HITACHI
Inspire the Next

JP1 Version 11

JP1/Base Function Reference

3021-3-A03-20(E)

Notices

■ Relevant program products

For details about the supported operating systems and the service packs or patches that are required by JP1/Base, see the *Release Notes*.

For Windows:

P-2A2C-6LBL JP1/Base 11-50 (for Windows Server 2008 R2, Windows 7, Windows Server 2012, Windows 8, Windows 8.1, Windows 10, Windows Server 2016)

For UNIX:

P-1J2C-6LBL JP1/Base 11-50 (for HP-UX (IPF))

P-9D2C-6LBL JP1/Base 11-50 (for Solaris (SPARC))

P-1M2C-6LBL JP1/Base 11-50 (for AIX)

P-812C-6LBL JP1/Base 11-50 (for Linux 6 (x64), Linux 7, Oracle Linux 6 (x64), Oracle Linux 7, CentOS 6 (x64), CentOS 7, SUSE Linux 12)

■ Trademarks

```
* [including the GNU Public Licence.]
*/
```

■ Microsoft product name abbreviations

This manual uses the following abbreviations for Microsoft product names.

Abbreviation		Full name or meaning
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 Enterprise
		Windows$^{(R)}$ 8 Pro
Windows 8.1		Windows$^{(R)}$ 8.1
		Windows$^{(R)}$ 8.1 Enterprise
		Windows$^{(R)}$ 8.1 Pro
Windows 10		Windows$^{(R)}$ 10 Enterprise
		Windows$^{(R)}$ 10 Home
		Windows$^{(R)}$ 10 Pro
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 Server 2016		Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2016 Datacenter
		Microsoft$^{(R)}$ Windows Server$^{(R)}$ 2016 Standard

Windows is sometimes used generically, referring to Windows Server 2008 R2, Windows 7, Windows Server 2012, Windows 8, Windows 8.1, Windows 10 and Windows Server 2016.

■ 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

■ Edition history

Nov. 2017: 3021-3-A03-20(E)

■ Copyright

Summary of amendments

The following table lists changes in this manual (3021-3-A03-20(E)) and product changes related to this manual.

Changes	Location
--	--

Legend:

--: Not applicable

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

Preface

This manual explains in detail the functions provided by JP1/Base and the procedures used to extend the functions of JP1/Base during development of systems linked to JP1/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.

■ 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
JP1/Base User's Guide	• Overview and functionality of JP1/Base • Setup of each function • Commands, definition files, JP1 events • Troubleshooting • Processes, port numbers, operation logs
JP1/Base Messages	Messages
JP1/Base Function Reference	• Procedures for issuing and acquiring JP1 events with JP1 programs and user applications • Functions

■ 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

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

Text formatting	Convention
Bold	Bold characters indicate text in a window, other than the window title. Such text includes menus, menu options, buttons, radio box options, or explanatory labels. For example: • From the **File** menu, choose **Open**. • Click the **Cancel** button. • In the **Enter name** entry box, type your name.
Italic	Italic characters indicate a placeholder for some actual text to be provided by the user or system. For example: • Write the command as follows: `copy` *source-file target-file* • The following message appears: `A file was not found. (file =` *file-name*`)` Italic characters are also used for emphasis. For example: • Do *not* delete the configuration file.
`Monospace`	Monospace characters indicate text that the user enters without change, or text (such as messages) output by the system. For example: • At the prompt, enter `dir`. • Use the `send` command to send mail. • The following message is displayed: `The password is incorrect.`

The following table explains the symbols used in this manual:

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

■ 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	*installation-folder*	In an x86 environment: *system-drive*`:\Program Files\HITACHI\JP1Base` In an x64 environment: *system-drive*`:\Program Files (x86)\HITACHI\JP1Base`

#

Denotes the installation folder used when the product is installed with initial settings. 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 might differ depending on the environment.

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`.

■ Other reference information

For other reference information, see *Reference Material for this Manual* in the *JP1/Base User's Guide*.

Contents

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 *Chapter 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 *JP1/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 *Chapter 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 *JP1/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 *Appendix 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

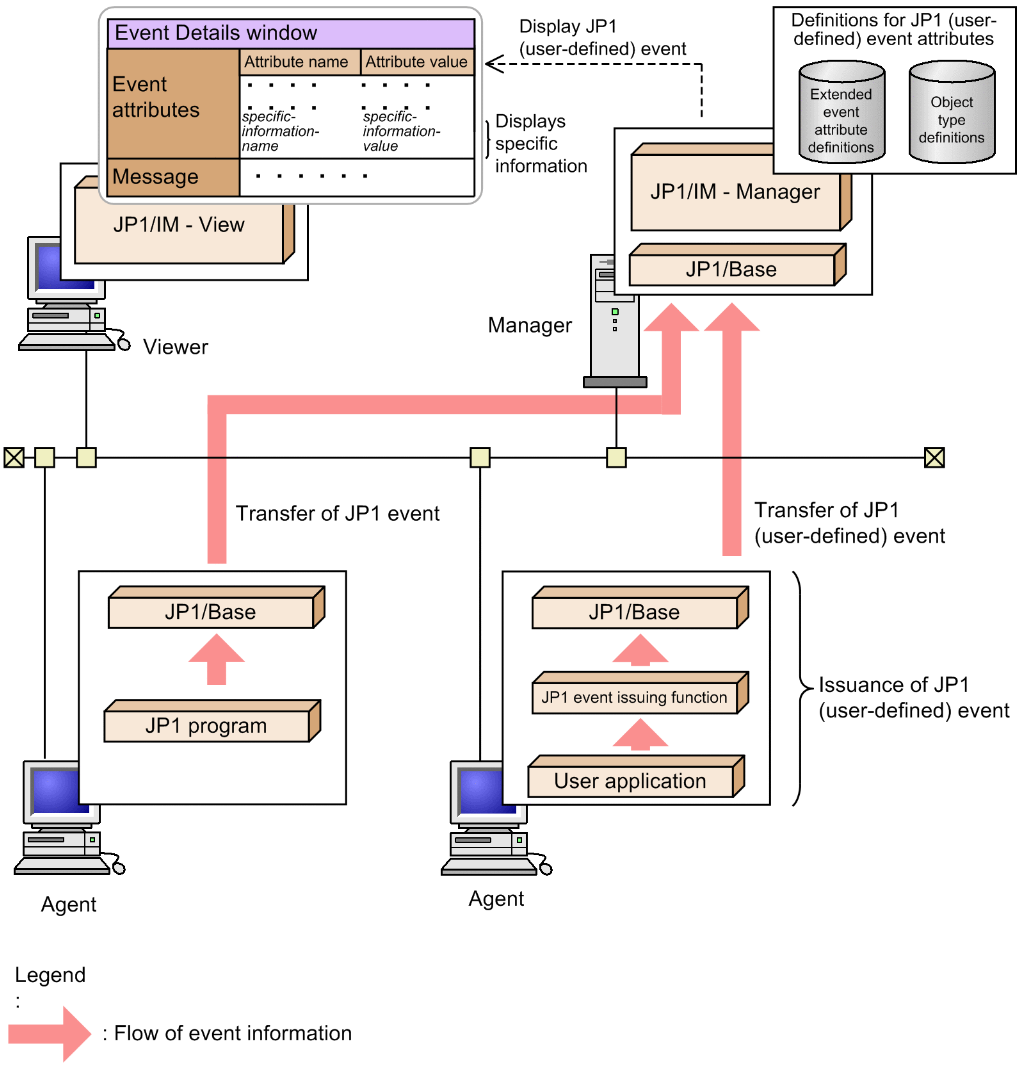
JP1/IM enables you to monitor events by converting application-specific log files, SNMP traps, and the Windows event log to JP1 events. However, you cannot use JP1/IM to define application-specific event attributes and other information in detail.

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


Event Details window
Attribute name
Attribute value
Event attributes
specific-information-name
specific-information-value
Message
Displays specific information
Display JP1 (user-defined) event
Definitions for JP1 (user-defined) event attributes
Extended event attribute definitions
Object type definitions
JP1/IM - Manager
JP1/Base
JP1/IM - View
Viewer
Manager
Transfer of JP1 event
Transfer of JP1 (user-defined) event
JP1/Base
JP1 program
JP1/Base
JP1 event issuing function
User application
Issuance of JP1 (user-defined) event
Agent
Agent
Legend
:
: Flow of event information

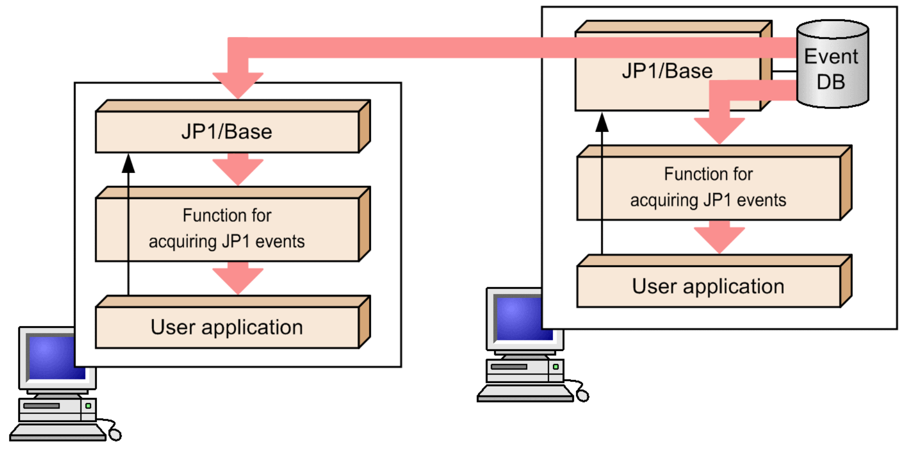
Acquiring JP1 events

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.

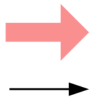
By using the JP1 event acquisition functions, user applications are able to acquire and use JP1 events directly from the JP1/Base event database.

The following figure shows an overview of JP1 event acquisition.

Figure 2–2: JP1 event acquisition


Event DB
JP1/Base
Function for acquiring JP1 events
User application
JP1/Base
Function for acquiring JP1 events
User application

Legend:



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	• Visual C++$^{(R)}$ 2010 • Visual C++$^{(R)}$ 2012 • Visual C++$^{(R)}$ 2012
HP-UX (IPF)	HP C/aC++ A.06.05 or later

OS	Compiler
Solaris (SPARC)	Sun Studio 12
AIX	• XL C/C++ Enterprise Edition V9.0 for AIX • XL C/C++ Enterprise Edition V10.1 for AIX • XL C/C++ Enterprise Edition V11.1 for AIX • XL C/C++ Enterprise Edition V12.1 for AIX • XL C/C++ Enterprise Edition V13.1 for AIX
Linux	• gcc version 4.4.5 • gcc version 4.4.7 • gcc version 4.8

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 *JP1/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 *Appendix 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 *JP1/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 file (`conf`) 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 *Chapter 3. Functions*. For details about what types of event attributes can be acquired, see *Appendix 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	Threading	Required library
Windows	32-bit multi-threaded	*installation-folder*`\lib\libJevApiA.lib`
	64-bit multi-threaded	*installation-folder*`\lib\libJevApiAx64.lib`
HP-UX (IPF)	32-bit single-threaded	`/opt/jp1base/lib/libJevApiAst32.a`

OS	Threading	Required library
	32-bit multi-threaded	`/opt/jp1base/lib/libJevApiAmt32.a`
	64-bit single-threaded	`/opt/jp1base/lib/libJevApiAst64.a`
	64-bit multi-threaded	`/opt/jp1base/lib/libJevApiAmt64.a`
Solaris (SPARC)	32-bit single-threaded	`/opt/jp1base/lib/libJevApiAst.a`
	32-bit multi-threaded	`/opt/jp1base/lib/libJevApiAmt.a`
• AIX • Linux	32-bit single-threaded	`/opt/jp1base/lib/libJevApiAst.a`
	32-bit multi-threaded	`/opt/jp1base/lib/libJevApiAmt.a`
	64-bit single-threaded	`/opt/jp1base/lib/libJevApiAst64.a`
	64-bit multi-threaded	`/opt/jp1base/lib/libJevApiAmt64.a`

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	Threading	Compile option
Windows	32-bit multi-threaded	`/MD /I "`*installation-folder*`\include"` (Implement in a 32-bit VC++ project configuration)
	64-bit multi-threaded	`/MD /I "`*installation-folder*`\include"` (Implement in a 64-bit VC++ project configuration)
HP-UX (IPF)	32-bit single-threaded	`-Aa`# `-I/opt/jp1base/include`
	32-bit multi-threaded	`-Aa`# `-mt -I/opt/jp1base/include`
	64-bit single-threaded	`+DD64 -Aa`# `-I/opt/jp1base/include`
	64-bit multi-threaded	`+DD64 -Aa`# `-mt -I/opt/jp1base/include`
Solaris (SPARC)	32-bit single-threaded	`-I/opt/jp1base/include`
	32-bit multi-threaded	`-mt -D_THREAD_SAFE -I/opt/jp1base/include`
AIX	32-bit single-threaded	`-I/opt/jp1base/include`
	32-bit multi-threaded	`-D_REENTRANT -D_THREAD_SAFE -I/opt/jp1base/include`
	64-bit single-threaded	`-q64 -I/opt/jp1base/include`
	64-bit multi-threaded	`-q64 -D_REENTRANT -D_THREAD_SAFE -I/opt/jp1base/include`
Linux	32-bit single-threaded	`-I/opt/jp1base/include`
	32-bit multi-threaded	`-D_REENTRANT -D_THREAD_SAFE -I/opt/jp1base/include`
	64-bit single-threaded	`-m64 -I/opt/jp1base/include`

OS	Threading	Compile option
	64-bit multi-threaded	`-m64 -D_REENTRANT -D_THREAD_SAFE -I/opt/jp1base/include`

#: The `-Aa` option for HP-UX (IPF) 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	Threading	Link option
Windows	32-bit multi-threaded	`"`*installation-folder*`\lib\libJevApiA.lib"` (Implement in a 32-bit VC++ project configuration)
	64-bit multi-threaded	`"`*installation-folder*`\lib\libJevApiAx64.lib"` (Implement in a 64-bit VC++ project configuration)
HP-UX (IPF)	32-bit single-threaded	`/opt/jp1base/lib/libJevApiAst32.a`
	32-bit multi-threaded	`-mt /opt/jp1base/lib/libJevApiAmt32.a`
	64-bit single-threaded	`+DD64 /opt/jp1base/lib/libJevApiAst64.a`
	64-bit multi-threaded	`+DD64 -mt /opt/jp1base/lib/libJevApiAmt64.a`
Solaris (SPARC)	32-bit single-threaded	`/opt/jp1base/lib/libJevApiAst.a -ldl`
	32-bit multi-threaded	`/opt/jp1base/lib/libJevApiAmt.a -ldl -lpthread`
AIX	32-bit single-threaded	`/opt/jp1base/lib/libJevApiAst.a -ldl`
	32-bit multi-threaded	`/opt/jp1base/lib/libJevApiAmt.a -ldl -lpthread`
	64-bit single-threaded	`/opt/jp1base/lib/libJevApiAst64.a -q64 -ldl`
	64-bit multi-threaded	`/opt/jp1base/lib/libJevApiAmt64.a -q64 -ldl -lpthread`
Linux	32-bit single-threaded	`/opt/jp1base/lib/libJevApiAst.a -ldl`
	32-bit multi-threaded	`/opt/jp1base/lib/libJevApiAmt.a -ldl -lpthread`
	64-bit single-threaded	`/opt/jp1base/lib/libJevApiAst64.a -m64 -ldl`
	64-bit multi-threaded	`/opt/jp1base/lib/libJevApiAmt64.a -m64 -ldl -lpthread`

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: • JP1/Base 09-10 • Compiler • User application	If JP1/Base is a later version: • JP1/Base 09-50 or later	Y
	If JP1/Base is the same version: • JP1/Base 09-10	Y
	If JP1/Base is an earlier version: • JP1/Base 09-00 or earlier	N

Legend:

Y: Assured.

N: Not assured.

Important

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 09-10 might not be able to run on an OS for which support was added in JP1/Base 09-50. 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 *Appendix 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* 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 3–3: Status code and meaning (JevFreeEvent)

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 `0`.
Abnormal termination	The function returns `-1`. Also, in the area specified in *lplStatus*, 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* 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 3–4: 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 *lplStatus*, 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* 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 3–5: 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 *lplStatus*, 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* lplStatus,
                JEVGETKEY key);
```

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 3–6: Status codes and meanings (JevGetClose)

Status code	Meaning
`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 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 (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 *lplStatus*, 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* lplStatus,
                  JP1EVENT event,
                  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 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 *lppszValue*, the function stores the pointer to the code set.
Abnormal termination	The function returns `-1`. Also, in the area specified in *lplStatus*, 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 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 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.

lppszValue

In *lppszValue*, 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 *lppszValue*, stores the pointer to the destination IP address. Also, in the area specified in *lpnSize*, the function stores the size of the destination IP address.
Abnormal termination	The function returns `-1`. Also, in the area specified in *lplStatus*, 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* lplStatus,
                            JP1EVENT event,
                            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 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).

lppszValue

In *lppszValue*, 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 *lppszValue*, the function stores the pointer to the destination event server name.
Abnormal termination	The function returns `-1`. Also, in the area specified in *lplStatus*, 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* lplStatus,
                            JP1EVENT event,
                            long* lplSize,
                            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 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).

lplSize

In *lplSize*, 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 *lppszValue*, stores the pointer to the detailed information. Also, in the area specified in *lplSize*, the function stores the length of the detailed information.
Abnormal termination	The function returns `-1`. Also, in the area specified in *lplStatus*, 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
`JEV_S_CONNECT_ERROR`	Failed to connect the event server.
`JEV_S_INVALID_SERVER`	The event server name is invalid.
`JEV_S_PARAM_ERROR`	An invalid parameter is specified.
`JEV_S_NO_EVENT`	No JP1 events satisfy the filter condition.
`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.

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 *lplStatus*, 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 *lplStatus*, 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* 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 3–13: 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 *lplStatus*, 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 *lppszName*, stores the pointer to the extended attribute name. Also, in the area specified in *lppszValue*, the function stores the pointer to the extended attribute value.
Abnormal termination	The function returns `-1`. Also, in the area specified in *lplStatus*, 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* lplStatus,
                  JP1EVENT event,
                  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 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 *lppszValue*, the function stores the pointer to the message.
Abnormal termination	The function returns `-1`. Also, in the area specified in *lplStatus*, the function stores the detailed error code.

JevGetNextExtAttr

Description

This function acquires the next specified extended attribute of a JP1 event after an extended attribute is acquired by a JP1 `JevGetFirstAttr()` or a `JevGetNextAttr()` event acquisition function.

Definition header

`JevApi.h`

Format

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

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.

Return values

Situation	Explanation
Normal termination	The function returns `0` and, in the area specified in *lppszName*, stores the pointer to the next extended attribute name. Also, in the area specified in *lppszValue*, the function stores the pointer to the next extended attribute value.
Abnormal termination	The function returns `-1`. Also, in the area specified in *lplStatus*, 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.#1 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.#2
`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_CANNOY_USED`	The regular expression library cannot be used.

#1: 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.

#2: The `users` parameter in the event server settings file (`conf`) 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 *JP1/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 *lplStatus*, 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 *lplStatus*, 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* 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 3–19: 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 *lplStatus*, 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* 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 3–20: 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).

lplSize

In *lplSize*, 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 *lplSize*, the function stores the event-originating group ID.
Abnormal termination	The function returns `-1`. Also, in the area specified in *lplStatus*, 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* lplStatus,
                          JP1EVENT event,
                          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 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 *lppszValue*, the function stores the pointer to the event-originating group name.
Abnormal termination	The function returns `-1`. Also, in the area specified in *lplStatus*, 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* 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 3–22: 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 *lplStatus*, 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 *lplSize*, the function stores the event-originating user ID.
Abnormal termination	The function returns `-1`. Also, in the area specified in *lplStatus*, 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* lplStatus,
                         JP1EVENT event,
                         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 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 *lppszValue*, the function stores the pointer to the event-originating user name.
Abnormal termination	The function returns `-1`. Also, in the area specified in *lplStatus*, 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* 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 3–25: 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 *lplStatus*, 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** 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 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.

lppszValue

In *lppszValue*, 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 *lppszValue*, stores the pointer to the event-originating IP address. Also, in the area specified in *lpnSize*, the function stores the length of the event-originating IP address.
Abnormal termination	The function returns `-1`. Also, in the area specified in *lplStatus*, 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* 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 3–27: 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 *lplStatus*, 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 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 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).

lppszValue

In *lppszValue*, 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 *lppszValue*, the function stores the pointer to the event-originating event server name.
Abnormal termination	The function returns `-1`. Also, in the area specified in *lplStatus*, 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.# 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.

#: 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 *Appendix A. Criteria for Setting JP1 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: • A clear explanation of the event • Written in one line without a new line code
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 *Table A-3 Event levels*.
`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: • */company-name/series-name/product-name* • */company-name/product-name* 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. • `JOB` • `JOBNET`

Attribute name	Item	Explanation
		• ACTION • ACTIONFLOW • PRINTJOB • PRINTQUEUE • PRINTER • BATCHQUEUE • PIPEQUEUE • JOBBOX • LOGFILE • LINK (for reporting events from a lower communication layer) • SERVICE (e.g., daemon process) • PRODUCT (for reporting other program-specific events) • CONFIGRATION • SERVER • BACKUP • RESTORE • MEDIA
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	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. ACTIVE The object became active. INACTIVE The object became inactive. START The object started. END The object terminated. NOTSTART The object failed to start. CANCEL The object was canceled.

Attribute name	Item	Explanation

		`LATESTART`
		Exceeded the scheduled start time.
		`LATEEND`
		Exceeded the scheduled end time.
		`SUBMIT`
		The object was submitted.
		`UNSUBMIT`
		Submitting the object was canceled.
		`ENQUEUE`
		The object was added to the queue.
		`DEQUEUE`
		The object was removed from the queue.
		`PAUSE`
		The object paused.
		`RELEASE`
		The object stopped pausing.
		`RESTART`
		The object restarted.
		`CREATE`
		The object was created.
		`DELETE`
		The object was deleted.
		`MODIFY`
		The object was modified.
		`RETRY`
		The object started a retry.
		`STOP`
		The object stopped.
		`MOVE`
		The object was moved.
		`COPY`
		The object was copied.
		`NOTICE`
		The object notified the operator.
		`REPLY`
		The object received a reply.
		`CONNECT`
		The object was connected.
		`DISCONNECT`
		The object was disconnected.
		`EXCEPTION`
		An exception occurred.

`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	**Emergency**	An emergency status. Normally, the events specifying this event level are broadcast to all users.
Alert	**Alert**	A status requiring an immediate recovery, such as damage to the system or database.
Critical	**Critical**	A critical status such as a hardware error.
Error	**Error**	An error.
Warning	**Warning**	A warning message.
Notice	**Notice**	A status that does not indicate an error, but indicates a situation that requires careful handling.
Information	**Information**	Information to users.
Debug	**Debug**	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 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 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 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;
}

int main()
{
       return get_start_event();
}
```

Index

Hitachi, Ltd.

6-6, Marunouchi 1-chome, Chiyoda-ku, Tokyo, 100–8280 Japan