

**Job Management Partner 1 Version 10**

**Job Management Partner 1/Base Function  
Reference**

---

**3021-3-303-20(E)**

---

## Notices

### ■ Relevant program products

For details about the supported operating systems and the service packs or patches that are required by Job Management Partner 1/Base, see the *Release Notes*.

*For Windows:*

P-2W2C-6LAL Job Management Partner 1/Base 10-50

The above product includes the following:

P-CC242C-6LAL Job Management Partner 1/Base 10-50 (for Windows XP Professional, Windows Server 2003)

P-CC2A2C-6LAL Job Management Partner 1/Base 10-50 (for Windows Vista, Windows Server 2008, Windows 7, Windows Server 2012, Windows 8, Windows 8.1)

*For UNIX:*

P-1J2C-6LAL Job Management Partner 1/Base 10-50 (for HP-UX (IPF))

P-9D2C-6LAL Job Management Partner 1/Base 10-50 (for Solaris (SPARC))

P-1M2C-6LAL Job Management Partner 1/Base 10-50 (for AIX)

P-812C-6LAL Job Management Partner 1/Base 10-50 (for Linux 5 (x86), Linux 5 (AMD/Intel 64), Linux 6 (x86), Linux 6 (AMD/Intel 64))

These products have been developed under a quality management system which has been certified to comply with ISO 9001.

### ■ Trademarks

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

AMD, AMD Opteron, and combinations thereof, are trademarks of Advanced Micro Devices, Inc.

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

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

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

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

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

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

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

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

POSIX stands for Portable Operating System Interface for Computer Environment, which is a set of standard specifications published by the Institute of Electrical and Electronics Engineers, Inc.

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

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

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

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

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

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

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

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

XPG4 stands for X/Open Portability Guide Issue 4, which is a set of specifications published by X/Open Company Limited.

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

The following program product contains some parts whose copyrights are reserved by Oracle Corporation and its affiliates: P-9D2C-6LAL.

The following program product contains some parts whose copyrights are reserved by UNIX System Laboratories, Inc.: P-9D2C-6LAL.

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

## ■ Microsoft product name abbreviations

This manual uses the following abbreviations for Microsoft product names.

Abbreviation	Full name or meaning
Hyper-V	Microsoft(R) Hyper-V(R)
Microsoft Cluster Server	Microsoft(R) Cluster Server
Microsoft Internet Explorer	Microsoft(R) Internet Explorer
	Windows(R) Internet Explorer(R)
Visual C++	Microsoft(R) Visual C++(R)
Windows 7	Microsoft(R) Windows(R) 7 Enterprise
	Microsoft(R) Windows(R) 7 Professional
	Microsoft(R) Windows(R) 7 Ultimate
Windows 8	Windows(R) 8
	Windows(R) 8 Pro
	Windows(R) 8 Enterprise
Windows 8.1	Windows(R) 8.1
	Windows(R) 8.1 Pro
	Windows(R) 8.1 Enterprise

Abbreviation		Full name or meaning
Windows NT		Microsoft(R) Windows NT(R) Server Enterprise Edition Version 4.0
		Microsoft(R) Windows NT(R) Server Network Operating System Version 4.0
		Microsoft(R) Windows NT(R) Workstation Operating System Version 4.0
Windows Server 2003	Windows Server 2003	Microsoft(R) Windows Server(R) 2003, Datacenter Edition
		Microsoft(R) Windows Server(R) 2003, Enterprise Edition
		Microsoft(R) Windows Server(R) 2003, Standard Edition
		Microsoft(R) Windows Server(R) 2003 R2, Datacenter Edition
		Microsoft(R) Windows Server(R) 2003 R2, Enterprise Edition
		Microsoft(R) Windows Server(R) 2003 R2, Standard Edition
	Windows Server 2003 (x64)	Microsoft(R) Windows Server(R) 2003, Datacenter x64 Edition
		Microsoft(R) Windows Server(R) 2003, Enterprise x64 Edition
		Microsoft(R) Windows Server(R) 2003, Standard x64 Edition
		Microsoft(R) Windows Server(R) 2003 R2, Datacenter x64 Edition
		Microsoft(R) Windows Server(R) 2003 R2, Enterprise x64 Edition
		Microsoft(R) Windows Server(R) 2003 R2, Standard x64 Edition
Windows Server 2008	Windows Server 2008	Microsoft(R) Windows Server(R) 2008 Datacenter
		Microsoft(R) Windows Server(R) 2008 Enterprise
		Microsoft(R) Windows Server(R) 2008 Enterprise without Hyper-V(R)
		Microsoft(R) Windows Server(R) 2008 Standard
		Microsoft(R) Windows Server(R) 2008 Standard without Hyper-V(R)
	Windows Server 2008 R2	Microsoft(R) Windows Server(R) 2008 R2 Datacenter
		Microsoft(R) Windows Server(R) 2008 R2 Enterprise
		Microsoft(R) Windows Server(R) 2008 R2 Standard
Windows Server 2012	Windows Server 2012	Microsoft(R) Windows Server(R) 2012 Datacenter
		Microsoft(R) Windows Server(R) 2012 Standard
	Windows Server 2012 R2	Microsoft(R) Windows Server(R) 2012 R2 Datacenter
		Microsoft(R) Windows Server(R) 2012 R2 Standard
Windows Vista		Microsoft(R) Windows Vista(R) Business
		Microsoft(R) Windows Vista(R) Enterprise
		Microsoft(R) Windows Vista(R) Ultimate

Abbreviation		Full name or meaning
Windows XP	Windows XP Professional	Microsoft(R) Windows(R) XP Professional Operating System

*Windows* is sometimes used generically, referring to Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows Server 2012, Windows 8, and Windows 8.1.

*Windows Vista* and later is sometimes used generically, referring to Windows Vista, Windows Server 2008, Windows 7, Windows Server 2012, Windows 8, and Windows 8.1.

## ■ 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.

## ■ Edition history

Dec. 2014: 3021-3-303-20(E)

## ■ Copyright

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

## Summary of amendments

The following table lists changes in this manual (3021-3-303-20(E)) and product changes related to this manual.

Changes	Location
The following OSs were added to the supported OSs: <ul style="list-style-type: none"> <li>Windows 8.1</li> <li>Windows Server 2012 R2</li> </ul>	--
The following functions were added to suppress forwarding of large numbers of events: <ul style="list-style-type: none"> <li>Event-forwarding suppression using an event-forwarding suppression command (<code>jevagtfw</code> command)</li> <li>Event-forwarding suppression using a threshold</li> </ul>	<i>User's Guide: 2.5, 3.1, 3.2.3(6), 3.3.4(6), 4.1, 4.8, 10.1.1, 10.1.2, 10.1.8, 10.1.9, 15. Commands, 15. jevagtfw, 15. jevfwstat, 15. jevreload, 16. Event server settings file, 16. Forwarding settings file, 16. Action definition file for log file trapping, 18.5.1(1), A.1, A.2, C.1, Appendix P</i> <i>Messages: --</i> <i>Function Reference: --</i>
Descriptions about the JP1 events to which local action execution conditions are applied were added.	<i>User's Guide: 2.8.1, 16. Local action execution definition file</i> <i>Messages: --</i> <i>Function Reference: --</i>
HNP_Admin was added to the permissions that are granted to JP1 users at initial settings for user management.	<i>User's Guide: 3.2.1, 3.3.1</i> <i>Messages: --</i> <i>Function Reference: --</i>
Descriptions about the language type in the LANG environment variable for an automated startup script were added.	<i>User's Guide: 3.4.1, 3.4.1(1), 5.8.3, 7.2.1, 15. jbs_start (UNIX only), 15. jbs_start.cluster (UNIX only)</i> <i>Messages: --</i> <i>Function Reference: --</i>
Japanese UTF-8 is now supported as a language type in JP1/Base in HP-UX, Solaris, and AIX.	<i>User's Guide: 3.4.1</i> <i>Messages: --</i> <i>Function Reference: --</i>
The <code>jbsgetcnf</code> command can now be used to collect definition information by specifying a component.	<i>User's Guide: 3.5.2(1)(b), 3.5.3(1)(b), 15. jbsgetcnf</i> <i>Messages: --</i> <i>Function Reference: --</i>
Exclusive control can now be enabled or disabled when outputting to an integrated trace log.	<i>User's Guide: 4.6, 15. hntr2conf, 15. hntr2getconf, 16. Action definition file for log file trapping</i> <i>Messages: --</i> <i>Function Reference: --</i>
The <code>jbshostsimport</code> command can now be used to forcibly register the <code>jp1hosts</code> information in an environment where <code>jp1hosts2</code> information is specified. Descriptions about the behavior when both <code>jp1hosts</code> information and <code>jp1hosts2</code> information is specified were also added.	<i>User's Guide: 5.6.1(4), 6.4.3, 6.4.4(2), 6.4.6, 6.6.1, 6.6.2, 15. jbshostsimport</i> <i>Messages: --</i> <i>Function Reference: --</i>
Notes on event service communication settings were added.	<i>User's Guide: 6.5.1(4), 6.6.3</i> <i>Messages: --</i>

Changes	Location
	<i>Function Reference: --</i>
Notes on using log file trapping to monitor a Unicode file in Windows were added.	<i>User's Guide: 11.1.3(3) Messages: -- Function Reference: --</i>
The JP1/Base setup information can now be collected by one operation.	<i>User's Guide: 15. Commands, 15. jbsparamdump, 16. Definition Files, 16. Collection information file, A.1, A.2 Messages: -- Function Reference: --</i>
Notes on using the jbsmkumap and jbssetumap commands to register user mapping information in the common definition information were added.	<i>User's Guide: 15. jbsmkumap, 15. jbssetumap Messages: -- Function Reference: --</i>
Notes on using the jbssetcnf command to change the JP1/AJS common definition information were added.	<i>User's Guide: 15. jbssetcnf Messages: -- Function Reference: --</i>
In Windows, Unicode (UTF-8)-formatted log files are now supported for monitoring by log file trapping.	<i>User's Guide: 15. jevlogstart, 16. Action definition file for log file trapping Messages: -- Function Reference: --</i>
The following JP1 events were added: 00003D05, 00003D06, 00003D07, 00003D08, 00003D09, 00003D0B, 00003D0C, 00003D0D, 00003D0E	<i>User's Guide: 17.2, 17.3.1 Messages: -- Function Reference: --</i>
The extended attributes that are set to a JP1 event issued by the log file trapping function now include a monitor ID and monitor name.	<i>User's Guide: 17.3.1(28) Messages: -- Function Reference: --</i>
The following items were added to the information that can be collected by the data collection tool: <ul style="list-style-type: none"> <li>• Assigning user rights</li> <li>• Group policy</li> <li>• Range of dynamic ports (IPV4)</li> <li>• Range of dynamic ports (IPV6)</li> <li>• Exclusive control of the integrated trace log</li> <li>• Output destination of the integrated trace log</li> </ul>	<i>User's Guide: 18.3.1(1), 18.3.1(2), 18.3.2(2) Messages: -- Function Reference: --</i>
The number and size of the configuration management log files were changed.	<i>User's Guide: A.1(2), A.2(2) Messages: -- Function Reference: --</i>
Descriptions about the parameters defined in the communication protocol settings file were added.	<i>User's Guide: H.10 Messages: -- Function Reference: --</i>
Descriptions about the differences between the communication protocols of JP1/Base version 06-51 or earlier and JP1/Base version 06-71 or later were added.	<i>User's Guide: H.11 Messages: -- Function Reference: --</i>
Descriptions about the communication protocols specified during cluster setup were added.	<i>User's Guide: H.12 Messages: --</i>

Changes	Location
	<i>Function Reference: --</i>
Descriptions about the messages that are output when a request is denied by connection restriction were added.	<i>User's Guide: M.5</i> <i>Messages: --</i> <i>Function Reference: --</i>
The following messages were added: KAJP1083-W, KAJP1084-I, KAJP1085-I, KAJP1086-W, KAJP1087-W, KAJP1350-I, KAJP1351-I, KAJP1352-E, KAJP1401-I, KAJP1402-E, KAJP1403-E, KAJP1404-I, KAJP1405-I, KAJP1406-E, KAJP1407-E, KAJP1408-I, KAJP1410-I, KAJP1411-E, KAJP1413-I, KAJP1414-I, KAJP1415-E, KAJP1416-I, KAJP1417-I, KAJP1418-I, KAJP1419-I, KAJP1420-I, KAJP1421-I, KAJP1422-I, KAJP1423-E, KAJP1424-I, KAJP1425-E, KAJP1426-I, KAJP1427-I, KAJP1428-I, KAJP1429-I, KAJP1430-I, KAJP1431-I, KAJP1432-E, KAJP1433-I, KAJP1434-E, KAJP1435-E, KNAM9001-I, KNAM9002-I, KNAM9003-E, KNAM9004-E, KNAM9005-E, KNAM9006-E, KNAM9007-E, KNAM9008-E, KNAM9009-E, KNAM9010-E, KNAM9011-W, KNAM9012-I, KNAM9013-E, KNAM9014-E, KNAM9015-W, KNAM9016-W, KNAM9020-I, KNAM9021-E, KNAM9022-I, KNAM9023-E, KNAM9024-I, KNAM9025-E, KNAM9026-I, KNAM9027-E, KNAM9028-W	<i>User's Guide: --</i> <i>Messages: 1.2.1, 1.2.17, 1.4.1, 2.1, 2.17</i> <i>Function Reference: --</i>
The following messages were modified: KAJP1033-E, KAVA0443-E, KAVA0590-E, KAVA0592-E, KAVA0678-E, KAVA0901-E, KAVA0906-E, KAVA3650-I, KAVA6726-E, KAVB2027-E, KAVB3098-E, KAVB3107-E, KNAM3130-E, KNAM3170-E	<i>User's Guide: --</i> <i>Messages: 2.1, 2.2, 2.4, 2.7, 2.9, 2.10, 2.15</i> <i>Function Reference: --</i>
Visual C++(R) 2012 was added to the supported compilers.	<i>User's Guide: --</i> <i>Messages: --</i> <i>Function Reference: 2.1.1(2)</i>

**Legend:**

*User's Guide: Job Management Partner 1/Base User's Guide*

*Messages: Job Management Partner 1/Base Messages*

*Function Reference: Job Management Partner 1/Base Function Reference*

--: N/A

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

# Preface

This manual explains in detail the functions provided by Job Management Partner 1/Base and the procedures used to extend the functions of Job Management Partner 1/Base during development of systems linked to Job Management Partner 1/Integrated Management. This manual is intended for all operating systems. When there is a difference in the functions available for the supported operating systems, a distinction to that effect is made in the manual.

In this manual, *Job Management Partner 1* is abbreviated to *JP1*.

## ■ Intended readers

This manual is intended for users who use JP1/IM and JP1/Base to develop systems that work with JP1/IM.

This manual assumes that the readers understand the functionality of JP1/IM and JP1/Base.

## ■ Organization of this manual

This manual consists of the following parts:

### PART 1: Overview

Part 1 provides a brief overview of customizing and extending the JP1/Base functions.

### PART 2: Operation

Part 2 explains how to customize the JP1/Base functions.

### PART 3: Reference

Part 3 describes the JP1/Base functions in reference format.

## ■ JP1/Base manual organization

The JP1/Base documentation is divided into three manuals. Read the manual appropriate to your purpose, referring to the content of each manual shown in the following table.

Manual	Content
<i>Job Management Partner 1/Base User's Guide</i>	<ul style="list-style-type: none"><li>• Overview and functionality of JP1/Base</li><li>• Setup of each function</li><li>• Commands, definition files, JP1 events</li><li>• Troubleshooting</li><li>• Processes, port numbers, operation logs</li></ul>
<i>Job Management Partner 1/Base Messages</i>	Messages
<i>Job Management Partner 1/Base Function Reference</i>	<ul style="list-style-type: none"><li>• Procedures for issuing and acquiring JP1 events with JP1 programs and user applications</li><li>• Functions</li></ul>

## ■ Conventions: "Administrator permissions" as used in this manual

In this manual, *Administrator permissions* refers to Administrator permissions for the local PC. The local user, domain user, or user of the Active Directory environment can perform tasks requiring Administrator permissions if granted Administrator permissions for the local PC.

## ■ Conventions: Fonts and symbols

Font and symbol conventions are classified as:

- General font conventions
- Conventions in syntax explanations

These conventions are described below.

### General font conventions

The following table lists the general font conventions.

Font	Convention
<b>Bold</b>	Bold type indicates text on a window, other than the window title. Such text includes menus, menu options, buttons, radio box options, or explanatory labels. For example, bold is used in sentences such as the following: <ul style="list-style-type: none"><li>• From the <b>File</b> menu, choose <b>Open</b>.</li><li>• Click the <b>Cancel</b> button.</li><li>• In the <b>Enter name</b> entry box, type your name.</li></ul>
<i>Italics</i>	Italics are used to indicate a placeholder for some actual text provided by the user or system. Italics are also used for emphasis. For example: <ul style="list-style-type: none"><li>• Write the command as follows: <i>copy source-file target-file</i></li><li>• Do <i>not</i> delete the configuration file.</li></ul>
Code font	A code font indicates text that the user enters without change, or text (such as messages) output by the system. For example: <ul style="list-style-type: none"><li>• At the prompt, enter <code>dir</code>.</li><li>• Use the <code>send</code> command to send mail.</li><li>• The following message is displayed: <code>The password is incorrect.</code></li></ul>

Examples of coding and messages appear as follows (although there may be some exceptions, such as when coding is included in a diagram):

```
MakeDatabase
...
StoreDatabase temp DB32
```

In examples of coding, an ellipsis (...) indicates that one or more lines of coding are not shown for purposes of brevity.

### Conventions in syntax explanations

Syntax definitions appear as follows:

**StoreDatabase** [temp|perm] (*database-name* ...)

The following table lists the conventions used in syntax explanations:

Example font or symbol	Convention
	In syntax explanations, a vertical bar separates multiple items, and has the meaning of OR. For example: A   B   C means A, or B, or C.
{ }	In syntax explanations, curly brackets indicate that only one of the enclosed items is to be selected. For example: {A   B   C} means only one of A, or B, or C.
[ ]	In syntax explanations, square brackets indicate that the enclosed item or items are optional. For example: [A] means that you can specify A or nothing. [B   C] means that you can specify B, or C, or nothing.
...	In coding, an ellipsis (...) indicates that one or more lines of coding have been omitted. In syntax explanations, an ellipsis indicates that the immediately preceding item can be repeated as many times as necessary. For example: A, B, B, ... means that, after you specify A, B, you can specify B as many times as necessary.
()	Parentheses indicate the range of items to which the vertical bar ( ) or ellipsis (...) is applicable.
△	This symbol is used to explicitly indicate a space. For example, AAA △ BBB means that you must place a space between AAA and BBB.
\	When a syntax character shown above is used as an ordinary character, a backslash is prefixed to the character. For example, \  means   as an ordinary character, not as a syntax character.
StoreDatabase	Code-font characters must be entered exactly as shown.
<i>database-name</i>	This font style marks a placeholder that indicates where appropriate characters are to be entered in an actual command.
<b>SD</b>	Bold code-font characters indicate an abbreviation for a command.

## ■ Conventions: Version numbers

The version numbers of Hitachi program products are usually written as two sets of two digits each, separated by a hyphen. For example:

- version 1.00 (or version 1.0) is written as 01-00
- version 2.05 is written as 02-05
- version 2.50 (or version 2.5) is written as 02-50
- version 12.25 is written as 12-25

The version number might be shown on the spine of a manual as *Ver. 2.00*, but the same version number would be written in the program as *02-00*.

## ■ JP1/Base installation folder for Windows

This manual refers to the JP1/Base installation folder in Windows as follows:

Product name	Reference to installation folder	Installation folder#
JP1/Base	<i>installation-folder</i>	In an x86 environment: <i>system-drive</i> : \Program Files\HITACHI\JP1Base In an x64 environment: <i>system-drive</i> : \Program Files (x86)\HITACHI\JP1Base

#

Denotes the installation folder used when the product is installed with initial settings.

Example:

This manual uses the following convention to represent the full path of the `JevApi.h` header file needed to compile a source file:

*installation-folder*\include\JevApi.h

For the default installation folder, this path would actually be *system-drive*: \Program Files\HITACHI\JP1Base\include\JevApi.h.

For Windows Vista or later, the manual uses the expression *system-drive*: \ProgramData. The actual value is determined by the OS environment variable when the program is installed. The installation destination may differ depending on the environment.

## ■ Other reference information

For other reference information, see *Reference Material for this Manual* in the *Job Management Partner 1/Base User's Guide*.

# Contents

Notices	2
Summary of amendments	6
Preface	9

## Part 1: Overview

<b>1</b>	<b>Overview of Customizing and Extending the Functionality of JP1/Base 15</b>
1.1	Features 16
1.2	Sample source files of functions 17

## Part 2: Operation

<b>2</b>	<b>Issuing and Acquiring JP1 Events 18</b>
2.1	Overview of functions for issuing and acquiring events 19
2.1.1	Prerequisites 20
2.2	Procedures for issuing and acquiring JP1 events 22
2.2.1	Procedure for issuing JP1 events 22
2.2.2	Procedure for acquiring JP1 events 25
2.2.3	Compiling the source files 29
2.3	Migrating user applications from an earlier version 33
2.3.1	Migrating without recompiling 33
2.3.2	Recompiling before migrating 33

## Part 3: Reference

<b>3</b>	<b>Functions 34</b>
	Function description format 35
	Notes common to all functions 36
	List of functions 37
	JevFreeEvent 39
	JevGetArrivedTime 40
	JevGetBaseID 41
	JevGetClose 42
	JevGetCodeSet 43
	JevGetDestinationAddress 44
	JevGetDestinationServer 45
	JevGetDetailInformation 46
	JevGetEvent 47
	JevGetExtAttrDirect 49

JevGetExtID	50
JevGetFirstExtAttr	51
JevGetMessage	52
JevGetNextExtAttr	53
JevGetOpen	54
JevGetProcessID	56
JevGetRegistFactor	57
JevGetRegistGroupID	58
JevGetRegistGroupName	59
JevGetRegistTime	60
JevGetRegistUserID	61
JevGetRegistUserName	62
JevGetSequenceNumber	63
JevGetSourceAddress	64
JevGetSourceSequenceNumber	65
JevGetSourceServer	66
JevRegistEvent	67

## **Appendixes 69**

A	Criteria for Setting JP1 Event Attributes	70
A.1	Basic attributes	70
A.2	Extended attributes	71
B	Sample Source Files	75
B.1	Details of the sample source files	75

## **Index 80**

# 1

## Overview of Customizing and Extending the Functionality of JP1/Base

This chapter provides an overview of what you can do with JP1/Base by extending its functionality, along with samples of the functions provided by JP1/Base.

## 1.1 Features

---

By using JP1/Base functions and definition files, you can perform the following operations:

### Issue user-defined JP1 events

You can use JP1/Base to define system-issued events as JP1 events with user-defined event attributes. You can also configure such events so that they can be issued by user applications. JP1 events defined by users are called *user-defined events*. You can freely define the attributes of user-defined events.

You can use the JP1 event issuing function to issue user-defined events. For details about how to use the JP1 event issuing function to issue user-defined events, see [2. Issuing and Acquiring JP1 Events](#).

You can also display in JP1/IM - View the user-defined event attributes that have been added to a user-defined event. To do this, you must create definition files that contain descriptions of your user-defined event attributes. For details, see the manual *Job Management Partner 1/Integrated Management - Manager Command and Definition File Reference*.

### Acquire JP1 events

JP1 events that a JP1 program or user application has registered with JP1/Base's event database can be acquired directly by other JP1 programs and user applications. After a user-defined event is issued from a user application to JP1/Base and the event is registered with the event database as a JP1 event, other user applications can use the event.

You can use the JP1 event acquisition function to acquire JP1 events.

For details about how to use the JP1 event acquisition function to acquire JP1 events, see [2. Issuing and Acquiring JP1 Events](#).

### Add extended attributes and messages to events in JP1/SES format

JP1/SES is a JP1 program of Version 5 or earlier. An event that JP1/SES handled is called an *event in JP1/SES format*. You cannot display events in the JP1/SES format in the JP1/IM Event Console window, because they do not have extended attributes. However, by using JP1/Base to add extended attributes to events in JP1/SES format, you can monitor them in the JP1/IM Event Console window as JP1 events. In addition to adding extended attributes, you can also add a message.

For details about how to use a definition file to add extended attributes and messages in JP1/SES format, see the manual *Job Management Partner 1/Base User's Guide*.

## 1.2 Sample source files of functions

---

JP1/Base provides you sample source files of functions. Editing and compiling the sample source files enables users to easily create and issue JP1 events based on the user's environment, as well as to obtain JP1 events. For details of the sample source files, see *B. Sample Source Files*.

# 2

## Issuing and Acquiring JP1 Events

This chapter provides an overview of the function for issuing JP1 events with user-defined event attributes directly from user applications, and the functions for directly acquiring JP1 events in other JP1 programs and user applications. It also explains the prerequisites and procedures for each function.

## 2.1 Overview of functions for issuing and acquiring events

### Issuing JP1 events

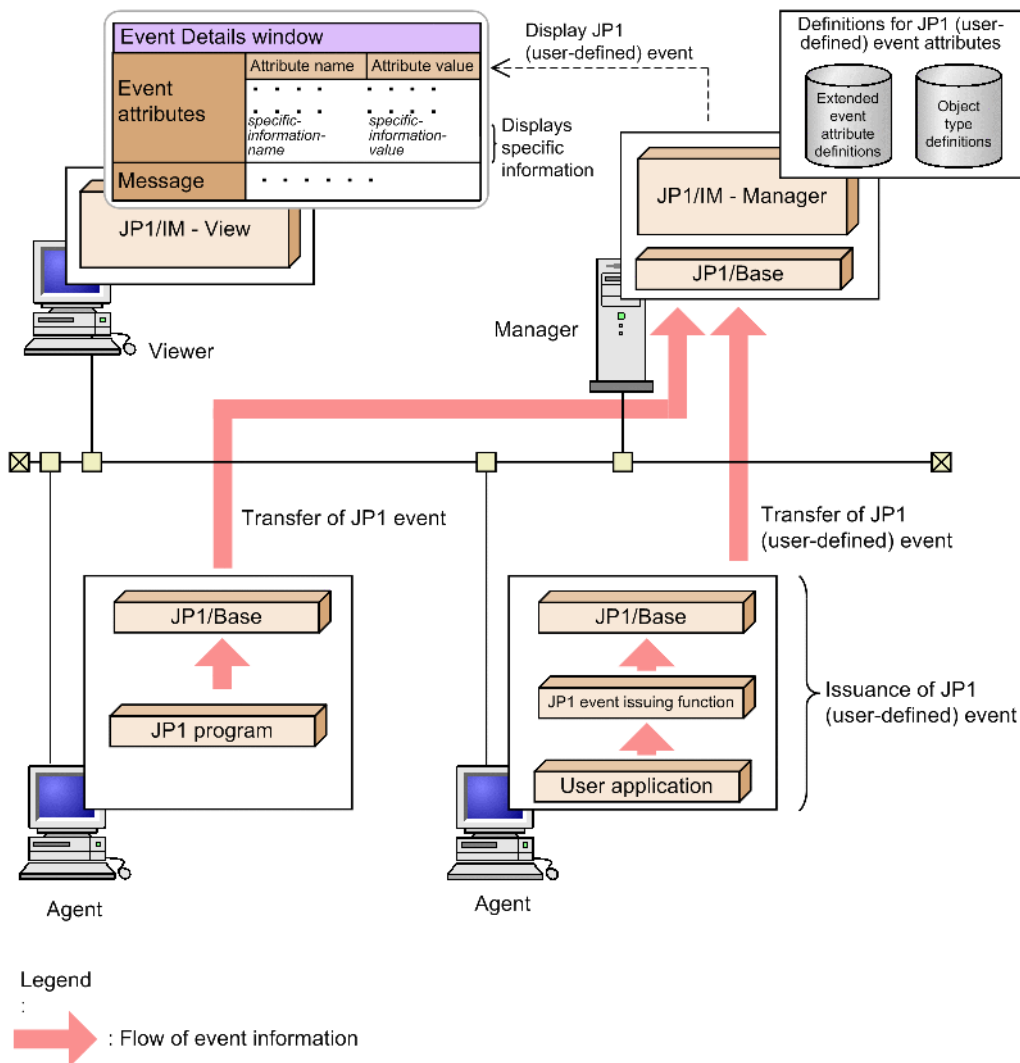
JP1/IM enables you to monitor events by converting application-specific log files, SNMP traps, and the Windows event log to JP1 events. However, you cannot use JP1/IM to define application-specific event attributes and other information in detail.

You can use the JP1 event issuing function of JP1/Base to issue user-defined events that include user-defined event attributes directly from user applications.

These user-defined events that include user-defined event attributes (specific information in extended attributes) can then be displayed in the Event Details window by using JP1/IM to create definition files.

The following figure provides an overview of issuing a JP1 (user-defined) event to display the user-defined event attributes.

Figure 2–1: Overview of issuing a JP1 (user-defined) event to display the user-defined event attributes



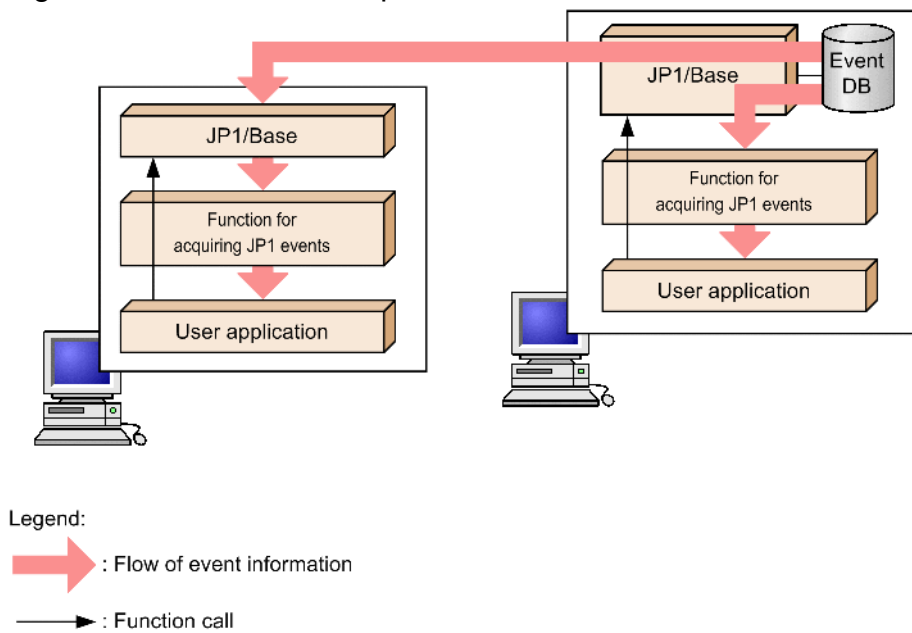
### Acquiring JP1 events

JP1/Base allows you to register and manage a wide variety of events issued by a system as JP1 events in the event database. However, user applications cannot directly use these JP1 events.

By using the JP1 event acquisition functions, user applications are able to acquire and use JP1 events directly from the JP1/Base event database.

The following figure shows an overview of JP1 event acquisition.

Figure 2–2: JP1 event acquisition



## 2.1.1 Prerequisites

The following lists the prerequisites required for using functions provided by JP1/Base (JP1 event issuing function and JP1 event acquisition functions).

- An environment for compiling the source files that use the functions (JP1 event issuing function and JP1 event acquisition functions)
  - JP1/Base for the OS in use
  - A compiler for the OS in use

### (1) Installation of JP1/Base

JP1/Base is required in order to compile and execute the source files that use the functions provided by JP1/Base. This is because the libraries and header files provided by JP1/Base are used during compiling and execution. Therefore, before you do anything else, install JP1/Base on the machine on which you will be performing compiling and execution.

### (2) Installation of a compiler

To compile the source files that use the functions provided by JP1/Base, you need one of the compilers listed in the following table. Before you start operations, install one of these compilers on the machine on which you will compile the source files.

Table 2–1: Compilers

OS	Compiler
Windows XP Professional	Visual C++(R) .NET 2003
Windows Vista	Visual C++(R) 2005
Windows Server 2003	Visual C++(R) 2008
Windows Server 2008	Visual C++(R) 2010
Windows 7	Visual C++(R) 2012

OS	Compiler
Windows 8 Windows 8.1 Windows Server 2012	
HP-UX (IPF)	HP C/aC++ A.06.05 or later
Solaris	Sun Studio 10 Sun Studio 12
AIX	XL C/C++ Enterprise Edition V7.0 for AIX XL C/C++ Enterprise Edition V8.0 for AIX XL C/C++ Enterprise Edition V9.0 for AIX
Linux 5 (x86) Linux 6 (x86)	gcc version 4.1.2 gcc version 4.4.5

## 2.2 Procedures for issuing and acquiring JP1 events

---

### 2.2.1 Procedure for issuing JP1 events

To issue JP1 events:

1. Decide the types and attributes of the JP1 events you want to issue.
2. Write code that uses the JP1 event issuing function.
3. Compile the source files.

To be able to view in JP1/IM - View the user-defined event attributes that are appended to a JP1 event, you must then use JP1/IM to create the following definition files on a machine where JP1/IM - Manager is installed:

- Definition file for extended event attributes
- Definition file for object types

For details about how to create these definition files using JP1/IM, see the manual *Job Management Partner 1/Integrated Management - Manager Command and Definition File Reference*.

#### (1) Determining the types and attributes of the JP1 events to be issued

To issue JP1 (user-defined) events, you must first decide what kinds of events to issue as JP1 events. The performance of the JP1/Base event service depends on the number of JP1 events that may be issued. Therefore, we recommend that you issue JP1 (user-defined) events only for those JP1 events that are needed by JP1/IM to perform system monitoring.

Next, you must determine the types of event attributes you wish to issue. To determine the *event attributes*, consider the information you need to know about an application when you monitor events with JP1/IM. Determine the event attributes of all JP1 (user-defined) events for each application in advance.

You can use the JP1 event attribute values as arguments for initiating automated actions and invoking monitor windows in JP1/IM. For details on the event attributes used in the example described below, see [A. Criteria for Setting JP1 Event Attributes](#).

The example below uses the Windows application `SAMPLE` to explain issuing the startup event and the abnormal termination event. The character strings enclosed in parentheses are the names of the arguments in the JP1 event issuing function.

Types of JP1 events

- Startup event (JP1 event issued at the startup of the application)  
Event ID (BaseID): `0x00000001`  
Message (message): `Starts the SAMPLE application.`
- Abnormal termination event (JP1 event issued at the abnormal termination of the application)  
Event ID (BaseID): `0x00000002`  
Message (message): `The SAMPLE application terminated abnormally.`

Event attributes including extended attributes (`extattrs`): Startup event

Assign the following attributes to the startup event of the `SAMPLE` application.

**Table 2–2: Attributes to be assigned to the startup event**

Attribute type	Item	Attribute name	Description
Basic attribute	Event ID	--	0x00000001
	Message	--	Starts the SAMPLE application.
Extended attributes (common attributes)	Event level	SEVERITY	Notice
	User name	USER_NAME	SAMPLE_USER
	Product name	PRODUCT_NAME	/COMPANY/APP1/SAMPLE_PRODUCT (product name)
	Object type	OBJECT_TYPE	SAMPLE
	Object name	OBJECT_NAME	SAMPLE_NAME
	Root object type	ROOT_OBJECT_TYPE	ROOT_SAMPLE
	Root object name	ROOT_OBJECT_NAME	ROOT_SAMPLE_NAME
	Object ID	OBJECT_ID	SAMPLE_ID
	Occurrence	OCCURRENCE	START
	Start time	START_TIME	Start time of the SAMPLE application. The number of seconds from 00:00:00 on UTC 1970-01-01.
	Platform type	PLATFORM	NT
	Version information	ACTION_VERSION	0600
Extended attributes (user-specific attributes)	SAMPLE common attribute 1	COMMON_ATTR1	NATIVE
	SAMPLE common attribute 2	COMMON_ATTR2	TRUE
	SAMPLE start attribute 1	START_ATTR1	SAMPLE1
	SAMPLE start attribute 2	START_ATTR2	SAMPLE2

Event attributes including extended attributes (extattrs): Abnormal termination event

Assign the following attributes to the abnormal termination event of the SAMPLE application.

**Table 2–3: Attributes to be assigned to the abnormal termination event**

Attribute type	Item	Attribute name	Description
Basic attribute	Event ID	--	0x00000002
	Message	--	The SAMPLE application terminated abnormally.
Extended attributes (common attributes)	Event level	SEVERITY	Error
	User name	USER_NAME	SAMPLE_USER
	Product name	PRODUCT_NAME	/COMPANY/APP1/SAMPLE_PRODUCT (product name)
	Object type	OBJECT_TYPE	SAMPLE
	Object name	OBJECT_NAME	SAMPLE_NAME
	Root object type	ROOT_OBJECT_TYPE	ROOT_SAMPLE

Attribute type	Item	Attribute name	Description
	Root object name	ROOT_OBJECT_NAME	ROOT_SAMPLE_NAME
	Object ID	OBJECT_ID	SAMPLE_ID
	Occurrence	OCCURRENCE	END
	End time	END_TIME	End time of the SAMPLE application. The number of seconds from 00:00:00 on UTC 1970-01-01.
	Result code	RESULT_CODE	Termination code that the SAMPLE application returns when it terminates
	Platform type	PLATFORM	NT
	Version information	ACTION_VERSION	0600
Extended attributes(user-specific attributes)	SAMPLE common attribute 1	COMMON_ATTR1	NATIVE
	SAMPLE common attribute 2	COMMON_ATTR2	TRUE
	SAMPLE end attribute 1	END_ATTR1	SAMPLE1
	SAMPLE end attribute 2	END_ATTR2	SAMPLE2

## (2) Writing code that uses the JP1 event issuing function

The coding example for the SAMPLE application to issue the startup event is as follows:

```
#include <stdio.h>
#include <time.h>
#include "JevApi.h"

int regist_start_event()
{
    int rc;                /* Return code */
    long status = 0;       /* Detailed error code */
    const char* server;    /* Event server name */
    long baseID;           /* Event ID */
    const char* message;   /* Message */
    char starttime[32];
    const char* extattrs[16]; /* Array for storing extended
attributes */

    /* Set the destination event server name. */
    server = NULL;

    /* Set the event ID. */
    baseID = 0x00000001;

    /* Set the message. */
    message = "Starts the SAMPLE application.";

    /* Set the extended attributes. */
    extattrs[0] = "SEVERITY=Notice";
    extattrs[1] = "USER_NAME=SAMPLE_USER";
    extattrs[2] = "PRODUCT_NAME=/COMPANY/APP1/SAMPLE_
```

```

        PRODUCT";
extattrs[3] = "OBJECT_TYPE=SAMPLE";
extattrs[4] = "OBJECT_NAME=SAMPLE_NAME";
extattrs[5] = "OBJECT_ROOT_TYPE=ROOT_SAMPLE";
extattrs[6] = "OBJECT_ROOT_NAME=ROOT_SAMPLE_NAME";
extattrs[7] = "OBJECT_ID=SAMPLE_ID";
extattrs[8] = "OCCURRENCE=START";
sprintf(starttime, "START_TIME=%ld", time(NULL));
extattrs[9] = starttime;
extattrs[10] = "PLATFORM=NT";
extattrs[11] = "VERSION=0600";
extattrs[12] = "COMMON_ATTR1=NATIVE";
extattrs[13] = "COMMON_ATTR2=TRUE";
extattrs[14] = "START_ATTR1=SAMPLE1";
extattrs[15] = "START_ATTR2=SAMPLE2";

/* Register the JP1 event. */
rc = JevRegistEvent(&status,
                    server,
                    baseID,
                    message,
                    extattrs,
                    16);

if(rc < 0) {
    fprintf(stderr,
            "JevRegistEvent() failed. status = %ld\n",
            status);
    return -1;
}

return 0;
}

```

## 2.2.2 Procedure for acquiring JP1 events

To acquire JP1 events:

1. Determine the types and attributes of the JP1 events you want to acquire.
2. Define an event acquisition filter to specify the JP1 events to acquire.
3. Write code that uses JP1 event acquisition functions.
4. Compile the source files.

### (1) Determining the types and attributes of the JP1 events to be acquired

JP1/Base registers a wide variety of event types to the event database as JP1 events. Therefore, you must first determine the types of events you want to acquire from the event database.

Next, determine the event attributes to acquire from these JP1 events. When you are deciding which event attributes to acquire, consider the information you need to know about the application in question. Determine the event attributes of all JP1 events being acquired for each application in advance.

The following explanations are based on acquiring the startup events listed in *2.2.1(1) Determining the types and attributes of the JP1 events to be issued* that are issued as JP1 events in the application SAMPLE.

## (2) Defining an event acquisition filter to specify the JP1 events to acquire

To select only the JP1 events you want to acquire, you must define an event acquisition filter. For details about the syntax of event acquisition filters, see the section on filter syntax in the manual *Job Management Partner 1/Base User's Guide*. This subsection provides an example of an event acquisition filter for acquiring the startup events listed in *2.2.1(1) Determining the types and attributes of the JP1 events to be issued* that are coded in the application SAMPLE.

To acquire startup events, you must first consider how to create an event acquisition filter with the following conditions:

- Event ID: 0x00000001
- Value of the extended attribute SEVERITY: Notice
- Value of the extended attribute PRODUCT\_NAME: /COMPANY/APP1/SAMPLE\_PRODUCT

A filter targeting JP1 events that satisfy the above conditions can be used to acquire startup events. The following is an example of such an event acquisition filter:

```
B.ID IN 00000001
E.SEVERITY IN Notice
E.PRODUCT_NAME IN /COMPANY/APP1/ SAMPLE_PRODUCT
```

*Note:*

- If you specify a Japanese string for a condition of an event acquisition filter, make sure that the character set matches the locale information (such as the LANG environment variable) used for execution of JP1 event acquisition functions. If the character set for the string specified for a condition of an event acquisition filter differs from the locale information used for execution of JP1 event acquisition functions, JP1 events cannot be acquired.
- If you define an exclusion condition in an event acquisition filter, connect to an event server of version 09-00 or later. An error occurs (JEV\_S\_FILTER\_ERROR) if you connect to an event server of version 08-00 or earlier.

## (3) Writing code that uses JP1 event acquisition functions

JP1 event acquisition functions are used when a JP1 program or user application acquires JP1 events. The following explains how to issue JP1 event acquisition functions to acquire JP1 events from the event database of JP1/Base.

To issue JP1 event acquisition functions:

1. Issue a function that requests starting the acquisition of JP1 events.

Issue the `JevGetOpen` function to the event server to request starting the acquisition of JP1 events, and to connect a JP1 program or user application to the event server. Note that the user who requests starting the acquisition of JP1 events must be preconfigured in the `users` parameter in the event server settings (`conf`) file for JP1/Base.

2. Issue functions that request acquisition of JP1 events.

Use various functions to acquire JP1 events and the attributes set in the JP1 events.

3. Issue a function that reports ending the acquisition of JP1 events.

Issue the `JevGetClose` function to the event server to notify the server of the end of JP1 event acquisition, and to disconnect the JP1 program or user application from the event server.

For details about JP1 event acquisition functions, see [3. Functions](#). For details about what types of event attributes can be acquired, see [A. Criteria for Setting JP1 Event Attributes](#).

The following is a coding example for acquiring the startup events listed in [2.2.1\(1\) Determining the types and attributes of the JP1 events to be issued](#) that are coded in the application SAMPLE.

```
#include <stdio.h>
#include <string.h>
#include "JevApi.h"

int get_start_event()
{
    int rc;                /* Return code */
    long position;        /* Sequence number within the event database */
    long status;          /* Status code address */
    char filter[256];     /* Filter statement buffer */
    const char *server;   /* Event server name */
    const char *message;  /* Pointer to the message */
    const char *name;     /* Pointer to the extended attribute name */
    const char *value;    /* Pointer to the extended attribute value */
    JEVGETKEY key;        /* Handle for acquiring JP1 events */
    JP1EVENT event;       /* Handle for accessing JP1 events */
    JEVACCESSTYPE access; /* Action when no JP1 event exists*/

    /* Set the filter statement to acquire JP1 events. */
    strcpy(filter, "B.ID IN 0x00000001\n");
    strcat(filter, "E.SEVERITY IN Notice\n");
    strcat(filter,
            "E.PRODUCT_NAME IN /COMPANY/APP1/SAMPLE_PRODUCT");

    /* Connect to the event server of the physical host. */
    status = 0;
    /* Event server of the physical host to connect to */
    server = NULL;
    /* Acquisition starts with sequence number 0 within the event database. */
    position = 0;
    key = JevGetOpen(&status, server, filter, position);
    if(key == NULL){
        fprintf(stderr,
                "JevGetOpen() failed. Status = %ld\n",
                status);
        return -1;
    }

    /* Acquire all the JP1 events which match the filter condition. */
    while(1) {
        status = 0;
        /* Error return when no JP1 event matches the filter condition */
        access = JEVGET_NOWAIT;
        event = JevGetEvent(&status, key, access);
        if(event == NULL){
            if(status == JEV_S_NO_EVENT) {
                /* No more JP1 event matches the filter condition. */
                break;
            }
            else {
                /* Error occurred while acquiring JP1 events. */
            }
        }
    }
}
```

```

        fprintf(stderr,
                "JevGetEvent() failed. Status = %ld\n",
                status);
        JevGetClose(&status, key);
        return -1;
    }
}

/* Acquire a message. */
status = 0;
rc = JevGetMessage(&status, event, &message);
if(rc < 0){
    fprintf(stderr,
            "JevGetMessage() failed. Status = %ld\n",
            status);
    JevFreeEvent(&status, event);
    JevGetClose(&status, key);
    return -1;
}
else{
    printf("JevGetMessage() message = %s\n", message);
}

/* Acquire the (first) extended attribute. */
status = 0;
rc = JevGetFirstExtAttr(&status, event, &name, &value);
if(rc < 0){
    fprintf(stderr,
            "JevGetFirstExtAttr() failed. Status = %ld\n",
            status);
    JevFreeEvent(&status, event);
    JevGetClose(&status, key);
    return -1;
}
else{
    printf("JevGetFirstExtAttr() name = %s\n", name);
    printf("JevGetFirstExtAttr() value = %s\n", value);
}

/* Acquire the (subsequent) extended attribute. */
while(1) {
    status = 0;
    rc = JevGetNextExtAttr(&status, event, &name, &value);
    if(rc < 0 ){
        if(status == JEV_S_EXTATTR_EOD) {
            /* No more extended attribute exists. */
            break;
        }
        else {
/* Error occurred while acquiring extended attributes. */
            fprintf(stderr,
                    "JevGetNextExtAttr() failed.
                    Status = %ld\n", status);
            JevFreeEvent(&status, event);
            JevGetClose(&status, key);
            return -1;
        }
    }
}
}

```

```

        else {
            printf("JevGetNextExtAttr() name = %s\n", name);
            printf("JevGetNextExtAttr() value = %s\n", value);
        }
    }

    /* Release the memory allocated for the acquired JP1 events. */
    rc = JevFreeEvent(&status, event);
    if(rc < 0){
        fprintf(stderr,
            "JevFreeEvent() failed. Status = %ld\n",
            status);
        JevGetClose(&status, key);
        return -1;
    }
}

/* Disconnect the event server.*/
rc = JevGetClose(&status, key);
if(rc < 0){
    fprintf(stderr,
        "JevGetClose() failed. Status = %ld\n",
        status);
    return -1;
}

return 0;
}

```

## 2.2.3 Compiling the source files

To issue and acquire JP1 events, you must first compile and link the code source files.

Files needed for compiling:

- JP1/Base header file (installed when JP1/Base is installed)
- Source files created in C or C++ (user-created files)

The location of the header file is as follows:

Windows: *installation-folder*\include\JevApi.h

UNIX: /opt/jp1base/include/JevApi.h

Files needed for linking:

- Libraries (installed when JP1/Base is installed)

Note that the required libraries vary by OS and by compiler. The following table lists the libraries required for each OS.

**Table 2–4: Libraries required for each OS**

OS	Required library
Windows XP Professional Windows Vista Windows Server 2003 Windows Server 2008	<i>installation-folder</i> \lib\libJevApiA.lib (for 32-bit multi-threaded applications)

OS	Required library
Windows 7 Windows 8 Windows 8.1 Windows Server 2012	
Solaris AIX Linux 5 (x86) Linux 6 (x86)	/opt/jplbase/lib/libJevApiAst.a (for 32-bit single-threaded applications) /opt/jplbase/lib/libJevApiAmt.a (for 32-bit multi-threaded applications)
HP-UX (IPF)	/opt/jplbase/lib/libJevApiAst32.a (for 32-bit single-threaded applications) /opt/jplbase/lib/libJevApiAmt32.a (for 32-bit multi-threaded applications) /opt/jplbase/lib/libJevApiAst64.a (for 64-bit single-threaded applications) /opt/jplbase/lib/libJevApiAmt64.a (for 64-bit multi-threaded applications)

The table below lists for each OS the options to specify when compiling and linking the source files.

*Note:*

When you compile and link source files in Visual Studio Integrated Development Environment (GUI) for Windows, use options that are appropriate for configuring the environment, from among the compile and link options listed in the tables below.

**Table 2–5: Compile options**

OS	Compile option
Windows XP Professional Windows Vista Windows Server 2003 Windows Server 2008 Windows 7 Windows 8 Windows 8.1 Windows Server 2012 (32-bit multi-threaded)	/MD /I " <i>installation-folder</i> \include"
HP-UX (IPF) (32-bit single-threaded)	-Aa# -I/opt/jplbase/include
HP-UX (IPF) (32-bit multi-threaded)	-Aa# -mt -I/opt/jplbase/include
HP-UX (IPF) (64-bit single-threaded)	+DD64 -Aa# -I/opt/jplbase/include
HP-UX (IPF) (64-bit multi-threaded)	+DD64 -Aa# -mt -I/opt/jplbase/include
Solaris (32-bit single-threaded)	-I/opt/jplbase/include
Solaris (32-bit multi-threaded)	-mt -D_THREAD_SAFE -I/opt/jplbase/include
AIX (32-bit single-threaded)	-I/opt/jplbase/include
AIX	-D_REENTRANT -D_THREAD_SAFE -I/opt/jplbase/include

OS	Compile option
(32-bit multi-threaded)	
Linux 5 (x86) Linux 6 (x86) (32-bit single-threaded)	-I/opt/jplbase/include
Linux 5 (x86) Linux 6 (x86) (32-bit multi-threaded)	-D_REENTRANT -D_THREAD_SAFE -I/opt/jplbase/include

#: The `-Aa` option for HP-UX is needed only when you use the C compiler (`cc`) to compile. You can replace the `-Aa` option with the `-Ae` option, but do not specify the `-Ac` option. The `-Aa` option can be omitted if you use the C++ compiler (`aCC`).

**Table 2–6: Link options**

OS	Link option
Windows XP Professional Windows Vista Windows Server 2003 Windows Server 2008 Windows 7 Windows 8 Windows 8.1 Windows Server 2012 (32-bit multi-threaded)	<code>"installation-folder\lib\libJevApiA.lib"</code>
HP-UX (IPF) (32-bit single-threaded)	<code>/opt/jplbase/lib/libJevApiAst32.a</code>
HP-UX (IPF) (32-bit multi-threaded)	<code>-mt /opt/jplbase/lib/libJevApiAmt32.a</code>
HP-UX (IPF) (64-bit single-threaded)	<code>+DD64 /opt/jplbase/lib/libJevApiAst64.a</code>
HP-UX (IPF) (64-bit multi-threaded)	<code>+DD64 -mt /opt/jplbase/lib/libJevApiAmt64.a</code>
Solaris (32-bit single-threaded)	<code>/opt/jplbase/lib/libJevApiAst.a -ldl</code>
Solaris (32-bit multi-threaded)	<code>/opt/jplbase/lib/libJevApiAmt.a -ldl -lpthread</code>
AIX (32-bit single-threaded)	<code>/opt/jplbase/lib/libJevApiAst.a -ldl</code>
AIX (32-bit multi-threaded)	<code>/opt/jplbase/lib/libJevApiAmt.a -ldl -lpthread</code>
Linux 5 (x86) Linux 6 (x86) (32-bit single-threaded)	<code>/opt/jplbase/lib/libJevApiAst.a -ldl</code>
Linux 5 (x86) Linux 6 (x86) (32-bit multi-threaded)	<code>/opt/jplbase/lib/libJevApiAmt.a -ldl -lpthread</code>

*Notes:*

- The libraries provided by JP1/Base are static libraries (or an archive for UNIX). They are not DLL import libraries or shared libraries.
- The libraries provided by JP1/Base are dynamically loaded by means of a DLL (or a shared library for UNIX) bundled with JP1/Base. This means that created programs will run on a machine on which JP1/Base is not installed, but certain functions will fail with a `JEV_NO_LIBRARY` error.
- DLLs dynamically loaded from the libraries provided by JP1/Base for Windows are independent of the libraries packaged with side-by-side assembly, so no manifest is provided.
- Do not use the `-l` option when linking the libraries provided by JP1/Base on UNIX.
- When linking on UNIX, we recommend that you use the same linkage editor you used for the compiling (`cc`, for example), rather than using `ld`. If you do use `ld` to link the files, specify the same options in the same order as the compiler when it automatically passes to `ld`.
- To compile source files in the Linux x64 environment, add `-m32` to the compile options and link options.

## 2.3 Migrating user applications from an earlier version

This section explains the procedure for migrating user applications created in an earlier version of JP1/Base to the current version.

### 2.3.1 Migrating without recompiling

JP1/Base assures binary compatibility with user applications created in earlier versions of JP1/Base. Therefore, previously created user applications can run on the most recent version of JP1/Base without having to be recompiled.

Binary compatibility of the user applications is assured, provided the JP1/Base execution environment version is the same or later than the JP1/Base development environment version. Therefore, for a user application that is executed by several different versions of JP1/Base, use a JP1/Base development environment version that is no later than the earliest version of the JP1/Base execution environment that is being used.

The following table lists examples of the ranges of binary compatibility that are assured between the development environment and the execution environment of JP1/Base.

Table 2–7: Examples of the ranges of binary compatibility that are assured

Development environment	Execution environment	Binary compatibility
If a user application was created in the following development environment: <ul style="list-style-type: none"><li>• JP1/Base 07-51</li><li>• Compiler</li><li>• User application</li></ul>	If JP1/Base is a later version: <ul style="list-style-type: none"><li>• JP1/Base 08-00 or later</li></ul>	Y
	If JP1/Base is the same version: <ul style="list-style-type: none"><li>• JP1/Base 07-51</li></ul>	Y
	If JP1/Base is an earlier version: <ul style="list-style-type: none"><li>• JP1/Base 07-50 or earlier</li></ul>	N

Legend:

Y: Assured.

N: Not assured.

#### Important note

The above table does not consider which OS versions are assured for running user applications generated by the compiler that is being used. For example, a user application generated with a compiler supported in JP1/Base 7-51 might not be able to run on an OS for which support was added in JP1/Base 08-00. For details about which OS versions are assured for running user applications generated with a particular compiler, see the compiler documentation.

### 2.3.2 Recompiling before migrating

JP1/Base assures compatibility of source code created in earlier versions. Therefore, by recompiling the source code of the user application, you can run recompiled user applications on the latest version of JP1/Base without having to modify the source code.

# 3

## Functions

This chapter describes the functions for issuing and acquiring JP1 events.

## Function description format

---

This section lists the headings that are used in this chapter for the descriptions of the JP1 event issuing function and the JP1 event acquisition functions.

### Description

Describes the functionality of the function.

### Definition header

Indicates the header for defining the function.

### Format

Shows the format of the function.

### Argument(s)

Describes the arguments and values that can be specified in the function.

### Return value(s)

Describes the values that the function may return after its execution.

### Note(s)

Gives the points you must remember when you use the function. For the notes common to all functions, see the next section *Notes common to all functions*.

## Notes common to all functions

---

The notes common to all functions provided by JP1/Base are as follows.

- For the Windows and UNIX versions, the operation of a function called from a multi-thread program is assured.
- For the Windows and UNIX versions, when using the functions in a multi-thread program, you cannot use any of these functions in a thread that was generated before the first function was called.

## List of functions

JP1/Base provides a JP1 event issuing function and the JP1/event acquisition functions described below. The following two tables list and explain these functions. For details about the event attributes of the JP1 events used by the JP1 event issuing function and the JP1 event acquisition functions, see *A. Criteria for Setting JP1 Event Attributes*.

Table 3–1: JP1 event issuing function

Function name	Explanation
JevRegistEvent	Issues a JP1 event to the event server of JP1/Base.

Table 3–2: JP1 event acquisition functions

Function name	Explanation
JevGetOpen	Connects the program to the event server of JP1/Base so that the program can acquire JP1 events.
JevGetEvent	Acquires a JP1 event.
JevGetSequenceNumber	Acquires a basic attribute of the JP1 event (the serial number of the JP1 event in the event database).
JevGetBaseID	Acquires a basic attribute of the JP1 event (the base part of the event ID).
JevGetExtID	Acquires a basic attribute of the JP1 event (the extended part of the event ID).
JevGetRegistFactor	Acquires a basic attribute of the JP1 event (registered reason).
JevGetProcessID	Acquires a basic attribute of the JP1 event (source process ID).
JevGetRegistTime	Acquires a basic attribute of the JP1 event (registered time).
JevGetArrivedTime	Acquires a basic attribute of the JP1 event (arrived time).
JevGetRegistUserID	Acquires a basic attribute of the JP1 event (source user ID).
JevGetRegistGroupID	Acquires a basic attribute of the JP1 event (source group ID).
JevGetRegistUserName	Acquires a basic attribute of the JP1 event (source user name).
JevGetRegistGroupName	Acquires a basic attribute of the JP1 event (source group name).
JevGetSourceServer	Acquires a basic attribute of the JP1 event (source event server name).
JevGetDestinationServer	Acquires a basic attribute of the JP1 event (destination event server name).
JevGetSourceAddress	Acquires a basic attribute of the JP1 event (source IP address).
JevGetDestinationAddress	Acquires a basic attribute of the JP1 event (destination IP address).
JevGetSourceSequenceNumber	Acquires a basic attribute of the JP1 event (source serial number).
JevGetCodeSet	Acquires a basic attribute of the JP1 event (code set).
JevGetMessage	Acquires a basic attribute of the JP1 event (message).
JevGetDetailInformation	Acquires a basic attribute of the JP1 event (detailed information).
JevGetExtAttrDirect	Acquires an extended attribute of the JP1 event.
JevGetFirstExtAttr	Acquires the first extended attribute of the JP1 event.
JevGetNextExtAttr	Acquires the next extended attribute of the JP1 event.
JevFreeEvent	Deallocates the memory for an acquired JP1 event.

Function name	Explanation
JevGetClose	Disconnects the program from the event server.

The following sections describe the above functions in alphabetical order.

# JevFreeEvent

---

## Description

This function deallocates the area containing a JP1 event that you can access by using the return value of a `JevGetEvent ()` function.

## Definition header

`JevApi.h`

## Format

```
int JevFreeEvent(long* lpIStatus,
                JP1EVENT event);
```

## Arguments

### *lpIStatus*

In *lpIStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–3: Status code and meaning (JevFreeEvent)

Status code	Meaning
<code>JEV_S_PARAM_ERROR</code>	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent ()` function).

## Return values

Situation	Explanation
Normal termination	The function returns 0.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lpIStatus</i> , the function stores the detailed error code.

# JevGetArrivedTime

---

## Description

This function acquires the time when the JP1 event arrived, as a basic attribute of the JP1 event. The time is represented by the number of seconds from 1970-01-01 00:00:00 (UTC).

## Definition header

JevApi.h

## Format

```
long JevGetArrivedTime(long* lpIStatus,  
                        JP1EVENT event);
```

## Arguments

### *lpIStatus*

In *lpIStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–4: Status code and meaning (JevGetArrivedTime)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

## Return values

Situation	Explanation
Normal termination	The function returns the arrival time of the JP1 event that can be referenced with the specified handle.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lpIStatus</i> , the function stores the detailed error code.

# JevGetBaseID

---

## Description

This function acquires the base part of the event ID, as a basic attribute of the JP1 event. The base part of the event ID is the first four bytes of the eight-byte event ID.

## Definition header

JevApi.h

## Format

```
long JevGetBaseID(long* lpIStatus,  
                 JP1EVENT event);
```

## Arguments

### *lpIStatus*

In *lpIStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–5: Status code and meaning (JevGetBaseID)

Situation	Explanation
JEV_S_PARAM_ERROR	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

## Return values

Situation	Explanation
Normal termination	The function returns the base part of the ID of the JP1 event that can be referenced with the specified handle.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lpIStatus</i> , the function stores the detailed error code.

# JevGetClose

---

## Description

This function disconnects the program from the event server and closes the JP1-event acquisition handle returned by the `JevGetOpen()` function.

The JP1-event acquisition handle returned by the `JevGetOpen()` function must be closed by using the `JevGetClose()` function. In Windows, if the process terminates without calling this function, a system-resource leak error occurs.

## Definition header

`JevApi.h`

## Format

```
int JevGetClose(long* lpIStatus,  
                JEVGETKEY key);
```

## Arguments

### *lpIStatus*

In *lpIStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status codes that may be returned.

Table 3–6: Status codes and meanings (JevGetClose)

Status code	Meaning
<code>JEV_S_CONNECT_ERROR</code>	Failed to connect the event server.
<code>JEV_S_PARAM_ERROR</code>	An invalid parameter is specified.
<code>JEV_S_MAXOPEN</code>	The number of opened files reached the maximum.
<code>JEV_S_NOMEMORY</code>	Memory is insufficient.
<code>JEV_S_IO_ERR</code>	An I/O error occurred.
<code>JEV_S_SYSTEM_ERROR</code>	A system error occurred (the system resource became insufficient).

### *key*

In *key*, specify the handle for acquiring the target JP1 event (returned by the `JevGetOpen()` function).

## Return values

Situation	Explanation
Normal termination	The function returns 0.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lpIStatus</i> , the function stores the detailed error code.

# JevGetCodeSet

---

## Description

This function acquires the code set as a basic attribute of the JP1 event.

## Definition header

JevApi.h

## Format

```
int JevGetCodeSet(long* lpIStatus,
                  JP1EVENT event,
                  const char** const lppszValue);
```

## Arguments

### *lpIStatus*

In *lpIStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–7: Status code and meaning (JevGetCodeSet)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

### *lppszValue*

In *lppszValue*, specify the pointer to the area for storing the pointer to the acquired code set. When the corresponding data does not exist, a NULL pointer is set.

## Return values

Situation	Explanation
Normal termination	The function returns 0. Also, in the area specified in <i>lppszValue</i> , the function stores the pointer to the code set.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lpIStatus</i> , the function stores the detailed error code.

# JevGetDestinationAddress

---

## Description

This function acquires the destination IP address as a basic attribute of the JP1 event.

## Definition header

JevApi.h

## Format

```
int JevGetDestinationAddress(long* lplStatus,
                             JP1EVENT event,
                             int* lpnSize,
                             const char** const lppsValue);
```

## Arguments

### *lplStatus*

In *lplStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–8: Status code and meaning (JevGetDestinationAddress)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

### *lpnSize*

In *lpnSize*, specify the pointer to the area for storing the length of the destination IP address. For JP1 events acquired in an IPv6 environment, the length of the destination IP address must be 16.

### *lppsValue*

In *lppsValue*, specify the pointer to the area for storing the pointer to the acquired destination IP address.

## Return values

Situation	Explanation
Normal termination	The function returns 0 and, in the area specified in <i>lppsValue</i> , stores the pointer to the destination IP address. Also, in the area specified in <i>lpnSize</i> , the function stores the size of the destination IP address.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lplStatus</i> , the function stores the detailed error code.

# JevGetDestinationServer

---

## Description

This function acquires the destination event server name as a basic attribute of the JP1 event.

## Definition header

JevApi.h

## Format

```
int JevGetDestinationServer(long* lpIStatus,
                           JP1EVENT event,
                           const char** lppsValue);
```

## Arguments

### *lpIStatus*

In *lpIStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–9: Status code and meaning (JevGetDestinationServer)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

### *lppsValue*

In *lppsValue*, specify the pointer to the area for storing the pointer to the acquired destination event server name. When the corresponding data does not exist, a NULL pointer is set.

## Return values

Situation	Explanation
Normal termination	The function returns 0. Also, in the area specified in <i>lppsValue</i> , the function stores the pointer to the destination event server name.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lpIStatus</i> , the function stores the detailed error code.

# JevGetDetailInformation

---

## Description

This function acquires the detailed information of the JP1 event as a basic attribute of the JP1 event.

## Definition header

JevApi.h

## Format

```
int JevGetDetailInformation(long* lpIStatus,
                           JP1EVENT event,
                           long* lpISize,
                           const char** const lppszValue);
```

## Arguments

### *lpIStatus*

In *lpIStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–10: Status code and meaning (JevGetDetailInformation)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

### *lpISize*

In *lpISize*, specify the pointer to the area for storing the length of the detailed information.

### *lppszValue*

In *lppszValue*, specify the pointer to the area for storing the pointer to the acquired detailed information. When the corresponding data does not exist, a NULL pointer is set.

## Return values

Situation	Explanation
Normal termination	The function returns 0 and, in the area specified in <i>lppszValue</i> , stores the pointer to the detailed information. Also, in the area specified in <i>lpISize</i> , the function stores the length of the detailed information.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lpIStatus</i> , the function stores the detailed error code.

# JevGetEvent

---

## Description

This function acquires a JP1 event that matches the condition specified in the `JevGetOpen()` function. You can call this function any number of times to acquire the JP1 events that satisfy the filter condition specified in the `JevGetOpen()` function in the order in which the events were registered with the event database.

## Definition header

`JevApi.h`

## Format

```
JP1EVENT JevGetEvent(long* lplStatus,  
                    JEVGETKEY key,  
                    JEVACCESSTYPE access);
```

## Arguments

### *lplStatus*

In *lplStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status codes that may be returned.

Table 3–11: Status code and meaning (JevGetEvent)

Status code	Meaning
<code>JEV_S_CONNECT_ERROR</code>	Failed to connect the event server.
<code>JEV_S_INVALID_SERVER</code>	The event server name is invalid.
<code>JEV_S_PARAM_ERROR</code>	An invalid parameter is specified.
<code>JEV_S_NO_EVENT</code>	No JP1 events satisfy the filter condition.
<code>JEV_S_MAXOPEN</code>	The number of opened files reached the maximum.
<code>JEV_S_NOMEMORY</code>	Memory is insufficient.
<code>JEV_S_IO_ERR</code>	An I/O error occurred.

### **key**

In *key*, specify the handle for acquiring the target JP1 event (returned by the `JevGetOpen()` function).

### **access**

In *access*, specify either of the following values for specifying the action to be taken if no JP1 events satisfy the condition specified for acquiring JP1 events.

`JEVGET_WAIT`

Does not return the control until the corresponding JP1 event occurs.

`JEVGET_NOWAIT`

Returns an error immediately if the corresponding JP1 event is not found.

## Return values

Situation	Explanation
Normal termination	The function returns the handle for accessing the JP1 event.
Abnormal termination	The function returns a null pointer. Also, in the area specified in <i>lpStatus</i> , the function stores the detailed error code.

# JevGetExtAttrDirect

---

## Description

This function acquires an extended attribute of the JP1 event.

## Definition header

JevApi.h

## Format

```
const char*JevGetExtAttrDirect(long* lplStatus,
                               JP1EVENT event,
                               const char* lpszName);
```

## Arguments

### *lplStatus*

In *lplStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status codes that may be returned.

Table 3–12: Status codes and meanings (JevGetExtAttrDirect)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.
JEV_S_NOT_DEFINED	The specified attribute is not defined.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

### *lpszName*

In *lpszName*, specify the pointer to the character string that specifies the extended attribute name.

## Return values

Situation	Explanation
Normal termination	The function returns the extended attribute value of the JP1 event that can be referenced with the specified handle.
Abnormal termination	The function returns a null pointer. Also, in the area specified in <i>lplStatus</i> , the function stores the detailed error code.

# JevGetExtID

---

## Description

This function acquires the extended part of the event ID, as a basic attribute of the JP1 event. The extended part of the event ID is the last four bytes of the eight-byte event ID.

## Definition header

JevApi.h

## Format

```
long JevGetExtID(long* lpIStatus,  
                JP1EVENT event);
```

## Arguments

### *lpIStatus*

In *lpIStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–13: Status code and meaning (JevGetExtID)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

## Return values

Situation	Explanation
Normal termination	The function returns the extended part of the ID of the JP1 event that can be referenced with the specified handle.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lpIStatus</i> , the function stores the detailed error code.

# JevGetFirstExtAttr

---

## Description

This function acquires the first extended attribute specified in the JP1 event.

## Definition header

JevApi.h

## Format

```
int JevGetFirstExtAttr(long* lplStatus,
                      JP1EVENT event,
                      const char** const lppszName,
                      const char** const lppszValue);
```

## Arguments

### *lplStatus*

In *lplStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status codes that may be returned.

Table 3–14: Status codes and meanings (JevGetFirstExtAttr)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.
JEV_S_EXTATTR_EOD	This JP1 event includes no more extended attributes.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

### *lppszName*

In *lppszName*, specify the pointer to the area for storing the pointer to the acquired extended attribute name.

### *lppszValue*

In *lppszValue*, specify the pointer to the area for storing the pointer to the acquired extended attribute value. When the corresponding data does not exist, a NULL pointer is set.

## Return values

Situation	Explanation
Normal termination	The function returns 0 and, in the area specified in <i>lppszName</i> , stores the pointer to the extended attribute name. Also, in the area specified in <i>lppszValue</i> , the function stores the pointer to the extended attribute value.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lplStatus</i> , the function stores the detailed error code.

# JevGetMessage

---

## Description

This function acquires the message in the JP1 event as a basic attribute of the JP1 event.

## Definition header

JevApi.h

## Format

```
int JevGetMessage(long* lpIStatus,  
                 JP1EVENT event,  
                 const char** const lppszValue);
```

## Arguments

### *lpIStatus*

In *lpIStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–15: Status code and meaning (JevGetMessage)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

### *lppszValue*

In *lppszValue*, specify the pointer to the area for storing the pointer to the acquired message. When the corresponding data does not exist, a NULL pointer is set.

## Return values

Situation	Explanation
Normal termination	The function returns 0. Also, in the area specified in <i>lppszValue</i> , the function stores the pointer to the message.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lpIStatus</i> , the function stores the detailed error code.

# JevGetNextExtAttr

---

## Description

This function acquires the next specified extended attribute of a JPI event after an extended attribute is acquired by a JPI `JevGetFirstAttr()` or a `JevGetNextAttr()` event acquisition function.

## Definition header

`JevApi.h`

## Format

```
int JevGetNextExtAttr(long* lpIStatus,
                    JPIEVENT event,
                    const char** const lppsName,
                    const char** const lppsValue);
```

## Arguments

### *lpIStatus*

In *lpIStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status codes that may be returned.

Table 3–16: Status codes and meanings (`JevGetNextExtAttr`)

Status code	Meaning
<code>JEV_S_PARAM_ERROR</code>	An invalid parameter is specified.
<code>JEV_S_EXTATTR_EOD</code>	This JPI event includes no more extended attributes.

### *event*

In *event*, specify the handle for accessing the target JPI event (the return value of the `JevGetEvent()` function).

### *lppsName*

In *lppsName*, specify the pointer to the area for storing the pointer to the acquired extended attribute name.

### *lppsValue*

In *lppsValue*, specify the pointer to the area for storing the pointer to the acquired extended attribute value.

## Return values

Situation	Explanation
Normal termination	The function returns 0 and, in the area specified in <i>lppsName</i> , stores the pointer to the next extended attribute name. Also, in the area specified in <i>lppsValue</i> , the function stores the pointer to the next extended attribute value.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lpIStatus</i> , the function stores the detailed error code.

# JevGetOpen

---

## Description

This function connects the program to the event server of JP1/Base so that the program can acquire JP1 events.

## Definition header

JevApi.h

## Format

```
JEVGETKEY JevGetOpen(long* lplStatus,  
                    const char* lpszServer,  
                    const char* lpszFilter,  
                    long lPosition);
```

## Arguments

### *lplStatus*

In *lplStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status codes that may be returned.

Table 3–17: Status codes and meanings (JevGetOpen)

Status code	Meaning
JEV_NO_LIBRARY	No library is found. <sup>#1</sup> Alternatively, the shared library cannot be found because too many files are open.
JEV_S_CONNECT_ERROR	Failed to connect the event server.
JEV_S_PARAM_ERROR	An invalid parameter is specified.
JEV_S_MAXOPEN	The number of open files reached the maximum.
JEV_S_NOMEMORY	Memory is insufficient.
JEV_S_IO_ERR	An I/O error occurred.
JEV_S_SYSTEM_ERROR	A system error occurred (system resources became insufficient).
JEV_S_NO_AUTHORITY	The JP1 program or user application does not have sufficient authority to connect the event server. <sup>#2</sup>
JEV_S_FILTER_ERROR	The filter contains an error (excluding errors in regular expressions).
JEV_S_REGEX_ERROR	The regular expression specified in the filter contains an error.
JEV_S_REGEX_CANNOT_USED	The regular expression library cannot be used.

<sup>#1</sup>: Check if necessary files have been deleted or if incorrect compile options are specified. If necessary files have been deleted, reinstall JP1/Base. If compile options are incorrect, reconfigure the option settings.

<sup>#2</sup>: The `users` parameter in the event server settings (`conf`) file for JP1/Base defines the authority to connect the event server.

### ***lpszServer***

In *lpszServer*, specify the pointer to a character string that indicates a destination event server name and ends with \0. If you specify a null pointer, the function connects the program to the event server that has the same name as the local host name. Specify an event server name of 256 bytes or less, including the \0.

### ***lpszFilter***

In *lpszFilter*, specify the pointer to a character string ending in \0 that indicates a filter, as described by the filter syntax section in the manual *Job Management Partner 1/Base User's Guide*. If you specify a null pointer, the function targets all the JP1 events.

### ***lPosition***

In *lPosition*, specify a serial number in the event database as the position from which to start acquiring JP1 events.

If you specify -1, the function can acquire the JP1 events registered after the issuance of this function. Note that events that occur during execution of this function might not be acquired. Therefore, acquisition is guaranteed for JP1 events that are registered after the completion of this function.

## **Return values**

Situation	Explanation
Normal termination	This function returns the handle for acquiring the JP1 event.
Abnormal termination	The function returns a null pointer. Also, in the area specified in <i>lpIStatus</i> , the function stores the detailed error code.

# JevGetProcessID

---

## Description

This function acquires the source process ID as a basic attribute of the JP1 event.

## Definition header

JevApi.h

## Format

```
long JevGetProcessID(long* lplStatus,
                    JP1EVENT event);
```

## Arguments

### *lplStatus*

In *lplStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–18: Status code and meaning (JevGetProcessID)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

## Return values

Situation	Explanation
Normal termination	The function returns the JP1 event-originating process ID that can be referenced with the specified handle.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lplStatus</i> , the function stores the detailed error code.

# JevGetRegistFactor

---

## Description

This function acquires the registration type of the JP1 event, as a basic attribute of the JP1 event.

## Definition header

JevApi.h

## Format

```
int JevGetRegistFactor(long* lpIStatus,
                      JP1EVENT event);
```

## Arguments

### *lpIStatus*

In *lpIStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–19: Status code and meaning (JevGetRegistFactor)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

## Return values

Situation	Explanation
Normal termination	The function returns the JP1-event registration type that can be referenced with the specified handle.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lpIStatus</i> , the function stores the detailed error code.

# JevGetRegistGroupID

---

## Description

This function acquires the source group ID as a basic attribute of the JP1 event.

## Definition header

JevApi.h

## Format

```
int JevGetRegistGroupID(long* lpIStatus,  
                        JP1EVENT event,  
                        long* lpISize);
```

## Arguments

### *lpIStatus*

In *lpIStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–20: Status code and meaning (JevGetRegistGroupID)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

### *lpISize*

In *lpISize*, specify the pointer to the area for containing the event-originating group ID.

## Return values

Situation	Explanation
Normal termination	The function returns 0. Also, in the area specified in <i>lpISize</i> , the function stores the event-originating group ID.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lpIStatus</i> , the function stores the detailed error code.

# JevGetRegistGroupName

---

## Description

This function acquires the source group name as a basic attribute of the JP1 event.

## Definition header

JevApi.h

## Format

```
int JevGetRegistGroupName(long* lpIStatus,  
                          JP1EVENT event,  
                          const char** const lppszValue);
```

## Arguments

### *lpIStatus*

In *lpIStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–21: Status code and meaning (JevGetRegistGroupName)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

### *lppszValue*

In *lppszValue*, specify the pointer to the area for storing the pointer to the acquired event-originating group name. When the corresponding data does not exist, a NULL pointer is set.

## Return values

Situation	Explanation
Normal termination	The function returns 0. Also, in the area specified in <i>lppszValue</i> , the function stores the pointer to the event-originating group name.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lpIStatus</i> , the function stores the detailed error code.

# JevGetRegistTime

---

## Description

This function acquires the time when the JP1 event was registered, as a basic attribute of the JP1 event. The time is represented by the number of seconds from 1970-01-01 00:00:00 (UTC).

## Definition header

JevApi.h

## Format

```
long JevGetRegistTime(long* lpIStatus,  
                      JP1EVENT event);
```

## Arguments

### *lpIStatus*

In *lpIStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–22: Status code and meaning (JevGetRegistTime)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

## Return values

Situation	Explanation
Normal termination	The function returns the registration time of the JP1 event that can be referenced with the specified handle.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lpIStatus</i> , the function stores the detailed error code.

# JevGetRegistUserID

---

## Description

This function acquires the source user ID as a basic attribute of the JP1 event.

## Definition header

JevApi.h

## Format

```
int JevGetRegistUserID(long* lplStatus,  
                      JP1EVENT event,  
                      long* lplSize);
```

## Arguments

### *lplStatus*

In *lplStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–23: Status code and meaning (JevGetRegistUserID)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

### *lplSize*

In *lplSize*, specify the pointer to the area for storing the event-originating user ID.

## Return values

Situation	Explanation
Normal termination	The function returns 0. Also, in the area specified in <i>lplSize</i> , the function stores the event-originating user ID.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lplStatus</i> , the function stores the detailed error code.

# JevGetRegistUserName

---

## Description

This function acquires the source user name as a basic attribute of the JP1 event.

## Definition header

JevApi.h

## Format

```
int JevGetRegistUserName(long* lpIStatus,  
                        JP1EVENT event,  
                        const char** const lppszValue);
```

## Arguments

### *lpIStatus*

In *lpIStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–24: Status code and meaning (JevGetRegistUserName)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

### *lppszValue*

In *lppszValue*, specify the pointer to the area for storing the pointer to the acquired event-originating user name. When the corresponding data does not exist, a NULL pointer is set.

## Return values

Situation	Explanation
Normal termination	The function returns 0. Also, in the area specified in <i>lppszValue</i> , the function stores the pointer to the event-originating user name.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lpIStatus</i> , the function stores the detailed error code.

# JevGetSequenceNumber

---

## Description

The function acquires the serial number in the event database, as a basic attribute of the JP1 event.

## Definition header

JevApi.h

## Format

```
long JevGetSequenceNumber(long* lpIStatus,  
                           JP1EVENT event);
```

## Arguments

### *lpIStatus*

In *lpIStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–25: Status code and meaning (JevGetSequenceNumber)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

## Return values

Situation	Explanation
Normal termination	The function returns the serial number of the JP1 event in the event database that can be referenced with the specified handle.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lpIStatus</i> , the function stores the detailed error code.

# JevGetSourceAddress

---

## Description

This function acquires the source IP address as a basic attribute of the JP1 event.

## Definition header

JevApi.h

## Format

```
int JevGetSourceAddress(long* lplStatus,
                       JP1EVENT event,
                       int* lpnSize,
                       const char** lppsValue);
```

## Arguments

### *lplStatus*

In *lplStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–26: Status code and meaning (JevGetSourceAddress)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

### *lpnSize*

In *lpnSize*, specify the pointer to the area for storing the length of the event-originating IP address. For JP1 events acquired in an IPv6 environment, the length of the destination IP address must be 16.

### *lppsValue*

In *lppsValue*, specify the pointer to the area for storing the pointer to the acquired event-originating IP address.

## Return values

Situation	Explanation
Normal termination	The function returns 0 and, in the area specified in <i>lppsValue</i> , stores the pointer to the event-originating IP address. Also, in the area specified in <i>lpnSize</i> , the function stores the length of the event-originating IP address.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lplStatus</i> , the function stores the detailed error code.

# JevGetSourceSequenceNumber

---

## Description

This function acquires the serial number for each event-originating program, as a basic attribute of the JP1 event.

## Definition header

JevApi.h

## Format

```
long JevGetSourceSequenceNumber(long* lpIStatus,  
                                JP1EVENT event);
```

## Arguments

### *lpIStatus*

In *lpIStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–27: Status code and meaning (JevGetSourceSequenceNumber)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

## Return values

Situation	Explanation
Normal termination	The function returns the JP1-event serial number for each event-originating program that can be referenced with the specified handle.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lpIStatus</i> , the function stores the detailed error code.

# JevGetSourceServer

---

## Description

This function acquires the source event server name as a basic attribute of the JP1 event.

## Definition header

JevApi.h

## Format

```
int JevGetSourceServer(long* lplStatus,
                      JP1EVENT event,
                      const char** const lppsValue);
```

## Arguments

### *lplStatus*

In *lplStatus*, specify the pointer to the area for containing the status code that this function returns if the function abnormally terminates. The following explains the status code.

Table 3–28: Status code and meaning (JevGetSourceServer)

Status code	Meaning
JEV_S_PARAM_ERROR	An invalid parameter is specified.

### *event*

In *event*, specify the handle for accessing the target JP1 event (the return value of the `JevGetEvent()` function).

### *lppsValue*

In *lppsValue*, specify the pointer to the area for storing the pointer to the acquired event-originating event server name.

## Return values

Situation	Explanation
Normal termination	The function returns 0. Also, in the area specified in <i>lppsValue</i> , the function stores the pointer to the event-originating event server name.
Abnormal termination	The function returns -1. Also, in the area specified in <i>lplStatus</i> , the function stores the detailed error code.

# JevRegistEvent

---

## Description

This function issues a JP1 event to the JP1/Base event server. Normal termination of this function assures that the local event server has successfully accepted the JP1 event.

## Definition header

JevApi.h

## Format

```
int JevRegistEvent(long* status,
                  const char* server,
                  long baseID,
                  const char* message,
                  const char** extattrs,
                  int extcount);
```

## Arguments

### *status*

In *status*, specify the address of the area for storing the status code returned if this function terminates abnormally. The following explains the status codes that may be returned.

Table 3–29: Status codes and meanings (JevRegistEvent)

Status code	Meaning
JEV_NO_LIBRARY	No library is found. <sup>#</sup> Alternatively, the shared library cannot be found because too many files are opened.
JEV_S_CONNECT_ERROR	Failed to connect the event service.
JEV_S_INVALID_ID	The event ID is invalid.
JEV_S_INVALID_SERVER	The event server name is invalid.
JEV_S_INVALID_EXT_NAME	An extended attribute name is invalid.
JEV_S_OVER_EXT_COUNT	The number of extended attributes exceeds the maximum.
JEV_S_OVER_EXT_SIZE	The total size of extended attributes exceeds the maximum.
JEV_S_OVER_MESSAGE	The message length exceeds the maximum.
JEV_S_PARAM_ERROR	An invalid parameter is specified.
JEV_S_NOT_SUPPORT	The version is not supported.
JEV_S_MAXOPEN	The number of opened files reached the maximum.
JEV_S_NOMEMORY	Memory is insufficient.
JEV_S_IO_ERR	An I/O error occurred.
JEV_S_SYSTEM_ERROR	A system error occurred.

<sup>#</sup>: Check if necessary files have been deleted or if incorrect compile options are specified. If necessary files have been deleted, reinstall JP1/Base. If compile options are incorrect, reconfigure the option settings.

## **server**

In *server*, specify a pointer to a character string that ends with `\0` and indicates the name of the destination event server running on the local host. If you specify a NULL pointer, this function attempts to connect the event server that has the same name as the local host. Specify an event server name of 256 bytes or less, including the `\0`.

## **baseID**

In *baseID*, specify a numeric value that indicates the basic part of the event ID you want to register. You can specify one of the following values:

- 0x00000000
- 0x00000001 to 0x00001FFF
- 0x7FFF8000 to 0x7FFFFFFF

## **message**

In *server*, specify a pointer to a character string that ends with `\0` and indicates the message you want to register. Specify a message of 1,024 bytes or less, including the `\0`.

## **extattrs**

In *extattrs*, specify a string array containing extended-attribute strings. Each extended-attribute string in the array has the *extended-attribute-name=extended-attribute-value* format and ends with `\0`.

*extended-attribute-name* is a character string that indicates the meaning of the attribute. You can use up to 32 alphanumeric characters including underscores (`_`) to specify *extended-attribute-name*. For alphabetic characters, you can use upper-case characters only. The specified character string must begin with an alphabetic character.

*extended-attribute-value* is a character string containing the value of the attribute. The character string can have 0 to 10,000 bytes.

You can specify up to 100 extended attributes. The maximum number of bytes used in all attribute values is 10,000 bytes.

If you specify a NULL pointer as an argument, extended attributes are not registered.

For details on extended attributes, see [A. Criteria for Setting JPI Event Attributes](#).

## **extcount**

In *extcount*, specify the number of extended attributes you want to register. This value is ignored if a NULL pointer is specified in *extattrs*.

## **Return values**

Situation	Explanation
Normal termination	The function returns 0.
Abnormal termination	The function returns -1.

## **Note**

If you specify character strings containing the same extended attribute name, the last character string takes effect.



# Appendixes

## A. Criteria for Setting JP1 Event Attributes

When you issue a user-defined event, you can set event attributes based on the criteria described in the following sections. You can also reference the criteria described in the following sections to determine whether to acquire those attributes when acquiring JP1 events.

### A.1 Basic attributes

Note that the event ID and the message are the basic attributes used when user-defined events are issued. Use the other basic attributes for acquiring JP1 events or other purposes.

Table A–1: Basic attributes of JP1 events

Attribute	Explanation
Serial number	The serial number of the JP1 events that arrived at the event server (including local events). Serial numbers are assigned regardless of the event-originating applications. This attribute is not stored when the JP1 event is transferred between event servers. The major purpose of this attribute is to ensure that the JP1 event will not be lost or duplicated when a user application acquires the event or the event is transferred to another event server.
Event ID	An eight-byte value indicating the event-originating application or the event that occurred in the application. Hitachi programs and user programs are allocated specific ranges of event IDs that they can use. The ranges of values that can be specified for user applications are from 0 to 0x1FFF and from 0x7FFF8000 to 0x7FFFFFFF. Each event ID must be unique within the entire system. The first four bytes of the event ID is the basic part and the last bytes of the event ID is the extended part.
Registered reason	The registration type of the JP1 event registered with the event server. This attribute is not stored when the JP1 event is transferred between event servers. There are the following registration types: 1 The event was issued from the local event server to the local event server. 2 The event is issued from the local event server to the remote event server. 3 The event was issued from the remote event server to the local event server. 4 The event was transferred from the remote event server to the local event server on the basis of the configuration settings.
Source process ID	The process ID of the event-originating application program.
Registered time	The time when the event was registered at the event-originating event server. The time is based on the clock of the event-originating host and represented with the number of seconds from 1970-01-01 00:00:00 (UTC).
Arrived time	The time when the event was registered at the local event server. The time is represented with the number of seconds from 1970-01-01 00:00:00 (UTC). This attribute is not stored when the JP1 event is transferred between event servers.
Source user ID	The user ID of the event-originating process. In Windows and Java, this user ID is specified in the environment setting as a fixed value (-1 to 65,535).
Source group ID	The group ID of the event-originating process. In Windows and Java, this group ID is specified in the environment setting as a fixed value (-1 to 65,535).
Source user name	The user name of the event-originating process.
Source group name	The group name of the event-originating process. In Windows and Java, this group name is a null string.

Attribute	Explanation
Source event server name	The event server name of the event-originating application. This attribute indicates the event server name of the host where the JP1 event originated, even when the JP1 event has been transferred.
Destination event server name	This attributes indicates the name of the event server to which the JP1 event will be transferred when the transfer to the event server is explicitly specified by the event-originating application.
Source IP address	The IP address for the event-originating event server. Note that if the JP1 event passes through NAT or a proxy, or if the JP1 event is transferred by the environment setting, this IP address is not always correct.
Destination IP address	The IP address for the destination event server. Note that if the JP1 event passes through NAT or a proxy, or if the JP1 event is transferred by the environment setting, this IP address is not always correct.
Source serial number	The serial number of the JP1 event in the event database at the event-originating host. This serial number does not change when the JP1 event is transferred.
Code set	The name of the code set with which the messages, detailed information, and extended attributes are written.
Message	The message should be: <ul style="list-style-type: none"> <li>• A clear explanation of the event</li> <li>• Written in one line without a new line code</li> </ul>
Detailed information	Any data

## A.2 Extended attributes

Extended attributes of JP1 events are classified into common extended attributes and user-specific extended attributes.

### (1) Common extended attributes

Table A–2: Extended attributes of JP1 events (common information)

Attribute name	Item	Explanation
SEVERITY	Event level	This attribute indicates the severity of the event. The possible event levels are as follows: Emergency, Alert, Critical, Error, Warning, Notice, Information, and Debug. For details about event levels (severity), see <i>Table A-3 Event levels</i> .
USER_NAME	User name	This attribute indicates the name of the user who is executing the job.
PRODUCT_NAME	Product name	This attribute indicates the product name. The value of this attribute consists of alphanumeric strings separated by a slant (/). The value must have either of the following formats and must be unique for each company: <ul style="list-style-type: none"> <li>• /company-name / series-name / product-name</li> <li>• /company-name / product-name</li> </ul> Note also that you cannot use HITACHI, because as a company name it is a reserved word.
OBJECT_TYPE	Object type	This attribute indicates the object type. As the value of this attribute, specify the type of the event-originating object. In the initial status, the object types listed below are provided. You may want to select or search JP1 events by specifying these object types in event filters, so you should assign the same object type to the JP1 events that have the same meaning. If you want to add new object types, create the definition file for object types and specify unique object types in the file. <ul style="list-style-type: none"> <li>• JOB</li> <li>• JOBNET</li> </ul>

Attribute name	Item	Explanation
		<ul style="list-style-type: none"> <li>• ACTION</li> <li>• ACTIONFLOW</li> <li>• PRINTJOB</li> <li>• PRINTQUEUE</li> <li>• PRINTER</li> <li>• BATCHQUEUE</li> <li>• PIPEQUEUE</li> <li>• JOBBOX</li> <li>• LOGFILE</li> <li>• LINK (for reporting events from a lower communication layer)</li> <li>• SERVICE (e.g., daemon process)</li> <li>• PRODUCT (for reporting other program-specific events)</li> <li>• CONFIGURATION</li> <li>• SERVER</li> <li>• BACKUP</li> <li>• RESTORE</li> <li>• MEDIA</li> </ul>
OBJECT_NAME	Object name	This attribute specifies the object name. As the value of this attribute, specify a name that identifies the type of the object. For example, if the object type is JOB, you may assign the job name.
ROOT_OBJECT_TYPE	Root object type	This attribute specifies the root object type. As the value of this attribute, specify the parent object type. This attribute is effective when objects have a hierarchical structure. For example, if the object type is JOB, the root object type is JOBNET. If the root object type does not exist, specify the same value as the object type. In the initial status, the same value as the object type is defined.
ROOT_OBJECT_NAME	Root object name	This attribute specifies the root object name. Specify a name that identifies the root object type. For example, specify a jobnet name.
OBJECT_ID	Object ID	A combination of this attribute and the PRODUCT_NAME attribute specifies a character string that uniquely identifies the object instance in the integrated system. The format of an object ID depends on the other products. This information is used to open the monitor of a product from the Tool Launcher window of JP1/IM.
OCCURRENCE	Occurrence	<p>This attribute specifies an object-specific occurrence that causes the event to occur. In the initial status, the occurrences listed below are provided. You can specify an occurrence and an object type in a filter to select a specific event for a specific object.</p> <p>ACTIVE The object became active.</p> <p>INACTIVE The object became inactive.</p> <p>START The object started.</p> <p>END The object terminated.</p> <p>NOTSTART The object failed to start.</p> <p>CANCEL The object was canceled.</p>

Attribute name	Item	Explanation
		<p>LATESTART Exceeded the scheduled start time.</p> <p>LATEEND Exceeded the scheduled end time.</p> <p>SUBMIT The object was submitted.</p> <p>UNSUBMIT Submitting the object was canceled.</p> <p>ENQUEUE The object was added to the queue.</p> <p>DEQUEUE The object was removed from the queue.</p> <p>PAUSE The object paused.</p> <p>RELEASE The object stopped pausing.</p> <p>RESTART The object restarted.</p> <p>CREATE The object was created.</p> <p>DELETE The object was deleted.</p> <p>MODIFY The object was modified.</p> <p>RETRY The object started a retry.</p> <p>STOP The object stopped.</p> <p>MOVE The object was moved.</p> <p>COPY The object was copied.</p> <p>NOTICE The object notified the operator.</p> <p>REPLY The object received a reply.</p> <p>CONNECT The object was connected.</p> <p>DISCONNECT The object was disconnected.</p> <p>EXCEPTION An exception occurred.</p>
START_TIME	Start time	This attribute indicates the execution start or restart time. As the value of this attribute, specify the number of seconds from 00:00:00 UTC on January 1, 1970. You can specify this attribute only when the OCCURRENCE attribute is START, RESTART, PAUSE, RELEASE, or END.
END_TIME	End time	This attribute indicates the time to end execution. As the value of this attribute, specify the number of seconds from 00:00:00 UTC on January 1,

Attribute name	Item	Explanation
		1970. You can specify this attribute only when the OCCURRENCE attribute is END.
RESULT_CODE	Result code	This attribute indicates a termination code consisting of decimal numbers. You can specify this attribute only when the OCCURRENCE attribute is END.
PLATFORM#	Platform type	This attribute indicates the platform type. Specify a character string for this attribute to specify the platform type in the definition file for extended event attributes or in the definition file for opening monitor windows. If you do not specify this attribute, base is used by default.
ACTION_VERSION#	Version information	This attribute indicates the version used for opening the monitor window. This attribute is necessary when the monitor windows to open differ depending on the version. If you do not specify this attribute, do not specify a version in the definition file for opening monitor windows.

#

The JP1/IM Event Details window displays these attributes only if they have been defined in the definition file for extended event attributes.

## Event levels

The following table explains event levels that may be specified in the SEVERITY common extended attribute. Use the following criteria as a guideline for setting the event level of user-defined events. Note that the JP1/IM Event Console window does not display events that did not have an event level specified when they were issued.

Table A–3: Event levels

Event level	Display name	Explanation
Emergency	<b>Emergency</b>	An emergency status. Normally, the events specifying this event level are broadcast to all users.
Alert	<b>Alert</b>	A status requiring an immediate recovery, such as damage to the system or database.
Critical	<b>Critical</b>	A critical status such as a hardware error.
Error	<b>Error</b>	An error.
Warning	<b>Warning</b>	A warning message.
Notice	<b>Notice</b>	A status that does not indicate an error, but indicates a situation that requires careful handling.
Information	<b>Information</b>	Information to users.
Debug	<b>Debug</b>	A message that normally contains information used only for debugging the program. This event level is not used for JP1 events because an excessive volume of messages might occur.

## (2) User-specific extended attributes

In addition to common extended attributes, you can add user-specific extended attributes for the program to JP1 events. To add user-specific extended attributes, you must define them with JP1/IM in the definition file for extended event attributes.

The following shows the rules for creating user-specific extended attributes:

- You can use a symbolic name having no meaning as an attribute name.
- For programs that have the same value in the PRODUCT\_NAME extended attribute, there must be a one-to-one correspondence between attribute names and meanings.

## B. Sample Source Files

JP1/Base provides the following sample source files written in C:

- `sender.c`
- `receiver.c`

The above two sample files are located in the following directory. Use the samples when required.

Windows: `installation-folder\tools\event\`

UNIX: `/opt/jp1base/tools/event/`

### B.1 Details of the sample source files

This section gives details of the sample source files.

#### (1) Events handled by the sample source files

Table B–1: Events handled by the sample source files

Attribute type	Item	Attribute name	Description
Basic attribute	Event ID	--	0x00000001
	Message	--	Starts the SAMPLE application.
Extended attributes (common attributes)	Event level	SEVERITY	Notice
	User name	USER_NAME	Name of the user who executes the application.
	Product name	PRODUCT_NAME	/COMPANY/APP1/SAMPLE_PRODUCT (product name)
	Object type	OBJECT_TYPE	SAMPLE
	Object name	OBJECT_NAME	SAMPLE_NAME
	Root object type	ROOT_OBJECT_TYPE	ROOT_SAMPLE
	Root object name	ROOT_OBJECT_NAME	ROOT_SAMPLE_NAME
	Object ID	OBJECT_ID	SAMPLE_ID
	Occurrence	OCCURRENCE	START
	Start time	START_TIME	Start time of the SAMPLE application. The number of seconds from 00:00:00 UTC on 1970-01-01.
	Platform	PLATFORM	NT
	Version information	ACTION_VERSION	0600
Extended attributes (user-specific attributes)	SAMPLE common attribute 1	COMMON_ATTR1	NATIVE
	SAMPLE common attribute 2	COMMON_ATTR2	TRUE

Attribute type	Item	Attribute name	Description
	SAMPLE start attributes 1	START_ATTR1	SAMPLE1
	SAMPLE start attributes 2	START_ATTR2	SAMPLE2

## (2) Coding of sample source files

### (a) sender.c code

```

#include <stdio.h>
#include <time.h>
#include "JevApi.h"

int regist_start_event()
{
    int rc; /* Return code */
    long status = 0; /* Detailed error code */
    const char* server; /* Event server name */
    long baseID; /* Event ID */
    const char* message; /* Message */
    char starttime[32];
    const char* extattrs[16]; /* Array for storing extended attributes */

    /* Set the destination event server name. */
    server = NULL;

    /* Set the event ID. */
    baseID = 0x00000001;

    /* Set the message. */
    message = "Starts the SAMPLE application.";

    /* Set the extended attributes. */
    extattrs[0] = "SEVERITY=Notice";
    extattrs[1] = "USER_NAME=SAMPLE_USER";
    extattrs[2] = "PRODUCT_NAME=/COMPANY/APP1/SAMPLE_PRODUCT";
    extattrs[3] = "OBJECT_TYPE=SAMPLE";
    extattrs[4] = "OBJECT_NAME=SAMPLE_NAME";
    extattrs[5] = "OBJECT_ROOT_TYPE=ROOT_SAMPLE";
    extattrs[6] = "OBJECT_ROOT_NAME=ROOT_SAMPLE_NAME";
    extattrs[7] = "OBJECT_ID=SAMPLE_ID";
    extattrs[8] = "OCCURRENCE=START";
    sprintf(starttime, "START_TIME=%ld", time(NULL));
    extattrs[9] = starttime;
    extattrs[10] = "PLATFORM=NT";
    extattrs[11] = "VERSION=0600";
    extattrs[12] = "COMMON_ATTR1=NATIVE";
    extattrs[13] = "COMMON_ATTR2=TRUE";
    extattrs[14] = "START_ATTR1=SAMPLE1";
    extattrs[15] = "START_ATTR2=SAMPLE2";

    /* Register the JP1 event. */
    rc = JevRegistEvent(&status,
                       server,

```

```

        baseID,
        message,
        extattrs,
        16);

if(rc < 0) {
    fprintf(stderr,
            "JevRegistEvent() failed. status = %ld\n",
            status);
    return -1;
}

return 0;
}

int main()
{
    return regist_start_event();
}

```

## (b) receiver.c code

```

#include <stdio.h>
#include <string.h>
#include "JevApi.h"

int get_start_event()
{
    int rc;                /* Return code */
    long position;        /* Sequence number within the event database */
    long status;          /* Status code address */
    char filter[256];     /* Filter statement buffer */
    const char *server;   /* Event server name */
    const char *message; /* Pointer to the message */
    const char *name;     /* Pointer to the extended attribute name */
    const char *value;    /* Pointer to the extended attribute value */
    JEVGETKEY key;        /* Handle for acquiring JPl events */
    JPlEVENT event;       /* Handle for accessing JPl events */
    JEVACCESSTYPE access; /* Action when no JPl event exists */

    /* Set the filter statement to acquire JPl events. */
    strcpy(filter, "B.ID IN 00000001\n");
    strcat(filter, "E.SEVERITY IN Notice\n");
    strcat(filter,
            "E.PRODUCT_NAME IN /COMPANY/APP1/SAMPLE_PRODUCT");

    /* Connect to the event server of the physical host. */
    status = 0;
    /* Event server of the physical host to connect to
    server = NULL; */

    /* Acquisition starts with sequence number 0 within the event database. */
    position = 0;
    key = JevGetOpen(&status, server, filter, position);
    if(key == NULL){
        fprintf(stderr,
                "JevGetOpen() failed. Status = %ld\n",
                status);
        return -1;
    }
}

```

```

    }

/* Acquire all the JPl events which match the filter condition. */
while(1) {
    status = 0;
/* Error return when no JPl event matches the filter condition */
    access = JEVGET_NOWAIT;
    event = JevGetEvent(&status, key, access);
    if(event == NULL){
        if(status == JEV_S_NO_EVENT) {
            /* No more JPl event matches the filter condition. */
            break;
        }
        else {
            /* Error occurred while acquiring JPl events. */
            fprintf(stderr,
                "JevGetEvent() failed. Status = %ld\n",
                status);
            JevGetClose(&status, key);
            return -1;
        }
    }
}

/* Acquire a message. */
status = 0;
rc = JevGetMessage(&status, event, &message);
if(rc < 0){
    fprintf(stderr,
        "JevGetMessage() failed. Status = %ld\n",
        status);
    JevFreeEvent(&status, event);
    JevGetClose(&status, key);
    return -1;
}
else{
    printf("JevGetMessage() message = %s\n", message);
}

/* Acquire the (first) extended attribute. */
status = 0;
rc = JevGetFirstExtAttr(&status, event, &name, &value);
if(rc < 0){
    fprintf(stderr,
        "JevGetFirstExtAttr() failed. Status = %ld\n",
        status);
    JevFreeEvent(&status, event);
    JevGetClose(&status, key);
    return -1;
}
else{
    printf("JevGetFirstExtAttr() name = %s\n", name);
    printf("JevGetFirstExtAttr() value = %s\n", value);
}

/* Acquire the (subsequent) extended attribute. */
while(1) {
    status = 0;
    rc = JevGetNextExtAttr(&status, event, &name, &value);

```

```

        if(rc < 0 ){
            if(status == JEV_S_EXTATTR_EOD) {
                /* No more extended attribute exists. */
                break;
            }
            else {
                /* Error occurred while acquiring extended
                attributes. */
                fprintf(stderr,
                    "JevGetNextExtAttr() failed.
                    Status = %ld\n", status);
                JevFreeEvent(&status, event);
                JevGetClose(&status, key);
                return -1;
            }
        }
        else {
            printf("JevGetNextExtAttr() name = %s\n", name);
            printf("JevGetNextExtAttr() value = %s\n", value);
        }
    }

/* Release the memory allocated for the acquired JPl events. */
    rc = JevFreeEvent(&status, event);
    if(rc < 0){
        fprintf(stderr,
            "JevFreeEvent() failed. Status = %ld\n",
            status);
        JevGetClose(&status, key);
        return -1;
    }
}

/* Disconnect the event server. */
rc = JevGetClose(&status, key);
if(rc < 0){
    fprintf(stderr,
        "JevGetClose() failed. Status = %ld\n",
        status);
    return -1;
}

return 0;
}

int main()
{
    return get_start_event();
}

```

# Index

## A

Administrator permissions conventions 10  
attributes, basic 70

## C

coding example  
  for acquiring JP1 events 27, 77  
  for issuing JP1 events 24, 76  
common information 71  
compiler, installation of 20  
compiling  
  files needed for 29  
  options 30  
conventions  
  Administrator permissions 10  
  fonts and symbols 10  
  version numbers 11

## E

event attributes 22  
event levels 74  
events in JP1/SES format 16  
extended attributes 71  
  common information 71  
  specific information 74

## F

features 16  
font conventions 10  
function  
  description format of 35  
  JevFreeEvent 39  
  JevGetArrivedTime 40  
  JevGetBaselID 41  
  JevGetClose 42  
  JevGetCodeSet 43  
  JevGetDestinationAddress 44  
  JevGetDestinationServer 45  
  JevGetDetailInformation 46  
  JevGetEvent 47  
  JevGetExtAttrDirect 49  
  JevGetExtID 50  
  JevGetFirstExtAttr 51

JevGetMessage 52  
JevGetNextExtAttr 53  
JevGetOpen 54  
JevGetProcessID 56  
JevGetRegistFactor 57  
JevGetRegistGroupID 58  
JevGetRegistGroupName 59  
JevGetRegistTime 60  
JevGetRegistUserID 61  
JevGetRegistUserName 62  
JevGetSequenceNumber 63  
JevGetSourceAddress 64  
JevGetSourceSequenceNumber 65  
JevGetSourceServer 66  
JevRegistEvent 67  
functions 34  
  list of 37  
  notes common to all 36  
  sample source files of 17

## J

JevFreeEvent 39  
JevGetArrivedTime 40  
JevGetBaselID 41  
JevGetClose 42  
JevGetCodeSet 43  
JevGetDestinationAddress 44  
JevGetDestinationServer 45  
JevGetDetailInformation 46  
JevGetEvent 47  
JevGetExtAttrDirect 49  
JevGetExtID 50  
JevGetFirstExtAttr 51  
JevGetMessage 52  
JevGetNextExtAttr 53  
JevGetOpen 54  
JevGetProcessID 56  
JevGetRegistFactor 57  
JevGetRegistGroupID 58  
JevGetRegistGroupName 59  
JevGetRegistTime 60  
JevGetRegistUserID 61  
JevGetRegistUserName 62  
JevGetSequenceNumber 63

- JevGetSourceAddress 64
- JevGetSourceSequenceNumber 65
- JevGetSourceServer 66
- JevRegistEvent 67
- JP1/Base
  - customizing functionality of 15
  - extending functionality of 15
  - features of 16
  - installation 20
- JP1/SES event 16
- JP1 event acquisition functions
  - JevFreeEvent 39
  - JevGetArrivedTime 40
  - JevGetBaseID 41
  - JevGetClose 42
  - JevGetCodeSet 43
  - JevGetDestinationAddress 44
  - JevGetDestinationServer 45
  - JevGetDetailInformation 46
  - JevGetEvent 47
  - JevGetExtAttrDirect 49
  - JevGetExtID 50
  - JevGetFirstExtAttr 51
  - JevGetMessage 52
  - JevGetNextExtAttr 53
  - JevGetOpen 54
  - JevGetProcessID 56
  - JevGetRegistFactor 57
  - JevGetRegistGroupID 58
  - JevGetRegistGroupName 59
  - JevGetRegistTime 60
  - JevGetRegistUserID 61
  - JevGetRegistUserName 62
  - JevGetSequenceNumber 63
  - JevGetSourceAddress 64
  - JevGetSourceSequenceNumber 65
  - JevGetSourceServer 66
  - writing code that uses 26
- JP1 event issuing function
  - JevRegistEvent 67
  - writing code that uses 24
- JP1 events
  - acquisition functions (JP1/Base) 37
  - attributes, criteria for setting 70
  - basic attributes 70
  - defining event acquisition filter 26
  - determining types and attributes of 22, 25

- extended attributes 71
- functions for issuing and acquiring 18
- issuing function (JP1/Base) 37
- overview of function for issuing 19
- overview of functions 19
- overview of functions for acquiring 19
- prerequisites 20
- procedure for issuing 22
- procedures for acquiring 25

## L

- linking
  - files needed for 29
  - libraries 29
  - options 31

## M

- migrating
  - from earlier version 33
  - without recompiling 33
  - with recompiling 33

## O

- overview 15

## R

- receiver.c (sample source file for acquiring JP1 events) 75, 77

## S

- sample source files 75
  - coding of 76
  - details of 75
  - events handled by 75
- sender.c (sample source file for issuing JP1 events) 75, 76
- SEVERITY 71
  - event levels 74
- source files, compiling 29
- specific information 74
- symbol conventions 10
- syntax conventions 10

## U

- user-defined events 16

## V

version number conventions 11