.                                      | Ended abnormally                       |
| 255          | System error  | Ended abnormally or failed to start    |



## Note

When an operation (such as killing a jobnet) is performed for all start conditions, or an error occurs for all start conditions, the return value of each event for all start conditions is set to 0. In this case, if an error occurs for a specific event (such as for a definition mistake), the return value is set as shown in the table.

Table A–2: Return values from JP1 event sending jobs

| Return value | Description  |
|--------------|--|
| 0            | The job ended normally.  |
| 1            | Argument error   |
| 100          | A transfer has failed.<br>The arrival of an event was not confirmed. This return value corresponds to return value 3 (transfer failure) for the <code>jevsendd</code> command in JP1/Base.   |
| 120          | Processing continues (if the arrival of an event has not been confirmed within the maximum waiting time).<br>The arrival of an event was not confirmed. This return value corresponds to return value 2 (processing continuing) for the <code>jevsendd</code> command in JP1/Base. |
| 150          | Insufficient memory  |
| 255          | Another error  |

## Note

If a return value other than 0 is returned, determine the cause of the problem from the job execution result messages.

Table A–3: Return values from email sending jobs (in Windows)

| Return value | Description   |
|--------------|---|
| 0            | The job ended normally.   |
| 90           | Parameter error (error in a definition item)  |
| 91           | Parameter error (error in a definition)   |
| 92           | Parameter error (required item undefined)   |
| 94           | Parameter error (excessive number of bytes for a definition)  |
| 95           | A text file, attached file, or list of attached files was not found.                                      |
| 99           | An error indicating that the JP1/AJS3 mail monitoring process or the JP1/AJS3 Mail service does not start |
| 100          | Environment settings reading error  |
| 101          | Environment settings undefined error  |
| 102          | JP1/AJS installation failure  |
| 103          | Environment settings information is corrupted.  |
| 104          | A non-existent profile was specified when the Send Mail job was executed without using Outlook.           |
| 105          | An attempt to access a profile failed when the Send Mail job was executed without using Outlook.          |
| 120          | Communication with the JP1/AJS mail monitoring process or mail monitoring service has failed.             |
| 121          | Sending mail information has failed.  |
| 150          | Insufficient memory   |
| 151          | Insufficient resources (for files)  |

| Return value | Description   |
|--------------|---|
| 152          | Insufficient resources (for processes)                        |
| 153          | The Send Mail job timed out.                                  |
| 154          | The profile is corrupted.                                     |
| 155          | The desktop heap is insufficient.                             |
| 180          | Acquiring the end status of a mail sending thread has failed. |
| 182          | Sending mail has failed.                                      |
| 255          | System error  |

Table A–4: Return values from email sending jobs (in UNIX)

| Return value | Description  |
|--------------|--|
| 0            | The job ended normally.  |
| 90           | Parameter error (error in a definition item)                             |
| 95           | Text file name is not found.   |
| 96           | No mail address specified error  |
| 97           | Error in a mail address specified  |
| 98           | Reading the text file has failed.  |
| 126          | The email transmission feature is not available.                         |
| 130          | Creating a temporary file to send mail has failed.                       |
| 131          | An incoming signal has interrupted mail sending.                         |
| 150          | Insufficient memory  |
| 254          | The system command used for mail sending has no privilege for execution. |
| 255          | System error   |

Table A–5: Return values from message-queue message sending jobs (in Windows)

| Return value | Description   |
|--------------|---|
| 0            | The job ended normally.   |
| 1            | Connecting to the specified queue has partially failed.                               |
| 3            | A message has been sent to the dead letter queue.                                     |
| 90           | Parameter error (error in a definition item)  |
| 91           | Parameter error (error in a definition)   |
| 92           | Parameter error (required item undefined)   |
| 94           | Parameter error (excessive number of bytes for a definition)                          |
| 95           | The specified file is not found.  |
| 98           | The JP1/AJS2 message queue monitoring process is not available.                       |
| 99           | An error indicating that the JP1/AJS2 message queue monitoring process does not start |
| 100          | Environment settings reading error  |

| Return value | Description   |
|--------------|---|
| 101          | Environment settings undefined error  |
| 102          | JP1/AJS2 installation failure   |
| 103          | The type of the message queuing system used in environment settings is incorrect. |
| 121          | Connecting to the specified queue has failed.                                     |
| 123          | Error in accessing the message queue  |
| 150          | Insufficient resources (for processes)  |
| 151          | Insufficient resources (for files)  |
| 152          | Insufficient resources (for threads)  |
| 153          | Creating a mutex has failed.  |
| 156          | Creating an event has failed.   |
| 157          | Creating a file has failed.   |
| 158          | Insufficient memory   |
| 159          | Sending a message to the dead letter queue has failed.                            |
| 180          | Acquiring the end status of a message sending thread has failed.                  |
| 181          | Acquiring the file size has failed.   |
| 182          | Reading a file has failed.  |
| 183          | Closing a file has failed.  |
| 250          | Another event occurred at startup of the MQSeries service.                        |
| 255          | System error  |

**Table A–6: Return values from message-queue message sending jobs (in UNIX)**

| Return value | Description   |
|--------------|---|
| 0            | The job ended normally.   |
| 2            | Error with warning during message sending   |
| 90           | Parameter error (error in a definition item)                                      |
| 91           | Parameter error (error in a definition)   |
| 95           | The specified file is not found.  |
| 103          | The type of the message queuing system used in environment settings is incorrect. |
| 123          | Error in accessing the message queue  |
| 126          | Message queue transmission processing is not available.                           |
| 150          | Insufficient resources (for processes)  |
| 156          | Creating an event has failed.   |
| 157          | Creating a file has failed.   |
| 158          | Insufficient memory   |
| 255          | System error  |

**Table A–7: Return values from MSMQ message sending jobs**

| Return value | Description   |
|--------------|---|
| 0            | The job ended normally.   |
| 90           | Parameter error (error in a definition item)                    |
| 91           | Parameter error (error in an operand)                           |
| 92           | Parameter error (error in a definition)                         |
| 93           | Parameter error (required item undefined)                       |
| 94           | Message sending error   |
| 95           | Fatal error<br>The MSMQ linkage functionality is not available. |

**Table A–8: Return values from OpenView Status Report job**

| Return value | Description                                   |
|--------------|---|
| 0            | The job ended normally.                       |
| 16           | Parameter error (error in a definition item)  |
| 17           | Parameter error (error in an operand)         |
| 18           | Parameter error (error in a definition)       |
| 19           | Parameter error (required item undefined)     |
| 30           | SNMP service not installed (For Windows only) |
| 31           | SNMP service not started (For Windows only)   |
| 32           | SNMP service community name not specified     |
| 33           | SNMP service trap sending destination not set |
| 34           | Message sending error                         |

**Table A–9: Return values from local power control jobs**

| Return value | Description                            |
|--------------|--|
| 0            | The job ended normally.                |
| Other than 0 | Local power supply control has failed. |

**Table A–10: Return values from remote power control jobs**

| Return value | Description                             |
|--------------|---|
| 0            | The job ended normally.                 |
| Other than 0 | Remote power supply control has failed. |

**Table A–11: Return values from custom event jobs**

| Return value | Description   | Job status after execution or time-out |
|--------------|---|--|
| 0            | The job ended normally.                             | Ended normally                         |
| 5            | Event-inherited information has exceeded the limit. | Ended normally                         |

| Return value            | Description   | Job status after execution or time-out           |
|-------------------------|---|--|
| 7                       | The job ended normally after a time-out.  | Ended normally                                   |
| 12                      | The job ended with a warning after a time-out.  | Ended with warning                               |
| 16                      | Condition is not satisfied (incomplete scheduled task).   | Bypassed   |
| 17                      | The job was killed after a time-out.  | Killed   |
| 18                      | The job was killed.   | Killed   |
| 25                      | The job ended abnormally after a time-out.  | Ended abnormally                                 |
| 30                      | The job ended because the event action agent has stopped.   | Ended abnormally                                 |
| 50                      | A communication error has occurred between the manager and the agent.                               | Ended abnormally or failed to start              |
| 56                      | Insufficient memory   | Ended abnormally                                 |
| 64                      | The file cannot be read (I/O error).  | Ended abnormally                                 |
| 65                      | User mapping failed.  | Failed to start                                  |
| 100 to 199 <sup>#</sup> | These return values are used for the executable program of JP1/AJS3 for Cloud Service Applications. | Ended abnormally or failed to start <sup>#</sup> |
| 250                     | No custom event job can run in this environment.  | Failed to start                                  |
| 255                     | System error  | Ended abnormally or failed to start              |

#

For details about each return value, including the status in which the job is placed after the job runs or times out, see the manual *JP1/Automatic Job Management System 3 for Cloud Service Applications*.

**Table A–12: Return values from action jobs (special)<sup>#1</sup>**

| Return value | Description   |
|--------------|---|
| 128          | The resource is insufficient. <sup>#2</sup>                 |
| -1           | Startup failure, forced end, ended abnormally <sup>#3</sup> |

#1

These return values might be set to values other than the return values of job processes of action jobs. Actions jobs include the following:

- JP1 event sending job
- Email sending job
- Message-queue message sending job
- MSMQ message sending job
- OpenView Status Report job
- Local power control job
- Remote power control job

#2

This return value is used only in Windows systems. For details, see *7.3 Notes on using PC jobs*. Because the OS sets this return value, the return value may change depending on the OS version.

#3

For details, see *7.11.3 Checking the return code of a job*.

## B. Information Passed by Event Jobs and Custom Event Jobs

This appendix describes the information that can be inherited from an event job and a custom event job.

Information received by an event job and a custom event job can be referenced by succeeding jobs or jobnets. To enable referencing, open the Detailed Definition - [*icon-name*] - [Passing Information] dialog box, then set the specific passing information as a macro variable. The information you can set in the macro variable depends on the type of event job and custom event job. For examples of defining a macro variable, see [2.4.4\(6\) Passing information received by an event job and a custom event job](#).

### B.1 Information Passed by Event Jobs

The following table lists the passing information for each type of event job.

Table B–1: Passing information for each type of event job

| Icon name         | Variable                              | Description  | Windows | UNIX | Size (bytes)             |
|-------------------|---------------------------------------|--|---------|------|--------------------------|
| Common            | CMTMOUT#1                             | Whether the job timed out<br>t: Timed out<br>f: Other than timeout   | Y       | Y    | 1                        |
| Receive JP1 event | EVID                                  | Event ID ( <i>basic-code : extended-code</i> #2)<br>A string representing the event ID in <i>basic-code : extended-code</i> format | Y       | Y    | 17                       |
|                   | EVUSR                                 | Source user name   | Y       | Y    | 20                       |
|                   | EVGRP                                 | Source group name  | Y       | Y    | 20                       |
|                   | EVHOST                                | Source event server  | Y       | Y    | 255                      |
|                   | EVIPADDR                              | Source IP address  | Y       | Y    | 39                       |
|                   | EVMSG                                 | Message text   | Y       | Y    | 1,023                    |
|                   | EVDETAIL                              | Detailed event information#3   | Y       | Y    | 1,023                    |
|                   | EVSEV                                 | Extended severity levels   | Y       | Y    | 11                       |
|                   | EV: <i>extended-attribute-name</i> #4 | Optional extended attribute  | Y       | Y    | 4,085                    |
|                   | EVENV1 to EVENV9#5                    | Extracted data   | Y#6     | Y    | 4,085                    |
|                   | EVUSRID                               | Source user ID   | Y       | Y    | 10                       |
|                   | EVGRPID                               | Source group ID  | Y       | Y    | 10                       |
|                   | EVPROCESSID                           | Source process ID  | Y       | Y    | 10                       |
|                   | EVDATE                                | Event date ( <i>yyyy/mm/dd</i> )   | Y       | Y    | 10                       |
| EVTIME            | Event time ( <i>hh:mm:ss</i> )        | Y  | Y       | 8    |                          |
| Monitoring files  | FLFNAME                               | Full path of changed file  | Y       | Y    | Windows:<br>258<br>UNIX: |

| Icon name            | Variable                                | Description  | Windows | UNIX | Size (bytes)                    |
|----------------------|---|--|---------|------|---------------------------------|
| Monitoring files     | FLFNAME                                 | Full path of changed file  | Y       | Y    | 509                             |
|                      | FLCOND                                  | One of the following established monitoring conditions:<br>c (Create)<br>d (Delete)<br>s (Change size)<br>m (Final time write) | Y       | Y    | 1                               |
|                      | FLCTIME                                 | File update time<br>( <i>yyyy/mm/dd . hh:mm:ss</i> )   | Y       | Y    | 19                              |
|                      | FLSIZE                                  | Size of changed file   | Y       | Y    | 20                              |
| Receive mail         | MLRCVADDRESS                            | Sender of received mail <sup>#7</sup>  | Y       | Y    | 256 <sup>#8</sup>               |
|                      | MLRCVSUBJECT                            | Subject of received mail   | Y       | Y    | 256 <sup>#9</sup>               |
|                      | MLRCVBODY                               | Name of file containing received mail text   | Y       | --   | 258                             |
|                      | MLRCVATTACHFILE $nn$ ( $nn$ : 01 to 20) | Name of attached file  | Y       | --   | 258 <sup>#10</sup>              |
|                      | MLRCVATTACHLIST                         | Name of file listing attached files  | Y       | --   | 258 <sup>#11</sup>              |
|                      | MLRCVMAILBODY                           | Name of file containing the body of received mail (1 file set internally)  | --      | Y    | 452                             |
|                      | MLRCVTIME                               | Mail arrival time  | Y       | Y    | 40                              |
| Receive MQ message   | MQRCVCORRELATION                        | Correlation ID of received message   | Y       | Y    | 24                              |
|                      | MQRCVDISCRIMINATION                     | Message ID of received message   | Y       | Y    | 24                              |
|                      | MQRCVQUEUE                              | Name of queue which received the message   | Y       | Y    | 48                              |
|                      | MQRCVMODELQUEUE                         | Model queue name   | Y       | Y    | 48                              |
|                      | MQRCVMESSAGEFILE                        | Name of file in which received message was stored as a message structure (1 file set internally)                               | Y       | Y    | Windows:<br>258<br>UNIX:<br>452 |
| Receive MSMQ message | MSRCVQUEUEPATH                          | Queue path of received message   | Y       | --   | 259                             |
|                      | MSRCVMUTUAL                             | Correlation ID of received message   | Y       | --   | 20                              |
|                      | MSRCVMESSELABEL                         | Message label of received message  | Y       | --   | 249                             |
|                      | MSRCVAPPLICATION                        | Application information about received message (hex numeric)   | Y       | --   | 8                               |
|                      | MSRCVMESSAGEFILE                        | Name of file in which received message was stored as a message structure (1 file set internally)                               | Y       | --   | 258                             |

| Icon name            | Variable | Description                                 | Windows | UNIX | Size (bytes) |
|----------------------|----------|---|---------|------|--------------|
| Log file trapping    | LFFNAME  | Full path of trapped log file               | Y       | Y    | 258          |
|                      | LFDATA   | Trapped data (truncated if over 511 bytes)  | Y       | Y    | 511          |
| Monitoring event log | NELOG    | Log type                                    | Y       | --   | 255          |
|                      | NEEVKIND | Event type                                  | Y       | --   | 20           |
|                      | NESOURCE | Source                                      | Y       | --   | 255          |
|                      | NECLASS  | Class                                       | Y       | --   | 4,085        |
|                      | NEEVID   | Event ID                                    | Y       | --   | 16           |
|                      | NEDETAIL | Explanation (truncated if over 1,023 bytes) | Y       | --   | 1,023        |
| Interval control     | N/A      | N/A   | N/A     | N/A  | N/A          |

Legend:

- Y: Can be specified
- : Cannot be specified
- N/A: Not applicable

Note1

The following characters are replaced by blanks if contained in the passing information:  
 \r, \n, \b, and \f

Note2

Estimate the sizes of the macro variable names and passing information in advance so that their total size does not exceed 4,096 bytes. Be especially careful when start conditions are combined using AND. In this case, the macro variable name and passing information of all event jobs defined in the start conditions are merged. The data merged in this way could easily exceed 4,096 bytes.

#1

If the event job ends normally on detecting an event, *f* is stored in CMTMOUT. If the timeout period is exceeded, *t* is stored in CMTMOUT. If the event job ends abnormally due to an error, nothing is stored in CMTMOUT.

#2

For *extension-code*, 0 is always set.

#3

*Detailed event information* refers to details about the basic attributes of a JP1 event, but is set only when the details are in text format. For details about the JP1 event attributes, see *A.2 JP1 event attributes* in the *JP1/Automatic Job Management System 3 Administration Guide*.

#4

The *extended-attribute-name* is a character string of up to 32 bytes, determined by the JP1/Base event service. For details on extended attribute names, see the *JP1/Base User's Guide*.

#5

The extracted data, if any, is stored in the following sequence in EVENV1 to EVENV9, after regular expressions are checked. Note that this sequence differs from the order in which the parameters are specified in the *ajsdefine* command.



- Source user name
- Source group name
- Source event server
- Message text
- Detailed event information
- Optional extended attribute

Within the extended attribute, regular expressions are checked in the order in which the JP1 events were received, and the extracted data, if any, is stored in order in `EVENV1` to `EVENV9`.

#6

This is set only when extended regular expressions are used.

#7

In Windows, this variable is set to the type of information specified for the `NextAddress` environment setting parameter. If you specify `Address`, the sender's email address is set. If you specify `Nickname`, the sender's display name (nickname) is set.

For the setup to link with a mail system on a Windows host, see *2.3.4 Setting up the environment for the mail system linkage* in the *JP1/Automatic Job Management System 3 Linkage Guide*.

#8

If the OS of the host on which an email reception monitoring job is executed is Windows, an upper limit is placed on the number of characters and the number of bytes. The upper limit on the number of characters is 255.

#9

If the OS of the host on which an email reception monitoring job is executed is Windows, an upper limit is placed on the number of characters and the number of bytes. The upper limit on the number of characters is 253.

#10

Each attached file is saved, and the file name is set to the full path.

Multiple attached files are not saved in any particular order. When 21 or more files are received, only 20 file names can be passed.

#11

The attached files are saved and a list of the files is created. The list file name is set to the full path.

## B.2 Information Passed by Custom Event Jobs

The following table lists the passing information for each type of custom event job.

Table B–2: Passing information for each type of custom event job

| Icon name | Variable                                       | Description  | Windows | UNIX | Size (bytes) |
|-----------|--|--|---------|------|--------------|
| Common    | <code>CMTMOUT</code> <sup>#1</sup>             | Whether the job timed out<br>t: Timed out<br>f: Other than timeout | Y       | Y    | 1            |
|           | <code>OBJECT01</code> to <code>OBJECT10</code> | Pieces of information defined in the linked program <sup>#2</sup>  | Y       | Y    | 4,085        |

Legend:

Y: Can be specified

Note1

The following characters are replaced by blanks if contained in the passing information:

\r, \n, \b, and \f

Note2

Estimate the sizes of the macro variable names and passing information in advance so that their total size does not exceed 4,096 bytes. Be especially careful when start conditions are combined using AND. In this case, the macro variable name and passing information of all custom event jobs defined in the start conditions are merged. The data merged in this way could easily exceed 4,096 bytes.

#1

If the custom event job ends normally on detecting an event,  $f$  is stored in CMTMOUT. If the timeout period is exceeded,  $t$  is stored in CMTMOUT. If the custom event job ends abnormally due to an error, nothing is stored in CMTMOUT.

#2

The information that the user defined in JP1/AJS3 for Cloud Service Applications is stored. For details, see *JP1/Automatic Job Management System 3 for Cloud Service Applications Description, User's Guide and Reference*.

## C. Files Used for HTTP Connection Jobs

This appendix describes the files used for HTTP connection jobs.

### C.1 Connection configuration file

In the *connection configuration file*, you set the HTTP-connection-related information that is required to call a web API.

If you include security information (such as a password) in this file, you do so on your own responsibility.

The following table describes the specifications of the connection configuration file.

Table C–1: Specifications of the connection configuration file

| No. | Definition         | Description  |
|-----|--------------------|--|
| 1   | Permission         | The job execution user requires read permission for the file.  |
| 2   | Character encoding | The file must be written with the same character encoding that is used on the agent host on which HTTP connection jobs are executed. |
| 3   | Maximum size       | 1,024 bytes  |

A model file for the connection configuration file is provided. You can create a connection configuration file by editing a copy of the model file. You can also create a new connection configuration file by following the format shown below.

The location of the model connection configuration file is as follows.

In Windows:

```
installation-folder-for-JP1/AJS3-Manager-or-JP1/AJS3-  
Agent\conf\ajshttpreq_network.conf.model
```

In Linux:

```
/etc/opt/jplajs2/conf/ajshttpreq_network.conf.model
```

After creating the connection configuration file, you can place it at any location on the agent host on which HTTP connection jobs are to be executed.

#### (1) Format

The format of the connection configuration file is as follows:

```
URL=connection-destination-URL  
HttpVer=HTTP-version  
ConnectTimeout=connection-timeout-value  
Timeout=maximum-processing-time-of-the-entire-HTTP-connection-job  
MaximumReceivedDataSize=maximum-size-of-receive-data  
ClientBindAddrIP=IP-address-of-the-client-to-be-bound  
Header=additional-header  
Authorization=whether-to-use-Basic-Authentication-on-the-web-server  
UserName=user-name-used-for-Basic-Authentication-on-the-web-server  
UserPass=password-used-for-Basic-Authentication-on-the-web-server  
Proxy=proxy-server-name  
ProxyAuthorization=whether-to-use-proxy-server-authentication  
ProxyUserName=user-name-used-for-proxy-server-authentication
```

```
ProxyUserPass=password-used-for-proxy-server-authentication
CAFile=certificate-file-name
CrlFile=certificate-revocation-list-file-name
```

Lines beginning with a hash mark (#) are treated as comment lines. Lines on which only single-byte spaces and/or tabs appear before a hash mark (#) are also treated as comment lines.

## (2) Setting items

The entries in the connection configuration file are described below. In the file, specify each entry on a separate line.

### (a) URL=<connection-destination-URL>

Specify the connection destination URL, using a character string that is 2,083 or fewer bytes long. The URL must begin with `http://` or `https://`. You can use only single-byte characters.

When specifying the connection-destination-URL, you cannot include URL parameters. To include URL parameters in the connection-destination-URL, specify the URL parameters in the transmission information file.

To use a port number that is not the default port number, include the port number in the URL. The default port number is as follows:

- For `http`: 80
- For `https`: 443

In the case of `https`, you must specify `CAFile=name-of-the-certificate-file`.

This entry cannot be omitted.

### (b) HttpVer=<HTTP-version>

Specify either of the following values as the HTTP version:

1.0

Uses HTTP 1.0 to send a request.

1.1

Uses HTTP 1.1 to send a request.

This entry can be omitted. If omitted, 1.1 is assumed.

### (c) ConnectTimeout=<connection-timeout-value>

Specify in seconds the maximum wait time before a connection to the destination host can be established. You can specify a value from 1 to 600.

If a connection to the destination host is not established within the specified time, the HTTP connection job terminates abnormally.

This entry can be omitted. If omitted, 10 is assumed.

#### Cautionary note

Depending on the OS settings on the host on which the HTTP connection job is executed and the network settings in the system, a timeout might occur before the specified time expires, causing the job to terminate abnormally. For details about the OS settings, see the documentation for the OS.

### **(d) Timeout=<maximum-processing-time-of-the-entire-HTTP-connection-job>**

Specify in seconds the maximum processing time of the entire HTTP connection job. You can specify a value from 1 to 86,400.

If the processing does not finish within the specified time, the HTTP connection job terminates abnormally.

This entry can be omitted. If omitted, 600 is assumed.

### **(e) MaximumReceivedDataSize=<maximum-size-of-receive-data>**

Specify in megabytes the maximum size of data that can be received from a web API. You can specify a value from 1 to 1,024.

The value specified for this entry is compared with the value of the `Content-Length` HTTP header of the received data. If the `Content-Length` value is larger, data reception stops and the HTTP connection job terminates abnormally.

If the data being received does not contain the `Content-Length` HTTP header, the size of the received data is checked in real time. When the size of received data in the HTTP body reaches the specified value, data reception stops and the HTTP connection job terminates abnormally.

This entry can be omitted. If omitted, 500 is assumed.

### **(f) ClientBindAddrIP=<IP-address-of-the-client-to-be-bound>**

To bind a client, specify the IP address of the client by using a character string that is 39 or fewer bytes long.

You can specify either an IPv4 address or an IPv6 address.

This entry can be omitted. If omitted, no client is bound.

#### Cautionary note

You must bind a client if you want to use either a specific IP address for passing through the firewall or a specific LAN.

### **(g) Header=<additional-header>**

To add a header to the HTTP header in a request, specify the header by using a character string that is 1,024 or fewer bytes long.

You can specify this entry on multiple lines to add multiple headers. If the settings of additional headers specified in `Header=additional-header` entries are the same as the header settings specified by the HTTP connection job, the settings specified by the `Header=additional-header` entries take precedence.

You can specify each `Header=additional-header` entry in one of the following formats:

`Header=header-name:header-value`

To add a custom header that has a value, specify the header name and value with an intervening colon (:). For example, to add a header whose name is ABC and whose value is DEF, specify `Header=ABC:DEF`.

`Header=header-name:`

To disable a header that is set by the HTTP connection job, specify the header name followed by a colon (:). For example, to disable the ABC header, specify `Header=ABC:.`

`Header=header-name;`

To add a custom header that does not have a value, specify the header name followed by a semicolon (;). For example, to add the ABC header that has no value, specify `Header=ABC;.`

This entry can be omitted. If omitted, no header is added.

#### Cautionary note

If you disable headers that are set by the HTTP connection job, the job might not be executed normally. Before you disable such headers, fully consider the impact.

### **(h) Authorization=<whether-to-use-Basic-Authentication-on-the-web-server>**

Specify either of the following values to determine whether to use Basic Authentication on the web server:

y

Uses Basic Authentication on the web server.

n

Does not use Basic Authentication on the web server.

If you specify y, the `Authorization` header is specified as an HTTP header in the request based on the values specified for the following entries: `UserName=user-name-used-for-Basic-Authentication-on-the-web-server` and `UserPass=password-used-for-Basic-Authentication-on-the-web-server`. Note that if a `Header=additional-header` entry is used to specify another `Authorization` header, the `Authorization` header specified as an additional header takes effect.

To use an authentication method other than Basic Authentication, you must use a `Header=additional-header` entry to specify the method.

If y is specified when the connection destination host does not require authentication on a web server, how the system behaves depends on the specifications of the web server for the connection destination.

This entry can be omitted. If omitted, n is assumed.

### **(i) UserName=<user-name-used-for-Basic-Authentication-on-the-web-server>**

If you choose to use Basic Authentication on a web server, specify the user name that is used for authentication by using a character string that is 256 or fewer bytes long. The user name you specify must not include colons (:).

To use Basic Authentication on a web server, this entry cannot be omitted.

### **(j) UserPass=<password-used-for-Basic-Authentication-on-the-web-server>**

If you choose to use Basic Authentication on a web server, specify the password that is used for authentication by using a character string that is 256 or fewer bytes long.

To use Basic Authentication on a web server, this entry cannot be omitted.

### **(k) Proxy=<proxy-server-name>**

To use a proxy server, specify the name of the proxy server by using a character string that is 2,083 or fewer bytes long.

To use Basic Authentication on a proxy server, this entry cannot be omitted.

### **(l) ProxyAuthorization=<whether-to-use-proxy-server-authentication>**

To use a proxy server, specify either of the following values to determine whether to use Basic Authentication on a proxy server:

y

Uses Basic Authentication on a proxy server.

n

Does not use Basic Authentication on a proxy server.

If you specify y, the ProxyAuthorization header is specified as an HTTP header in the request based on the values specified for the following entries: ProxyUserName=*user-name-used-for-proxy-server-authentication* and ProxyUserPass=*password-used-for-proxy-server-authentication*. Note that if a Header=*additional-header* entry is used to specify another ProxyAuthorization header, the ProxyAuthorization header specified as an additional header takes effect.

To use an authentication method other than Basic Authentication, you must use a Header=*additional-header* entry to specify the method.

If y is specified when the connection destination host does not require authentication on a proxy server, how the system behaves depends on the specifications of the proxy server for the connection destination.

This entry can be omitted. If omitted, n is assumed.

### **(m) ProxyUserName=<user-name-used-for-proxy-server-authentication>**

If you use Basic Authentication on a proxy server, specify the user name that is used for authentication by using a character string that is 256 or fewer bytes long. The user name you specify must not include colons (:).

To use Basic Authentication on a proxy server, this entry cannot be omitted.

### **(n) ProxyUserPass=<password-used-for-proxy-server-authentication>**

If you choose to use Basic Authentication on a proxy server, specify the password that is used for authentication by using a character string that is 256 or fewer bytes long.

To use Basic Authentication on a proxy server, this entry cannot be omitted.

### **(o) CAFile=<certificate-file-name>**

If you specified a character string starting with `https://` in `URL=connection-destination-URL`, specify the file name of the root certificate by using a character string that is 511 or fewer bytes long. The root certificate must be in PEM format. If you specify a root certificate in DER format, the KAVS8062-E message is output and the HTTP connection job terminates abnormally.

If you specified a character string starting with `http://` in `URL=connection-destination-URL`, this entry can be omitted.

### **(p) CrlFile=<certificate-revocation-list-file-name>**

To verify the server certificate, specify the name of the certificate revocation list file by using a character string that is 511 or fewer bytes long. The certificate revocation list file must be in PEM format. If you specify a file in DER format, the KAVS8062-E message is output and the HTTP connection job terminates abnormally.

This entry can be omitted. If omitted, the check of the certificate revocation status will not be performed.

## **(3) Format of the model file**

The format of the model file of the connection configuration file is as follows:

```
URL=
#HttpVer=1.1
#ConnectTimeout=10
#Timeout=600
#MaximumReceivedDataSize=500
#ClientBindAddrIP=
#Header=
#Authorization=n
#UserName=
#UserPass=
#Proxy=
#ProxyAuthorization=n
#ProxyUserName=
#ProxyUserPass=
#CAFile=
#CrlFile=
```

Omissible entries are commented out. To specify those entries explicitly with desired values, delete the hash marks (#) and specify the desired values.

## (4) Cautionary notes

- The following lines are ignored:
  - Blank lines
  - Lines containing only single-byte spaces or tabs
- Single-byte spaces or tabs at the beginning and end of an entry are deleted.
- The only entry that can be specified on multiple lines is `Header=additional-header`. If an entry other than `Header=additional-header` is specified on multiple lines, only the last specified one takes effect.
- If the connection configuration file contains entries other than the entries indicated in this appendix, the HTTP connection job terminates abnormally.
- If you specify only an entry name without specifying a value, the HTTP connection job terminates abnormally.

## C.2 Transmission information file

In a *transmission information file*, you set information about a request to a web API. The file format and items that can be specified in the file differ depending on the web API to be called. If there is no information to be sent when a request is sent, you do not need to create the transmission information file.

If you include security information (such as a password) in this file, you do so on your own responsibility.

This file is not used if extended mode is used when the versions of JP1/AJS3 - Manager and JP1/AJS3 - Agent are 12-50 or later. For details about the extended mode, see [2.4.12 Linking with a business system on the web \(example of defining a jobnet that uses HTTP connection jobs\)](#).

The following table describes the specifications of the transmission information file.



Table C–2: Specifications of the transmission information file

| No. | Definition         | Description   |
|-----|--------------------|---|
| 1   | Permission         | The job execution user requires read permission for the file.   |
| 2   | Character encoding | The character encoding to specify depends on the value specified for the <code>charset</code> parameter of the <code>Content-Type</code> header that is specified for the Header item in the connection configuration file. |
| 3   | Maximum size       | When the GET request is sent : 8 megabytes<br>When one of the POST request, PUT request, and DELETE request is sent : 10 megabytes  |

Place the transmission information file at any location on the agent host on which the HTTP connection job is executed.

#### Supplementary notes

- When the GET request is sent, the content of the transmission information file is recognized as a URL parameter.
- When one of the POST request, PUT request, and DELETE request is sent, the content of the transmission information file is recognized as the message body. Note that you can specify URL parameters in the message body, but you cannot specify request information other the URL parameters at the same time.
- To specify URL parameters, in the transmission information file, specify the parameters in the following format without using newline characters. Note that you do not need to add a question mark (?) at the beginning of a line.  
*parameter-name=parameter-value [ &parameter-name=parameter-value&parameter-name= . . . ]*
- If the request information contains a value that requires URL encoding, in the transmission information file, you must specify the value in the URL-encoded format.

### C.3 Transmission information file (URL parameter)

In the transmission information file (URL parameter), set the URL parameter of the request information for the Web API. The file format and items that can be specified in the file differ depending on the web API to be called. If there is no information to be sent when a request is sent, you do not need to create the transmission information file (URL parameter).

Create this file if extended mode is used when the versions of JP1/AJS3 - Manager and JP1/AJS3 - Agent are 12-50 or later. For details about the extended mode, see [2.4.12 Linking with a business system on the web \(example of defining a jobnet that uses HTTP connection jobs\)](#).

If you include security information (such as a password) in this file, you do so on your own responsibility.

The following table describes the specifications of the transmission information file (URL parameter).

Table C–3: Specifications of the transmission information file (URL parameter)

| No. | Definition         | Description   |
|-----|--------------------|---|
| 1   | Permission         | The job execution user requires read permission for the file.   |
| 2   | Character encoding | The character encoding to specify depends on the value specified for the <code>charset</code> parameter of the <code>Content-Type</code> header that is specified for the Header item in the connection configuration file. |
| 3   | Maximum size       | 8 megabytes   |

Place the transmission information file (URL parameter) at any location on the agent host on which the HTTP connection job is executed.

## Supplementary notes

- Specify the URL parameter in the following format without using a line break character. Note that you do not need to add a question mark (?) at the beginning of a line.

*parameter-name=parameter-value [ &parameter-name=parameter-value&parameter-name= . . . ]*

- If the request information contains a value that requires URL encoding, in the transmission information file (URL parameter), you must specify the value in the URL-encoded format.

## C.4 Transmission information file (Message body)

In the transmission information file (Message body), set the message body of the request information for the Web API. The file format and items that can be specified in the file differ depending on the web API to be called. If there is no information to be sent when a request is sent, you do not need to create the transmission information file (Message body).

Create this file if extended mode is used when the versions of JP1/AJS3 - Manager and JP1/AJS3 - Agent are 12-50 or later. For details about the extended mode, see [2.4.12 Linking with a business system on the web \(example of defining a jobnet that uses HTTP connection jobs\)](#).

If you include security information (such as a password) in this file, you do so on your own responsibility.

The following table describes the specifications of the transmission information file (Message body).

Table C–4: Specifications of the transmission information file (Message body)

| No. | Definition         | Description   |
|-----|--------------------|---|
| 1   | Permission         | The job execution user requires read permission for the file.   |
| 2   | Character encoding | The character encoding to specify depends on the value specified for the <code>charset</code> parameter of the <code>Content-Type</code> header that is specified for the Header item in the connection configuration file. |
| 3   | Maximum size       | 10 megabytes  |

Place the transmission information file (Message body) at any location on the agent host on which the HTTP connection job is executed.

### Supplementary note

If the request information contains a value that requires URL encoding, in the transmission information file (Message body), you must specify the value in the URL-encoded format.

## C.5 Status file

The HTTP status code of the data received from the web API is written to be the *status file*. This file is automatically created by the HTTP connection job if the HTTP status code is 1 or more.

The following table describes the specifications of the status file.

Table C–5: Specifications of the status file

| No. | Definition         | Description   |
|-----|--------------------|---|
| 1   | Permission         | The access permission settings of the folder in which to create the status file must permit the user who executes the job to create files.<br>For the created file, the following permission settings are specified:<br>In Windows:<br>Owner: User who executed the job<br>In Linux:<br>Permission: 644<br>Owner: User who executed the job<br>Group: Group that the user who executed the job belongs to |
| 2   | Character encoding | ASCII characters  |
| 3   | Maximum size       | 100 bytes   |

## (1) Format

The format of the status file is as follows:

```
HTTP-Status-Code=HTTP-status-value
```

## C.6 Received header file

The HTTP headers of the data received from a web API are written to the *received header file*. This file is automatically created by the HTTP connection job. The format in which data is written differs depending on the specifications of the web API that was called.

If you include security information (such as a password) in this file, you do so on your own responsibility.

The following table describes the specifications of the received header file.

Table C–6: Specifications of the received header file

| No. | Definition         | Description  |
|-----|--------------------|--|
| 1   | Permission         | The access permission settings of the folder in which to create the received header file must permit the user who executes the job to create files.<br>For the created file, the following permission settings are specified:<br>In Windows:<br>Owner: User who executed the job<br>In Linux:<br>Permission: 644<br>Owner: User who executed the job<br>Group: Group that the user who executed the job belongs to |
| 2   | Character encoding | Depends on the specifications of the web API to be called.   |
| 3   | Maximum size       | Not limited  |

For the receive data storage method in the definition items for HTTP connection jobs, if you choose to output the header and body to the same file, the body of the received data will also be written to the received header file.

## C.7 Received body file

The HTTP body of data received from a web API is written to the *received body file*. This file is automatically created by the HTTP connection job. The format in which data is written differs depending on the specifications of the web API that was called.

If you include security information (such as a password) in this file, you do so on your own responsibility.

The following table describes the specifications of the received body file.

Table C–7: Specifications of the received body file

| No. | Definition         | Description  |
|-----|--------------------|--|
| 1   | Permission         | The access permission settings of the folder in which to create the received header file must permit the user who executes the job to create files.<br>For the created file, the following permission settings are specified:<br>In Windows:<br>Owner: User who executed the job<br>In Linux:<br>Permission: 644<br>Owner: User who executed the job<br>Group: Group that the user who executed the job belongs to |
| 2   | Character encoding | Depends on the specifications of the web API to be called.   |
| 3   | Maximum size       | Specified by using the <code>MaximumReceivedDataSize=maximum-size-of-receive-data</code> entry in the connection configuration file.   |

The received body file is created if you choose to output the header and body to different files for the receive data storage method in the definition items for HTTP connection jobs. If you choose to output the header and body to the same file, the body of received data is appended to the received header file. In this case, the received body file is not created.

### Supplementary notes

- For HTTP connection jobs, whether the HTTP body is included in the received data is determined by the HTTP status code.  
If the HTTP status code is 100 to 199, 204, or 304, the HTTP connection job does not create the received body file, assuming that the HTTP body is not included in the received data.  
If the HTTP status code is other than the above codes, the HTTP connection job determines whether the HTTP body is included based on RFC2616. If the job determines that the HTTP body is not included, the received body file is not created.
- If `Y` is specified for the `TRANSFER_DECODING` environment setting parameter, data received in chunk format is decoded when it is output. For details about the `TRANSFER_DECODING` environment setting parameter, see *20.14.2(1) TRANSFER\_DECODING* in the *JP1/Automatic Job Management System 3 Configuration Guide*.

## D. Version Revisions

---

For details on the version revisions, see *B. Version Revisions* in the manual *JP1/Automatic Job Management System 3 Overview*.

## E. Reference Material for This Manual

---

For details on reference information that would be helpful in reading this manual, see *C. Reference Material for This Manual* in the manual *JP1/Automatic Job Management System 3 Overview*.

## F. Glossary

---

For the glossary, see *D. Glossary* in the manual *JP1/Automatic Job Management System 3 Overview*.

# Index

## A

- access permissions
  - setting 232
  - when defining and executing jobnets 233
  - when executing and operating QUEUE and submit jobs 235
  - when working with agent management information 236
- action job
  - return values 344
- automatic retry 178
- automatic retry in event of abnormal termination of job 178

## B

- broadcast agent 172
- broadcast execution 171

## C

- calendar
  - defining 195
- calendar and schedule planning
  - flow 194
- calendar work task
  - example 215
- connection configuration file 355
- custom event job 136
  - timeout 138
  - Types of events that can be monitored by a custom event job 138
- custom event jobs
  - return value 344

## D

- defining cycle jobs 207
- dependent unit
  - rerunning 131
- during retry 180

## E

- event arrival confirmation 153
- event information, passing 139
- event job 136
  - find events prior to execution 139

- monitoring by event job 269
- return values 344
- timeout 138
- types of events that can be monitored 137
- executable file
  - designing 21
- execution schedule
  - defining 196
- execution user 245
- execution-user fixing function 246

## F

- Find events prior to execution 139
- flexible job 170, 171

## G

- global macro variable 160

## H

- HTTP connection job 173
- HTTP connection jobs
  - files used for 355

## I

- interval control job 144
- Interval Control job
  - notes 297

## J

- job
  - defining 21
  - definition considerations 26
  - execution order considerations 30
- job definition
  - considerations 20
- job definitions
  - checking before JP1/AJS3 operation 336
- job execution
  - recovering from failure 38
- job execution order
  - considerations 20
- jobnet
  - definition considerations 30, 39



- definition examples 128
- execution registration methods 219
- jobnet connector 156
  - using 40
- job types
  - choosing 23
- JP1\_AJS\_Admin 233
- JP1\_AJS\_Editor 233
- JP1\_AJS\_Guest 233
- JP1\_AJS\_Manager 233
- JP1\_AJS\_Operator 233
- JP1\_JPQ\_Admin 235, 237
- JP1\_JPQ\_Operator 235, 237
- JP1\_JPQ\_User 235, 237
- JP1/AJS3 design
  - steps 16
- JP1/Base event log trapping 143
- JP1/Base log file trapping 141
- JP1 permission level 232
- JP1 resource group 232
- JP1 resource groups to be defined 232
- JP1 user
  - registering 230
- judgment job 130
  - based on presence/absence of file 131
  - usage example 130
  - working with return codes 130

## L

- load
  - processing with distributed load 119
- Local Power Control job
  - notes 308

## M

- macro variable 147
- mapping
  - user 254
- monitored events 275
- monitoring
  - by event job 269
  - events and messages issued by JP1/AJS3 301
  - target 269
- monitoring event log job 143
- Monitoring Event Log job
  - notes 295

- monitoring files job 140
  - monitoring options 140
- Monitoring Files job
  - conditions for passing monitoring status 286
  - detection of monitoring target files by file extension 291
  - events monitored by 275
  - in jobnet with monitoring target file name specified 283
  - in jobnet with monitoring target file name specified by wildcard 283
  - in start condition with monitoring target file name specified 284
  - in start condition with monitoring target file name specified by wildcard 285
  - notes 275
  - notes on defining 287
  - options 280
  - specifying file names 279
  - status passing option 286
  - using wildcards in file names 279
  - when start monitoring job option is enabled 280
- monitoring log files job 141
- Monitoring Log Files job
  - examples when acquired messages are lost and not lost with SEQ2 selected 294
  - notes 293

## N

- number of logs to keep 260
  - effects on performance 260
- number of retry executions 181

## O

- OpenView Status Report job
  - notes 307
- OR job 128
- overview of Monitoring log files job operation 142
- owner permission 238

## P

- passing information
  - notes on defining 298, 318
- passing information by custom event job type 353
- passing information by event job type 350
- passing information received by event job and a custom event job 147

passing information setting job 160  
primary authentication server 229

## R

received body file 364  
received header file 363  
receive JP1 event job 139  
Receive JP1 Event job  
    notes 271  
    notes on defining 274  
Receive Mail job  
    notes 293  
recovery job 154  
recovery jobnet 154  
recovery unit  
    precautions when using 267  
registered user  
    setting 230  
regular expression 274  
Remote Power Control job  
    notes 308  
retry execution 178  
retry information 180  
retry registration time 181  
retry setting 179  
retry start time 182  
retry status 180  
return code  
    checking the return code of job 329  
    set by JP1/AJS3 329  
return value 344  
Return values from custom event jobs 348  
return values from email sending jobs (in UNIX) 346  
return values from email sending jobs (in Windows) 345  
return values from event jobs and action jobs 344  
return values from JP1 event sending jobs 345  
return values from local power control jobs 348  
return values from message-queue message sending jobs (in UNIX) 347  
return values from message-queue message sending jobs (in Windows) 346  
return values from MSMQ message sending jobs 348  
return values from OpenView Status Report job 348  
return values from remote power control jobs 348  
reviewing schedules after changing to 48-hour schedule 200  
root jobnet

controlling execution order 40  
notes on defining thousands of root jobnets in one hierarchy level 259  
notes on number of root jobnets registered for execution 259

## S

schedule  
    considerations 196  
schedule by days from start 208  
secondary authentication server 229  
send JP1 event job 152  
Send JP1 Event job  
    notes 306  
Send Mail job  
    notes 307  
standard error output file  
    notes 322  
standard job 128  
standard output file  
    notes 322  
start condition  
    considerations 197  
start monitoring option 280  
status file 362  
status passing option  
    example of operation 287

## T

target  
    monitoring 269  
time monitoring 144  
transmission information file 360  
transmission information file (Message body) 362  
transmission information file (URL parameter) 361

## U

unit-attribute profile 241, 249  
unit whose end is being waited for 60  
unit with wait conditions 60  
upper-level unit-attribute inheritance function 239  
user  
    mapping 254  
user mapping 254  
    overview of processing execution 254

## W

wait condition 59, 157

wait status 62

wildcard characters 279

work task

    automating 17

work task access permissions 228

    flow when considering 228

work task automation

    flow 18

    key questions 19

    tips on 119

    useful JP1/AJS3 functions 125

work task design

    flow 16

    overview 15

---

 **Hitachi, Ltd.**

6-6, Marunouchi 1-chome, Chiyoda-ku, Tokyo, 100-8280 Japan

---