HITACHI
Inspire the Next

JP1 Version 13

JP1/Base 関数リファレンス

3021-3-L11-10

前書き

■ 対象製品

適用 OS のバージョン，JP1/Base が前提とするサービスパックやパッチなどの詳細についてはリリースノートで確認してください。

●JP1/Integrated Management 3 - Manager（適用 OS：Windows）

P-2A2C-8EDL JP1/Integrated Management 3 - Manager 13-10

製品構成一覧および内訳形名

P-CC2A2C-9MDL JP1/Integrated Management 3 - Manager 13-10（適用 OS：Windows Server 2016，Windows Server 2019，Windows Server 2022）

P-CC2A2C-6HDL JP1/Integrated Management 3 - View 13-10（適用 OS：Windows 10，Windows Server 2016，Windows Server 2019，Windows Server 2022，Windows 11）

P-CC2A2C-9GDL JP1/Integrated Management 3 - Agent 13-10（適用 OS：Windows Server 2016，Windows Server 2019，Windows Server 2022）

P-CC842C-9GDL JP1/Integrated Management 3 - Agent 13-10（適用 OS：Linux 7，Linux 8，Linux 9，Oracle Linux 7，Oracle Linux 8，Oracle Linux 9，SUSE Linux 12，SUSE Linux 15，Amazon Linux 2023）

P-CC2A2C-6LDL JP1/Base 13-10（適用 OS：Windows Server 2016，Windows Server 2019，Windows Server 2022）

P-CC842C-6LDL JP1/Base 13-10（適用 OS：Linux 7，Linux 8，Linux 9，Oracle Linux 7，Oracle Linux 8，Oracle Linux 9，SUSE Linux 12，SUSE Linux 15，Amazon Linux 2023）

P-CC1M2C-6LDL JP1/Base 13-10（適用 OS：AIX）

●JP1/Automatic Job Management System 3 - Manager（適用 OS：Windows）

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

製品構成一覧および内訳形名

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

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

P-CC8412-39DL JP1/Automatic Job Management System 3 - Web Console 13-10（適用 OS：Linux 7，Linux 8，Linux 9，Oracle Linux 7，Oracle Linux 8，Oracle Linux 9，SUSE Linux 12，SUSE Linux 15，Amazon Linux 2023）

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

P-CC2A2C-6LDL JP1/Base 13-10（適用 OS：Windows Server 2016，Windows Server 2019，Windows Server 2022）

●JP1/Integrated Management 3 - Manager（適用 OS：Linux）

P-842C-8EDL JP1/Integrated Management 3 - Manager 13-10

製品構成一覧および内訳形名

P-CC842C-9MDL JP1/Integrated Management 3 - Manager 13-10（適用 OS：Linux 7，Linux 8，Linux 9，Oracle Linux 7，Oracle Linux 8，Oracle Linux 9）

P-CC9W2C-9MDL JP1/Integrated Management 3 - Manager 13-10（適用 OS：SUSE Linux 12，SUSE Linux 15）

P-CC2A2C-6HDL JP1/Integrated Management 3 - View 13-10（適用 OS：Windows 10，Windows Server 2016，Windows Server 2019，Windows Server 2022，Windows 11）

P-CC2A2C-9GDL JP1/Integrated Management 3 - Agent 13-10（適用 OS：Windows Server 2016，Windows Server 2019，Windows Server 2022）

P-CC842C-9GDL JP1/Integrated Management 3 - Agent 13-10（適用 OS：Linux 7，Linux 8，Linux 9，Oracle Linux 7，Oracle Linux 8，Oracle Linux 9，SUSE Linux 12，SUSE Linux 15，Amazon Linux 2023）

P-CC2A2C-6LDL JP1/Base 13-10（適用 OS：Windows Server 2016，Windows Server 2019，Windows Server 2022）

P-CC842C-6LDL JP1/Base 13-10（適用 OS：Linux 7，Linux 8，Linux 9，Oracle Linux 7，Oracle Linux 8，Oracle Linux 9，SUSE Linux 12，SUSE Linux 15，Amazon Linux 2023）

P-CC1M2C-6LDL JP1/Base 13-10（適用 OS：AIX）

●JP1/Automatic Job Management System 3 - Manager（適用 OS：Linux）

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

製品構成一覧および内訳形名

P-CC8412-4KDL JP1/Automatic Job Management System 3 - Manager 13-10（適用 OS：Linux 7，Linux 8，Linux 9，Oracle Linux 7，Oracle Linux 8，Oracle Linux 9，SUSE Linux 12，SUSE Linux 15，Amazon Linux 2023）

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

P-CC8412-39DL JP1/Automatic Job Management System 3 - Web Console 13-10（適用 OS：Linux 7，Linux 8，Linux 9，Oracle Linux 7，Oracle Linux 8，Oracle Linux 9，SUSE Linux 12，SUSE Linux 15，Amazon Linux 2023）

P-CC8412-3NDL JP1/Automatic Job Management System 3 - Print Option Manager 13-10（適用 OS：Linux 7，Linux 8，Linux 9，Oracle Linux 7，Oracle Linux 8，Oracle Linux 9，SUSE Linux 12，SUSE Linux 15，Amazon Linux 2023）

P-CC842C-6LDL JP1/Base 13-10（適用 OS：Linux 7，Linux 8，Linux 9，Oracle Linux 7，Oracle Linux 8，Oracle Linux 9，SUSE Linux 12，SUSE Linux 15，Amazon Linux 2023）

●JP1/Automatic Job Management System 3 - Agent（適用 OS：Windows）

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

製品構成一覧および内訳形名

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

P-CC2A2C-6LDL JP1/Base 13-10（適用 OS：Windows Server 2016，Windows Server 2019，Windows Server 2022）

●JP1/Automatic Job Management System 3 - Agent（適用 OS：AIX）

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

製品構成一覧および内訳形名

P-CC1M12-43DL JP1/Automatic Job Management System 3 - Agent 13-10（適用 OS：AIX）

P-CC1M2C-6LDL JP1/Base 13-10（適用 OS：AIX）

●JP1/Automatic Job Management System 3 - Agent（適用 OS：Linux）

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

製品構成一覧および内訳形名

P-CC8412-43DL JP1/Automatic Job Management System 3 - Agent 13-10（適用 OS：Linux 7，Linux 8，Linux 9，Oracle Linux 7，Oracle Linux 8，Oracle Linux 9，SUSE Linux 12，SUSE Linux 15，Amazon Linux 2023）

P-CC842C-6LDL JP1/Base 13-10（適用 OS：Linux 7，Linux 8，Linux 9，Oracle Linux 7，Oracle Linux 8，Oracle Linux 9，SUSE Linux 12，SUSE Linux 15，Amazon Linux 2023）

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

■ マイクロソフト製品の表記について

このマニュアルでは，マイクロソフト製品の名称を次のように表記しています。

表記	正式名称
Visual C++	Microsoft(R) Visual C++(R)
Windows 10	Windows(R) 10 Enterprise 64-bit
	Windows(R) 10 Home 64-bit
	Windows(R) 10 Pro 64-bit
Windows 11	Windows(R) 11 Enterprise
	Windows(R) 11 Home
	Windows(R) 11 Pro
Windows Server 2016	Microsoft(R) Windows Server(R) 2016 Datacenter
	Microsoft(R) Windows Server(R) 2016 Standard
Windows Server 2019	Microsoft(R) Windows Server(R) 2019 Datacenter
	Microsoft(R) Windows Server(R) 2019 Standard
Windows Server 2022	Microsoft(R) Windows Server(R) 2022 Datacenter
	Microsoft(R) Windows Server(R) 2022 Standard

Windows 10，Windows Server 2016，Windows Server 2019，Windows Server 2022 および Windows11 を総称して Windows と表記することがあります。

■ 発行

2024 年 9 月　3021-3-L11-10

■ 著作権

変更内容

変更内容（3021-3-L11-10） JP1/Base 13-10

追加・変更内容	変更個所
次の適用 OS を追加した。 • Amazon Linux 2023	－

（凡例）

－：該当なし

単なる誤字・脱字などはお断りなく訂正しました。

はじめに

このマニュアルは，JP1/Base の機能を拡張して JP1/IM と連携するシステムを開発する際に必要な，JP1/Base の提供する関数の詳細および作業の手順について説明したものです。このマニュアルは各 OS 共通のマニュアルです。OS ごとに差異がある場合は，本文中でそのつど内容を書き分けています。

■ 対象読者

このマニュアルは，JP1/IM と JP1/Base を使って JP1/IM と連携するシステムを開発するソフトウェア開発者の方を対象としています。また，JP1/IM および JP1/Base の機能を理解していることを前提としています。

■ マニュアルの構成

このマニュアルは，次に示す編から構成されています。

第 1 編　概要

JP1/Base の機能を拡張してできることの概要について説明しています。

第 2 編　運用

JP1/Base の機能を拡張する方法について説明しています。

第 3 編　リファレンス

JP1/Base の関数についてリファレンス形式で説明しています。

■ JP1/Base マニュアルの使い分けについて

JP1/Base のマニュアルは 3 冊に分かれています。次に示す表で各マニュアルの記載内容をご確認の上，利用目的に合わせてマニュアルをお読みください。

マニュアル名	記載内容
JP1/Base 運用ガイド	• JP1/Base の概要・機能 • 各機能の設定 • コマンド，定義ファイル，JP1 イベント • トラブルシューティング • プロセス，ポート番号，操作ログ
JP1/Base メッセージ	メッセージ
JP1/Base 関数リファレンス	• JP1 プログラムやユーザーアプリケーションで JP1 イベントを発行・取得する方法 • 関数

■ コマンドの文法で使用する記号

コマンドとパラメーターの説明で使用する記号を，次のように定義します。

記号	意味
| (ストローク)	複数の項目に対し，項目間の区切りを示し，「または」の意味を示す。 (例) 「A | B | C」は，「A，B または C」を示す。
{ }	この記号で囲まれている複数の項目の中から，必ず 1 組の項目を選択する。項目の区切りは | で示す。 (例) {A | B | C} は「A，B または C のどれかを指定する」ことを示す。
[ ]	この記号で囲まれている項目は任意に指定できる（省略してもよい）。 複数の項目が記述されている場合には，すべてを省略するか，どれか一つを選択する。 (例) [A]は「何も指定しない」か「A を指定する」ことを示す。 [B | C]は「何も指定しない」か「B または C を指定する」ことを示す。
… (点線)	この記号の直前に示された項目を繰り返して複数個，指定できる。 (例) 「A，B，…」は「A のあとに B を必要な個数だけ指定する」ことを示す。
( )	この記号で囲まれている項目をグループ化したことを示す。 (例) (A|B) は，A または B を 1 個以上指定することを示す。
△	空白を空けることを意味する。 (例) AAA△BBB は「AAA」と「BBB」の間に 1 個の空白を入れることを示す。
¥	上記の文字を指定値として使用する場合には，「¥」を前に付けて表現する。 (例) 「|」の文字を指定値として使いたい場合には「¥|」と表記する。

■ JP1/Base のインストール先フォルダの表記

このマニュアルでは，JP1/Base のインストール先フォルダを次のように表記しています。

製品名	インストール先フォルダの表記	インストール先フォルダ※
JP1/Base	インストール先フォルダ	システムドライブ:¥Program Files (x86)¥Hitachi¥JP1Base

注※　各製品を初期設定のままインストールした場合のインストール先フォルダを表しています。「システムドライブ:¥ProgramData」と表記している部分は，インストール時の OS 環境変数によって決定されるため，環境によって異なる場合があります。

（例）

ソースファイルのコンパイル時に必要なヘッダーファイル，JevApi.h のフルパス表記は，マニュアルでは次のようになります。

インストール先フォルダ¥include¥JevApi.h

■ このマニュアルで使用する「Administrators 権限」について

このマニュアルで表記している「Administrators 権限」とは，ローカル PC に対する Administrators 権限です。ローカル PC に対して Administrators 権限を持つユーザーであれば，ローカルユーザー，ドメインユーザー，および Active Directory 環境で動作に違いはありません。

■ このほかの参考情報

このほかの参考情報については，マニュアル「JP1/Base 運用ガイド」の「このマニュアルの参考情報」を参照してください。

目次

1 機能拡張の概要

この章では，JP1/Base の機能を拡張してできることの概要と，JP1/Base が提供する関数のサンプルについて説明します。

1.1 特長

JP1/Base の関数や定義ファイルを利用することによって，次のことができるようになります。

ユーザー独自の JP1 イベントを発行する

システムで発生する事象を，JP1/Base でユーザー独自のイベント属性を付加した JP1 イベントとして定義して，ユーザーアプリケーションから発行するように設定できます。ユーザーが定義するこのような JP1 イベントを，独自イベントと呼びます。独自イベントの属性は自由に定義できます。

独自イベントを発行できるようにするには，JP1 イベント発行関数を使用します。JP1 イベント発行関数を使用して独自イベントを発行する方法については，「2. JP1 イベントを発行および取得する」を参照してください。

なお，独自イベントに付加したユーザー独自のイベント属性を JP1/IM - View で表示するには，ユーザー独自のイベント属性を記述した定義ファイルを作成する必要があります。詳細については，マニュアル「JP1/Integrated Management 3 - Manager コマンド・定義ファイルリファレンス」を参照してください。

JP1 イベントを取得する

JP1/Base のイベント DB に登録された JP1 イベントを，ほかの JP1 プログラムやユーザーアプリケーションが直接取得するように設定できます。ユーザーアプリケーションから独自イベントを JP1/Base に発行し，JP1 イベントとしてイベント DB に登録した後，その JP1 イベントを別のユーザーアプリケーションで利用するなどの運用ができます。

JP1 イベントを取得するには，JP1 イベント取得関数を使用します。

JP1 イベント取得関数を使用して JP1 イベントを取得する方法については，「2. JP1 イベントを発行および取得する」を参照してください。

1.2　関数のサンプルソースファイルの提供

JP1/Base は，関数のサンプルソースファイルを提供しています。サンプルソースファイルを簡単に編集してコンパイルすることによって，自社業務に合わせて JP1 イベントを発行したり，JP1 イベントを取得したりできます。なお，サンプルソースファイルの詳細については，「付録 B　サンプルソースファイル」を参照してください。

2 JP1 イベントを発行および取得する

この章では，独自のイベント属性を付加した JP1 イベントをユーザーアプリケーションから直接発行する機能および JP1 イベントをほかの JP1 プログラムやユーザーアプリケーションで直接取得する機能の概要，前提条件，作業手順について説明します。

2.1　機能の解説

JP1 イベントを発行する

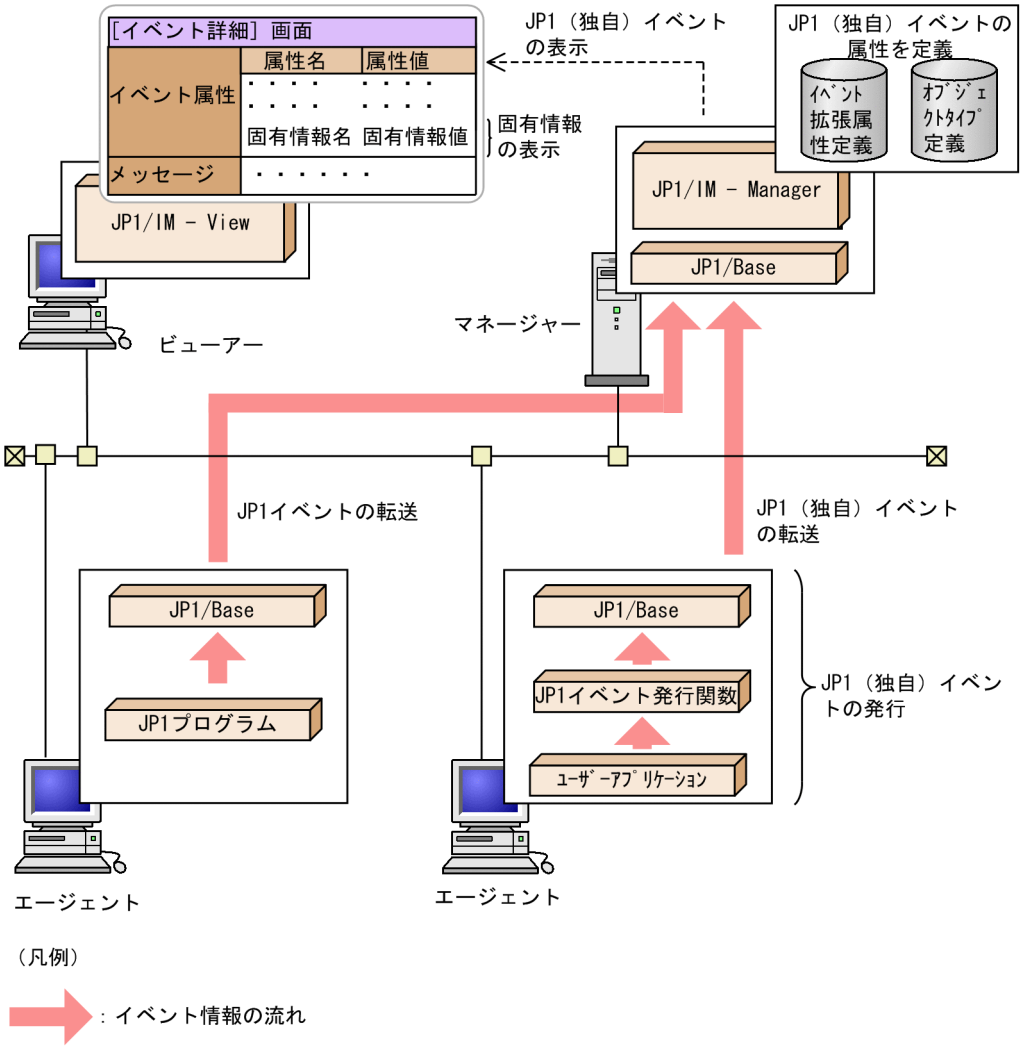
JP1/IM では，アプリケーション固有のログファイル，Windows のイベントログなどを JP1 イベントに変換してイベント監視ができます。しかし，個々のアプリケーションのイベント属性値などを独自にきめ細かく設定することはできません。

JP1/Base の JP1 イベント発行関数を使用することによって，ユーザー独自のイベント属性（拡張属性の固有情報）を付加した独自イベントをユーザーアプリケーションから直接，発行できるようになります。

JP1/IM では，定義ファイルを作成することによって，独自イベントのユーザー独自のイベント属性（拡張属性の固有情報）を［イベント詳細］画面に表示できます。

JP1（独自）イベントの発行からユーザー独自のイベント属性の表示までの概要を次の図に示します。

図 2-1　JP1（独自）イベントの発行からユーザー独自のイベント属性の表示までの概要



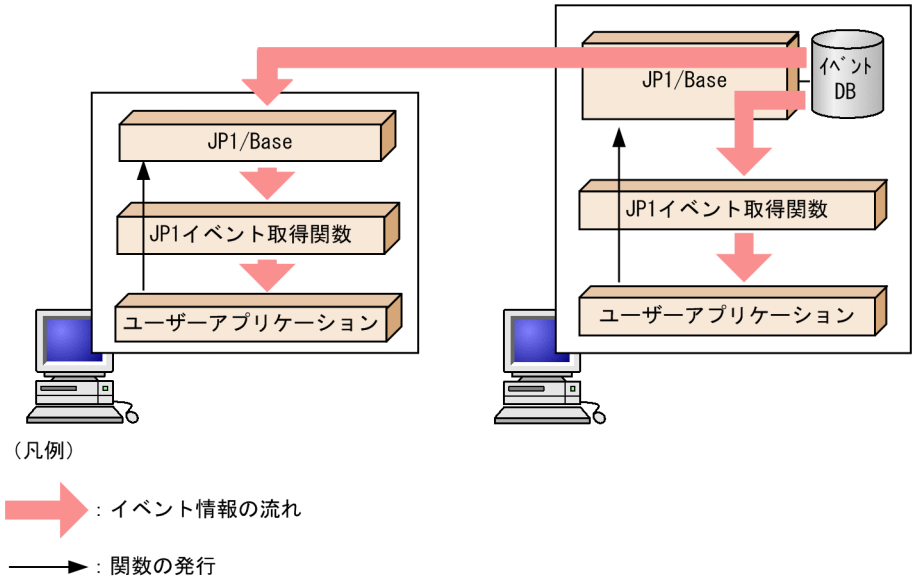
JP1 イベントを取得する

JP1/Base では，システム内で発生するさまざまな事象を JP1 イベントとしてイベント DB に登録，管理しています。しかし，ユーザーアプリケーションでは，この JP1 イベントを直接利用することはできません。

JP1/Base の JP1 イベント取得関数を利用することによって，ユーザーアプリケーションが，JP1/Base のイベント DB から JP1 イベントを直接取得し，利用できるようになります。

JP1 イベントの取得の概要を次の図に示します。

図 2-2　JP1 イベントの取得



2.1.1　前提条件

JP1/Base が提供する関数（JP1 イベント発行関数，JP1 イベント取得関数）を使用するための前提条件を次に示します。

- 関数（JP1 イベント発行関数，JP1 イベント取得関数）を使用したソースファイルをコンパイルするための環境

  次のものが必要です。

  - 各 OS 用の JP1/Base
  - 各 OS 用のコンパイラー

(1)　JP1/Base のインストール

JP1/Base が提供する関数を使用したソースファイルをコンパイルおよび実行するには，JP1/Base が必要です。これは，コンパイルおよび実行時に JP1/Base が提供するライブラリーおよびヘッダーファイルを利用するためです。コンパイルおよび実行をするマシンに，あらかじめ JP1/Base をインストールしておいてください。

(2) コンパイラーのインストール

JP1/Base が提供する関数を使用したソースファイルをコンパイルするには，次の表に示すどれかのコンパイラーが必要です。コンパイル作業をするマシンに，あらかじめコンパイラーをインストールしておいてください。

表 2-1　コンパイラーの種類

OS	コンパイラー
Windows	• Visual C++(R) 2010 • Visual C++(R) 2012 • Visual C++(R) 2013 • Visual C++(R) 2015 • Visual C++(R) 2017 • Visual C++(R) 2019
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　JP1 イベントを発行および取得する手順

2.2.1　JP1 イベントを発行する手順

作業の流れは次のとおりです。

1. 発行する JP1 イベントの種類とイベント属性を決める
2. JP1 イベント発行関数を使用してコーディングする
3. ソースファイルをコンパイルする

なお，JP1 イベントに付加したユーザー独自のイベント属性を JP1/IM - View に表示できるようにするには，このあと JP1/IM で，JP1/IM - Manager がインストールされたマシン上に，次の定義ファイルを作成する必要があります。

- イベント拡張属性定義ファイル
- オブジェクトタイプ定義ファイル

JP1/IM でこれらの定義ファイルを作成する方法については，マニュアル「JP1/Integrated Management 3 - Manager コマンド・定義ファイルリファレンス」を参照してください。

(1)　発行する JP1 イベントの種類とイベント属性を決める

JP1（独自）イベントを発行するためには，まず，どんな事象を JP1 イベントとして発行するのかを決めます。JP1/Base のイベントサービスでは，発行する JP1 イベントの数によって性能劣化が発生します。このため，JP1（独自）イベントは JP1/IM でシステム監視をするために必要な JP1 イベントだけに絞り込む必要があります。

次に，イベント属性をどのように設定するかを決めます。イベント属性を決めるときは，JP1/IM でイベント監視をする際にそのアプリケーションの情報として何が必要かを考えます。アプリケーション単位で，発行するすべての JP1（独自）イベントのイベント属性をあらかじめ決めておきます。

JP1 イベントの属性値は，JP1/IM で自動アクション機能およびモニター画面呼び出しの際の引数として使用できます。次の例で記述するイベント属性の詳細については，「付録 A　JP1 イベントの属性の設定基準」を参照してください。

ここでは，Windows 上で動作する「SAMPLE」という名称のアプリケーションについて，開始イベントと異常終了イベントを発行する例について説明します。（ ）内は，JP1 イベント発行関数の引数名です。

発行する JP1 イベントの種類

- アプリケーションの開始時に JP1 イベントを発行する（開始イベント）
  イベント ID（BaseID）：0x00000001
  メッセージ（message）：「SAMPLE アプリケーションを開始します。」

- アプリケーションの異常終了時に JP1 イベントを発行する（異常終了イベント）
  イベント ID（BaseID）：0x00000002
  メッセージ（message）：「SAMPLE アプリケーションが異常終了しました。」

イベント属性定義（拡張属性（extattrs））：開始イベント

SAMPLE アプリケーションの開始イベントには，次の属性を定義します。

表 2-2　開始イベントの属性

属性の種別	項目	属性名	内容
基本属性	イベント ID	−	0x00000001
	メッセージ	−	「Starts the SAMPLE application.」
拡張属性（共通情報）	重大度	SEVERITY	Notice
	ユーザー名	USER_NAME	SAMPLE_USER
	プロダクト名	PRODUCT_NAME	/COMPANY/APP1/SAMPLE_PRODUCT（プロダクトの名称）
	オブジェクトタイプ	OBJECT_TYPE	SAMPLE
	オブジェクト名	OBJECT_NAME	SAMPLE_NAME
	登録名タイプ	ROOT_OBJECT_TYPE	ROOT_SAMPLE
	登録名	ROOT_OBJECT_NAME	ROOT_SAMPLE_NAME
	オブジェクト ID	OBJECT_ID	SAMPLE_ID
	事象種別	OCCURRENCE	START
	開始時刻	START_TIME	SAMPLE アプリケーションの開始時刻。UTC 1970 年 1 月 1 日 00:00:00 からの秒数。
	プラットフォーム種別	PLATFORM	NT
	バージョン情報	ACTION_VERSION	0600
拡張属性（固有情報）	SAMPLE 共通属性 1	COMMON_ATTR1	NATIVE
	SAMPLE 共通属性 2	COMMON_ATTR2	TRUE
	SAMPLE 開始属性 1	START_ATTR1	SAMPLE1
	SAMPLE 開始属性 2	START_ATTR2	SAMPLE2

イベント属性定義（拡張属性（extattrs））：異常終了イベント

SAMPLE アプリケーションの異常終了イベントには，次の属性を定義します。

表 2-3　異常終了イベントの属性

属性の種別	項目	属性名	内容
基本属性	イベント ID	−	0x00000002
	メッセージ	−	「The SAMPLE application terminated abnormally.」
拡張属性（共通情報）	重大度	SEVERITY	Error
	ユーザー名	USER_NAME	SAMPLE_USER
	プロダクト名	PRODUCT_NAME	/COMPANY/APP1/SAMPLE_PRODUCT（プロダクトの名称）
	オブジェクトタイプ	OBJECT_TYPE	SAMPLE
	オブジェクト名	OBJECT_NAME	SAMPLE_NAME
	登録名タイプ	ROOT_OBJECT_TYPE	ROOT_SAMPLE
	登録名	ROOT_OBJECT_NAME	ROOT_SAMPLE_NAME
	オブジェクト ID	OBJECT_ID	SAMPLE_ID
	事象種別	OCCURRENCE	END
	終了時刻	END_TIME	SAMPLE アプリケーションの終了時刻。UTC 1970 年 1 月 1 日 00:00:00 からの秒数。
	終了コード	RESULT_CODE	SAMPLE アプリケーション終了時の終了コード
	プラットフォーム種別	PLATFORM	NT
	バージョン情報	ACTION_VERSION	0600
拡張属性（固有情報）	SAMPLE 共通属性 1	COMMON_ATTR1	NATIVE
	SAMPLE 共通属性 2	COMMON_ATTR2	TRUE
	SAMPLE 終了属性 1	END_ATTR1	SAMPLE1
	SAMPLE 終了属性 2	END_ATTR2	SAMPLE2

(2)　JP1 イベント発行関数を使用してコーディングする

前の項で説明した SAMPLE アプリケーションで，開始イベントを発行する場合のコーディング例を次に示します。

```
#include <stdio.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 */
    const char* extattrs[15];   /* 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";
    extattrs[9] = "PLATFORM=NT";
    extattrs[10] = "VERSION=0600";
    extattrs[11] = "COMMON_ATTR1=NATIVE";
    extattrs[12] = "COMMON_ATTR2=TRUE";
    extattrs[13] = "START_ATTR1=SAMPLE1";
    extattrs[14] = "START_ATTR2=SAMPLE2";

    /* Register the JP1 event. */
    rc = JevRegistEvent(&status,
                        server,
                        baseID,
                        message,
                        extattrs,
                        15);
    if(rc < 0) {
        fprintf(stderr,
                "JevRegistEvent() failed. status = %ld¥n",
                 status);
        return -1;
    }

    return 0;
}
```

2.2.2 JP1 イベントを取得する手順

作業の流れは次のとおりです。

1. 取得する JP1 イベントの種類とイベント属性を決める

2. 取得する JP1 イベントをイベント取得フィルターによって指定する

3. JP1 イベント取得関数を使用してコーディングする

4. ソースファイルをコンパイルする

(1)　取得する JP1 イベントの種類とイベント属性を決める

JP1/Base では，さまざまな種類の事象が JP1 イベントとしてイベント DB に登録されます。まず，このイベント DB の中からどのような種類の JP1 イベントを取得するかを決めます。

次に，その JP1 イベントからどのイベント属性を取得すればよいかを決めます。取得するイベント属性を決めるときは，そのアプリケーションの情報として何が必要かを考えます。アプリケーション単位で，取得するすべての JP1 イベントのイベント属性をあらかじめ決めておきます。

ここでは，「2.2.1(1)　発行する JP1 イベントの種類とイベント属性を決める」で JP1 イベントとして発行した SAMPLE アプリケーションの「開始イベント」の取得を例に説明します。

(2)　取得する JP1 イベントをイベント取得フィルターによって指定する

必要な JP1 イベントだけを取得するには，イベント取得フィルターを定義する必要があります。イベント取得フィルターの文法の詳細については，マニュアル「JP1/Base 運用ガイド」の，フィルターの文法の項を参照してください。ここでは，「2.2.1(1)　発行する JP1 イベントの種類とイベント属性を決める」で示した SAMPLE アプリケーションの「開始イベント」を取得するためのイベント取得フィルターの例を次に示します。

まず，「開始イベント」を取得するために，次に示す条件を付けたイベント取得フィルターを作成することを検討します。

- イベント ID：0x00000001
- 拡張属性「SEVERITY」の値：Notice
- 拡張属性「PRODUCT_NAME」の値：/COMPANY/APP1/SAMPLE_PRODUCT

上記条件に合致する JP1 イベントを取得対象にすることで，「開始イベント」が取得できるようになります。上記条件に合致するイベント取得フィルターの例を次に示します。

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

注意事項

- イベント取得フィルターの条件として日本語文字列を指定する場合，その文字コードは，JP1 イベント取得関数実行時のロケール情報（環境変数 LANG など）と合致させてください。イベント取得

フィルターの条件として指定した文字列の文字コードと JP1 イベント取得関数実行時のロケール情報（環境変数 LANG など）が異なる場合，JP1 イベントは取得できません。

- イベント取得フィルターに除外条件を定義するときは，08-50 以降のイベントサーバに接続してください。08-11 以前のイベントサーバに接続するとエラー（JEV_S_FILTER_ERROR）になります。

(3) JP1 イベント取得関数を使用してコーディングする

ほかの JP1 プログラムやユーザーアプリケーションが JP1 イベントを取得する場合，JP1 イベント取得関数を利用します。JP1/Base のイベント DB から JP1 イベントを取得するには，次の順序で JP1 イベント取得関数を発行します。

1. JP1 イベントの取得開始を要求する関数を発行する

   イベントサーバに対して関数（JevGetOpen）を使って JP1 イベントの取得開始を要求し，イベントサーバに接続します。なお，取得開始を要求するユーザーは，あらかじめ JP1/Base のイベントサーバ設定ファイル（conf）のusers パラメーターで設定する必要があります。

2. JP1 イベントの取得を要求する関数を発行する

   さまざまな関数を使って JP1 イベントを取得したり，JP1 イベントに設定されたさまざまなイベント属性を取得したりします。

3. JP1 イベントの取得終了を通知する関数を発行する

   イベントサーバに対して関数（JevGetClose）を使って JP1 イベントの取得終了を通知し，イベントサーバとの接続を切断します。

JP1 イベント取得関数の詳細については，「3. 関数」を参照してください。また，取得できるイベント属性がどのようなものであるかについては，「付録 A JP1 イベントの属性の設定基準」を参照してください。

ここでは，「2.2.1(1) 発行する JP1 イベントの種類とイベント属性を決める」に示した SAMPLE アプリケーションの「開始イベント」を取得するためのコーディング例を次に示します。

```
#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;
}
```

2.2.3　ソースファイルをコンパイルする

JP1 イベントを発行および取得するには，コーディングしたソースファイルをコンパイル・リンクする必要があります。

[コンパイル時に必要なファイル]

- ヘッダーファイル（JP1/Base インストール時にインストールされます）
- C または C++で作成したソースファイル（ユーザーが作成するものです）

ヘッダーファイルの格納先は次のとおりです。

Windows：**インストール先フォルダ¥include¥JevApi.h**

UNIX：/opt/jp1base/include/JevApi.h

[リンク時に必要なファイル]

- ライブラリー（JP1/Base インストール時にインストールされます）

必要となるライブラリーは，各 OS，コンパイラーによって異なるため，注意が必要です。各 OS で必要なライブラリーを次の表に示します。

なお，ライブラリーには，2038 年非対応と 2038 年対応の 2 種類があります。2038 年以降も使用する場合は，2038 年対応のライブラリーをリンクするようにしてください。

表 2-4　各 OS で必要なライブラリー（2038 年非対応）

OS	スレッド	必要なライブラリー
Windows	32 ビット，マルチスレッド	**インストール先フォルダ¥lib¥libJevApiA.lib**
	64 ビット，マルチスレッド	**インストール先フォルダ¥lib¥libJevApiAx64.lib**
• AIX • Linux	32 ビット，シングルスレッド	/opt/jp1base/lib/libJevApiAst.a
	32 ビット，マルチスレッド	/opt/jp1base/lib/libJevApiAmt.a
	64 ビット，シングルスレッド	/opt/jp1base/lib/libJevApiAst64.a
	64 ビット，マルチスレッド	/opt/jp1base/lib/libJevApiAmt64.a

表 2-5　各 OS で必要なライブラリー（2038 年対応）

OS	スレッド	必要なライブラリー
Windows	32 ビット，マルチスレッド	**インストール先フォルダ¥lib¥libJevApiAT.lib**
	64 ビット，マルチスレッド	**インストール先フォルダ¥lib¥libJevApiATx64.lib**
• AIX • Linux	32 ビット，シングルスレッド	/opt/jp1base/lib/libJevApiATmt.a
	32 ビット，マルチスレッド	/opt/jp1base/lib/libJevApiATst.a

OS	スレッド	必要なライブラリー
	64 ビット，シングルスレッド	/opt/jp1base/lib/libJevApiATmt.a
	64 ビット，マルチスレッド	/opt/jp1base/lib/libJevApiATst64.a

各 OS でのコンパイル，リンク時に指定するオプションを次の表に示します。

なお，リンクオプションには，2038 年非対応と 2038 年対応の 2 種類があります。2038 年以降も使用する場合は，2038 年対応のリンクオプションを使用するようにしてください。

注意事項

Windows の Visual Studio 統合開発環境（GUI）上でコンパイルやリンクをする場合，次の表に示すコンパイルオプションやリンクオプションから，該当するオプションを使用して環境を設定してください。

表 2-6　コンパイルオプション

OS	スレッド	コンパイルオプション
Windows	32 ビット，マルチスレッド	/MD /I ”インストール先フォルダ¥include” （32 ビットの VC++プロジェクト構成で実施）
	64 ビット，マルチスレッド	/MD /I ”インストール先フォルダ¥include” （64 ビットの VC++プロジェクト構成で実施）
AIX	32 ビット，シングルスレッド	-I/opt/jp1base/include
	32 ビット，マルチスレッド	-D_REENTRANT -D_THREAD_SAFE -I/opt/jp1base/include
	64 ビット，シングルスレッド	-q64 -I/opt/jp1base/include
	64 ビット，マルチスレッド	-q64 -D_REENTRANT -D_THREAD_SAFE -I/opt/jp1base/include
Linux	32 ビット，シングルスレッド	-I/opt/jp1base/include
	32 ビット，マルチスレッド	-D_REENTRANT -D_THREAD_SAFE -I/opt/jp1base/include
	64 ビット，シングルスレッド	-m64 -I/opt/jp1base/include
	64 ビット，マルチスレッド	-m64 -D_REENTRANT -D_THREAD_SAFE -I/opt/jp1base/include

注※　HP-UX (IPF)の-Aa オプションは，C コンパイラー（cc）でコンパイルする場合だけ必要です。-Ae オプションに置き換えられますが，-Ac オプションは指定しないでください。なお，C++コンパイラー（aCC）を使用する場合は省略できます。

表 2-7　リンクオプション（2038 年非対応）

OS	スレッド	リンクオプション
Windows	32 ビット，マルチスレッド	”インストール先フォルダ¥lib¥libJevApiA.lib” （32 ビットの VC++プロジェクト構成で実施）
	64 ビット，マルチスレッド	”インストール先フォルダ¥lib¥libJevApiAx64.lib” （64 ビットの VC++プロジェクト構成で実施）
AIX	32 ビット，シングルスレッド	/opt/jp1base/lib/libJevApiAst.a -ldl

OS	スレッド	リンクオプション
	32 ビット，マルチスレッド	/opt/jp1base/lib/libJevApiAmt.a -ldl -lpthread
	64 ビット，シングルスレッド	/opt/jp1base/lib/libJevApiAst64.a -q64 -ldl
	64 ビット，マルチスレッド	/opt/jp1base/lib/libJevApiAmt64.a -q64 -ldl -lpthread
Linux	32 ビット，シングルスレッド	/opt/jp1base/lib/libJevApiAst.a -ldl
	32 ビット，マルチスレッド	/opt/jp1base/lib/libJevApiAmt.a -ldl -lpthread
	64 ビット，シングルスレッド	/opt/jp1base/lib/libJevApiAst64.a -m64 -ldl
	64 ビット，マルチスレッド	/opt/jp1base/lib/libJevApiAmt64.a -m64 -ldl -lpthread

表 2-8　リンクオプション（2038 年対応）

OS	スレッド	リンクオプション
Windows	32 ビット，マルチスレッド	”インストール先フォルダ¥lib¥libJevApiAT.lib” （32 ビットの VC++プロジェクト構成で実施）
	64 ビット，マルチスレッド	”インストール先フォルダ¥lib¥libJevApiATx64.lib” （64 ビットの VC++プロジェクト構成で実施）
AIX	32 ビット，シングルスレッド	/opt/jp1base/lib/libJevApiATst.a -ldl
	32 ビット，マルチスレッド	/opt/jp1base/lib/libJevApiATmt.a -ldl -lpthread
	64 ビット，シングルスレッド	/opt/jp1base/lib/libJevApiATst64.a -q64 -ldl
	64 ビット，マルチスレッド	/opt/jp1base/lib/libJevApiATmt64.a -q64 -ldl -lpthread
Linux	32 ビット，シングルスレッド	/opt/jp1base/lib/libJevApiATst.a -ldl
	32 ビット，マルチスレッド	/opt/jp1base/lib/libJevApiATmt.a -ldl -lpthread
	64 ビット，シングルスレッド	/opt/jp1base/lib/libJevApiATst64.a -m64 -ldl
	64 ビット，マルチスレッド	/opt/jp1base/lib/libJevApiATmt64.a -m64 -ldl -lpthread

注意事項

- JP1/Base が提供するライブラリーは，スタティックライブラリー（UNIX の場合はアーカイブ）です。DLL 用のインポートライブラリーや共有ライブラリーではありません。
- JP1/Base が提供するライブラリーは，JP1/Base に同梱されている DLL（UNIX の場合は共有ライブラリー）をダイナミックロードします。このため，JP1/Base がインストールされていない環境でも，作成されたプログラムは動作しますが，その場合，関数が JEV_NO_LIBRARY で失敗します。
- Windows 用の JP1/Base 提供ライブラリーからダイナミックロードされる DLL は，Side by side アセンブリーでパッケージ化されたライブラリーとの依存関係がないため，マニフェストは提供していません。
- UNIX で JP1/Base 提供ライブラリーをリンクするときは，-l オプションでリンクしないでください。

- UNIX でリンクするときは，ld ではなく，コンパイラーと同じリンケージエディター（cc など）を使用することを推奨します。ld でリンクする場合は，コンパイラーが自動的に ld に引き渡すオプションと同じものを同じ順序で指定してください。
- x64 版の Linux 環境でコンパイルする場合，コンパイルオプションおよびリンクオプションに「-m32」を追加してください。
- プログラムを実行する環境とビルド（コンパイルおよびリンク）する環境が異なる場合，実行する環境にはビルドする環境よりも上位のバージョンまたは同じバージョンの OS や OS のパッチを適用してください。実行する環境の OS や OS のパッチのバージョンがビルドする環境よりも下位のバージョンである場合，シンボル解決エラーになり正常に起動できないなど，プログラムが正常に実行できない場合があります。
- JevGetRegistTime()関数，JevGetArrivedTime()関数を time_t 型の戻り値を返す関数として使用する場合，コンパイルオプションに「D_JEVTIME_T」を指定してください。

2.3　旧バージョンから移行する

旧バージョンの JP1/Base で作成したユーザーアプリケーションを移行する手順について説明します。

なお，ユーザーアプリケーションを 2038 年以降に使用しない場合は，ソースファイルの再コンパイルは不要です。詳細については，「2.3.3　2038 年対応」を参照してください。

2.3.1　再コンパイルしないで移行する

JP1/Base では，旧バージョンで作成したユーザーアプリケーションとのバイナリ互換性を保証しています。したがって，ユーザーアプリケーションを再コンパイルすることなく，最新の JP1/Base 上で動作します。

なお，ユーザーアプリケーションのバイナリ互換性を保証しているのは，実行環境の JP1/Base が開発環境の JP1/Base と同じバージョン，または上位バージョンの場合です。そのため，一つのユーザーアプリケーションを複数バージョンの JP1/Base で実行させる場合は，開発環境の JP1/Base を実行環境の最下位バージョンに合わせてください。

開発環境と実行環境とで JP1/Base のバージョンが異なる場合，バイナリ互換性の保証範囲の例を次の表に示します。

表 2-9　バイナリ互換性の保証範囲の例

開発環境	実行環境	ユーザーアプリケーションのバイナリ互換性
次のような開発環境で，ユーザーアプリケーションを開発した場合 • JP1/Base 09-10 • コンパイラー • ユーザーアプリケーション	JP1/Base が上位バージョンである場合 • JP1/Base 09-50 以降	○
	JP1/Base が同じバージョンである場合 • JP1/Base 09-10	○
	JP1/Base が下位バージョンである場合 • JP1/Base 09-00 以前	×

（凡例）

○：保証している。

×：保証していない。

重要

上記の表では，使用するコンパイラーで生成するユーザーアプリケーションの動作が保証されている OS のバージョンについて考慮していません。例えば，JP1/Base 09-10 でサポートしているコンパイラーで生成したユーザーアプリケーションが，JP1/Base 09-50 で新たにサポートされた OS 上で動作できるとは限りません。コンパイラーで生成するユーザーアプリケーショ

ンの動作が保証されている OS バージョンについては，コンパイラーのマニュアルなどで確認してください。

2.3.2　再コンパイルして移行する

JP1/Base では，旧バージョンで作成したソースコード互換性を保証しています。したがって，ユーザーアプリケーションのソースコードを変更することなく，再コンパイルすることで，最新の JP1/Base 上で動作します。

2.3.3　2038 年対応

2038 年以降も使用する場合は，2038 年対応ライブラリーをリンクしてください。

また，次に示す関数は，2038 年以降は使用できません。

- JevGetArrivedTime()関数
- JevGetRegistTime()関数

ソースファイルで上記の関数を使用している場合は，次のどちらかの方法で対処してください。

- 2038 年以降も使用できる関数に書き換える方法
  - JevGetArrivedTime()関数をJevGetArrivedTimeT()関数に書き換えてください。
  - JevGetRegistTime()関数をJevGetRegistTimeT()関数に書き換えてください。
- コンパイルオプションに「D_JEVTIME_T」を指定する方法

  JevGetRegistTime()関数またはJevGetArrivedTime()関数を使用している場合，time_t 型の戻り値を返す関数として使用するために，コンパイルオプションに「D_JEVTIME_T」を指定してください。

詳細については，「2.2.3　ソースファイルをコンパイルする」の 2038 年対応のライブラリー，2038 年対応のリンクオプション，および，注意事項でコンパイルオプションの「D_JEVTIME_T」について説明している個所を参照してください。

3 関数

この章では，JP1 イベントを発行および取得するために使用できる関数について説明します。

関数の記述形式

JP1 イベント発行関数や JP1 イベント取得関数の説明で使用する見出しについて説明します。

機能

関数の機能について説明します。

定義ヘッダー

関数を定義するヘッダーを示します。

形式

関数の記述形式を示します。

引数

関数の引数に指定できる値とその意味について説明します。

戻り値

関数の実行後に戻される値とその意味を示します。

注意事項

各関数を使用するときに注意が必要な事柄について説明します。なお，各関数で共通の注意事項については，次節の「各関数共通の注意事項」を参照してください。

各関数共通の注意事項

JP1/Base が提供する各関数に共通する注意事項を次に示します。

- Windows 版，UNIX 版ともに，マルチスレッドプログラムから呼ばれたときの動作を保証しています。
- Windows 版，UNIX 版のマルチスレッドプログラムで各関数を利用する場合，最初に呼び出す関数よりも前に生成されたスレッドで，各関数を利用することはできません。
- 物理ホストのイベントサーバ定義が FQDN 名のイベントサービス名だけの場合に，JP1 イベント取得関数を使用するときは，イベントサーバ名「*」（自ホスト名のイベントサーバ名）の定義を追加してください。下線部の追加する定義のディレクトリ名の記述は一致させてください。一致していない場合は動作を保証できません。また定義がない場合は，JP1 イベントの取得に失敗します。

  イベントサーバインデックスファイルの定義例

```
#----------------------------------------------------------------------
# JP1/Base - Event Server Index
#----------------------------------------------------------------------
server hostX.d1.hitachi.co.jp default
server * default
```

関数一覧

JP1/Base が提供する関数には JP1 イベント発行関数と JP1 イベント取得関数があります。JP1/Base が提供するこれらの関数とその機能について次の表に示します。なお，JP1 イベント発行関数，JP1 イベント取得関数で利用する JP1 イベントのイベント属性の詳細については，「付録 A　JP1 イベントの属性の設定基準」を参照してください。

表 3-1　JP1 イベント発行関数

機能	関数名
JP1/Base のイベントサーバに JP1 イベントを発行する。	JevRegistEvent

表 3-2　JP1 イベント取得関数

機能	関数名
JP1 イベントを取得するために JP1/Base のイベントサーバに接続する。	JevGetOpen
JP1 イベントを 1 件取得する。	JevGetEvent
JP1 イベントの基本属性(イベント DB 内の通し番号)を取得する。	JevGetSequenceNumber
JP1 イベントの基本属性(イベント ID の基本部)を取得する。	JevGetBaseID
JP1 イベントの基本属性(イベント ID の拡張部)を取得する。	JevGetExtID
JP1 イベントの基本属性(登録要因)を取得する。	JevGetRegistFactor
JP1 イベントの基本属性(発行元プロセス ID)を取得する。	JevGetProcessID
JP1 イベントの基本属性(登録時刻)を取得する。	• JevGetRegistTime（戻り値long型）※1 • JevGetRegistTime（戻り値time_t型）※2 • JevGetRegistTimeT 注※1 2038 年以降は使用できません。 注※2 2038 年以降も使用できます。
JP1 イベントの基本属性(到着時刻)を取得する。	• JevGetArrivedTime（戻り値long型）※1 • JevGetArrivedTime（戻り値time_t型）※2 • JevGetArrivedTimeT 注※1 2038 年以降は使用できません。 注※2 2038 年以降も使用できます。
JP1 イベントの基本属性(発行元ユーザー ID)を取得する。	JevGetRegistUserID
JP1 イベントの基本属性(発行元グループ ID)を取得する。	JevGetRegistGroupID
JP1 イベントの基本属性(発行元ユーザー名)を取得する。	JevGetRegistUserName

機能	関数名
JP1 イベントの基本属性(発行元グループ名)を取得する。	JevGetRegistGroupName
JP1 イベントの基本属性(発行元イベントサーバ名)を取得する。	JevGetSourceServer
JP1 イベントの基本属性(送信先イベントサーバ名)を取得する。	JevGetDestinationServer
JP1 イベントの基本属性(発行元 IP アドレス)を取得する。	JevGetSourceAddress
JP1 イベントの基本属性(送信先 IP アドレス)を取得する。	JevGetDestinationAddress
JP1 イベントの基本属性(発行元別通し番号)を取得する。	JevGetSourceSequenceNumber
JP1 イベントの基本属性(コードセット)を取得する。	JevGetCodeSet
JP1 イベントの基本属性(メッセージ)を取得する。	JevGetMessage
JP1 イベントの基本属性(詳細情報)を取得する。	JevGetDetailInformation
JP1 イベントの拡張属性を取得する。	JevGetExtAttrDirect
JP1 イベントの最初の拡張属性を取得する。	JevGetFirstExtAttr
JP1 イベントの次の拡張属性を取得する。	JevGetNextExtAttr
取得した JP1 イベント用のメモリーを解放する。	JevFreeEvent
イベントサーバから切断する。	JevGetClose

なお，各関数の詳細説明は，次の節以降でアルファベット順に記載しています。

JevFreeEvent

機能

JevGetEvent()関数で返された値で，アクセスできる JP1 イベントの格納領域を解放します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-3　状態コードと意味（JevFreeEvent）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

戻り値

正常終了	0 を返す。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetArrivedTime（戻り値 long 型）

機能

JP1 イベントの基本属性(到着時刻)を取得します。到着時刻は，UTC の 1970-01-01 00:00:00 からの通算秒で表されます。この関数は，2038 年以降は使用できません。取得した到着時刻が 2038/01/19 03:14:07 以降の場合，戻り値-1 を返します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-4　状態コードと意味（JevGetArrivedTime）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。
JEV_S_EXTATTR_EOD	この JP1 イベントにはこれ以上拡張属性は含まれていません。 （2038/01/19 03:14:07 以降の日付の場合，この状態コードを返します。）

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

戻り値

正常終了	指定されたハンドルで参照できる JP1 イベントの到着時刻(long 型)を返す。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

注意事項

この関数は，2038 年以降は使用できません。JevGetArrivedTimeT()関数またはJevGetArrivedTime()関数（戻り値 time_t 型）を使用してください。

JevGetArrivedTime（戻り値 time_t 型）

機能

JP1 イベントの基本属性(到着時刻)を取得します。到着時刻は，UTC の 1970-01-01 00:00:00 からの通算秒で表されます。time_t 型の戻り値を返します。

この関数を使用する場合，コンパイル時のコンパイルオプションに「D_JEVTIME_T」を指定してください。

定義ヘッダー

JevApi.h

形式

```
time_t JevGetArrivedTime(long* lplStatus,
                         JP1EVENT event);
```

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-5　状態コードと意味（JevGetArrivedTime）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

戻り値

正常終了	指定されたハンドルで参照できる JP1 イベントの到着時刻(time_t 型)を返す。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetArrivedTimeT

機能

JP1 イベントの基本属性(到着時刻)を取得します。到着時刻（time_t 型）は，UTC の 1970-01-01 00:00:00 からの通算秒で表されます。

定義ヘッダー

JevApi.h

形式

```
time_t JevGetArrivedTimeT(long* lplStatus,
                          JP1EVENT event);
```

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-6　状態コードと意味（JevGetArrivedTimeT）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

戻り値

正常終了	指定されたハンドルで参照できる JP1 イベントの到着時刻(time_t 型)を返す。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetBaseID

機能

JP1 イベントの基本属性(イベント ID の基本部)を取得します。イベント ID の基本部とは，イベント ID（8 バイト）の上位 4 バイトを指します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-7　状態コードと意味（JevGetBaseID）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

戻り値

正常終了	指定されたハンドルで参照できる JP1 イベントのイベント ID の基本部を返す。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetClose

機能

イベントサーバから切断し，JevGetOpen()関数の戻り値で返されたJP1イベント取得用ハンドルを閉じます。

JevGetOpen()関数の戻り値で返されたJP1イベント取得用ハンドルは，この関数を使って閉じる必要があります（Windowsでは，この関数を呼び出さずにプロセスが終了してしまうとシステムリソースリークとなります)。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-8　状態コードと意味（JevGetClose）

状態コード	意味
JEV_S_CONNECT_ERROR	イベントサービスに接続できません。
JEV_S_PARAM_ERROR	パラメーターが不正です。
JEV_S_MAXOPEN	ファイルオープン数が限界に達しました。
JEV_S_NOMEMORY	メモリー不足です。
JEV_S_IO_ERR	入出力エラーです。
JEV_S_SYSTEM_ERROR	システムエラーです（システムリソース不足です)。

key

JP1イベント取得用ハンドル(JevGetOpen()関数の戻り値)を指定します。

戻り値

正常終了	0を返す。
異常終了	-1を返す。また，失敗時の詳細エラーコードをlplStatusで指定された領域に格納する。

JevGetCodeSet

機能

JP1 イベントの基本属性(コードセット)を取得します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-9　状態コードと意味（JevGetCodeSet）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

lppszValue

取得したコードセットへのポインターを保存するための領域を示すポインターを指定します。該当するデータがない場合，NULL ポインターが設定されます。

戻り値

正常終了	0 を返す。また，コードセットへのポインターを lppszValue で指定された領域に格納する。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetDestinationAddress

機能

JP1 イベントの基本属性(送信先 IP アドレス)を取得します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-10　状態コードと意味（JevGetDestinationAddress）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

lpnSize

送信先 IP アドレス長を格納する領域へのポインターを指定します。なお，IPv6 環境で取得した JP1 イベントの送信先 IP アドレス長は，16 になります。

lppszValue

取得した送信先 IP アドレスへのポインターを保存するための領域を示すポインターを指定します。

戻り値

正常終了	0 を返し，送信先 IP アドレスへのポインターを lppszValue で指定された領域に格納する。また，送信先 IP アドレス長を lpnSize で指定された領域に格納する。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetDestinationServer

機能

JP1 イベントの基本属性(送信先イベントサーバ名)を取得します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-11　状態コードと意味（JevGetDestinationServer）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

lppszValue

取得した送信先イベントサーバ名へのポインターを保存するための領域を示すポインターを指定します。該当するデータがない場合，NULL ポインターが設定されます。

戻り値

正常終了	0 を返す。また，送信先イベントサーバ名へのポインターを lppszValue で指定された領域に格納する。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetDetailInformation

機能

JP1 イベントの基本属性(詳細情報)を取得します。

定義ヘッダー

JevApi.h

形式

```
int JevGetDetailInformation(long* lplStatus,
                            JP1EVENT event,
                            long* lplSize,
                            const char** const lppszValue);
```

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-12　状態コードと意味（JevGetDetailInformation）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

lplSize

詳細情報の長さを格納する領域へのポインターを指定します。

lppszValue

取得した詳細情報へのポインターを保存するための領域を示すポインターを指定します。該当するデータがない場合，NULL ポインターが設定されます。

戻り値

正常終了	0 を返し，詳細情報へのポインターを lppszValue で指定された領域に格納する。また，詳細情報の長さを lplSize で指定された領域に格納する。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetEvent

機能

JevGetOpen()関数で指定した条件に合致するJP1イベントを1件取得します。この関数を繰り返し呼び出すことによって，JevGetOpen()関数で指定したフィルターに合致するJP1イベントを，イベントDBに登録された順に取得できます。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-13　状態コードと意味（JevGetEvent）

状態コード	意味
JEV_S_CONNECT_ERROR	イベントサービスに接続できません。
JEV_S_INVALID_SERVER	イベントサーバ名が不正です。
JEV_S_PARAM_ERROR	パラメーターが不正です。
JEV_S_NO_EVENT	フィルターに一致するJP1イベントが存在しません。
JEV_S_MAXOPEN	ファイルオープン数が限界に達しました。
JEV_S_NOMEMORY	メモリー不足です。
JEV_S_IO_ERR	入出力エラーです。

key

JP1イベント取得用ハンドル(JevGetOpen()関数の戻り値)を指定します。

access

JP1イベントを取得するときに，該当するJP1イベントが存在しない場合の動作を指定するための値を指定します。値は，次に示すどちらかです。

JEVGET_WAIT

該当する JP1 イベントが発生するまで制御を戻しません。

JEVGET_NOWAIT

該当する JP1 イベントがなければ，すぐにエラーリターンします。

戻り値

正常終了	JP1 イベントアクセス用ハンドルを返す。
異常終了	NULL ポインターを返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetExtAttrDirect

機能

JP1 イベントの拡張属性を取得します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-14　状態コードと意味（JevGetExtAttrDirect）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。
JEV_S_NOT_DEFINED	指定した属性は未登録です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

lpszName

拡張属性名を指定する文字列へのポインターを指定します。

戻り値

正常終了	指定されたハンドルで参照できる JP1 イベントの拡張属性値を返す。
異常終了	NULL ポインターを返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetExtID

機能

JP1 イベントの基本属性(イベント ID の拡張部)を取得します。イベント ID の拡張部とは，イベント ID（8 バイト）の下位 4 バイトを指します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-15　状態コードと意味（JevGetExtID）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

戻り値

正常終了	指定されたハンドルで参照できる JP1 イベントのイベント ID の拡張部を返す。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetFirstExtAttr

機能

JP1 イベントの最初に指定されている拡張属性を取得します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-16　状態コードと意味（JevGetFirstExtAttr）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。
JEV_S_EXTATTR_EOD	この JP1 イベントにはこれ以上拡張属性は含まれていません。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

lppszName

取得した拡張属性名へのポインターを保存するための領域を示すポインターを指定します。

lppszValue

取得した拡張属性値へのポインターを保存するための領域を示すポインターを指定します。該当するデータがない場合，NULL ポインターが設定されます。

戻り値

正常終了	0 を返し，拡張属性名へのポインターを lppszName で指定された領域に格納する。また，拡張属性値へのポインターを lppszValue で指定された領域に格納する。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetMessage

機能

JP1 イベントの基本属性(メッセージ)を取得します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-17　状態コードと意味（JevGetMessage）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

lppszValue

取得したメッセージへのポインターを保存するための領域を示すポインターを指定します。該当するデータがない場合，NULL ポインターが設定されます。

戻り値

正常終了	0 を返す。また，メッセージへのポインターを lppszValue で指定された領域に格納する。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetNextExtAttr

機能

JP1 イベント取得関数（JevGetFirstAttr()関数またはJevGetNextAttr()関数）を使って取得した拡張属性の次に指定されている JP1 イベントの拡張属性を取得します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-18　状態コードと意味（JevGetNextExtAttr）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。
JEV_S_EXTATTR_EOD	この JP1 イベントにはこれ以上拡張属性は含まれていません。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

lppszName

取得した拡張属性名へのポインターを保存するための領域を示すポインターを指定します。

lppszValue

取得した拡張属性値へのポインターを保存するための領域を示すポインターを指定します。

戻り値

正常終了	0 を返し，次の拡張属性名へのポインターを lppszName で指定された領域に格納する。また，次の拡張属性値へのポインターを lppszValue で指定された領域に格納する。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetOpen

機能

JP1 イベントを取得するために JP1/Base のイベントサーバに接続します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-19　状態コードと意味（JevGetOpen）

状態コード	意味
JEV_NO_LIBRARY	ライブラリーが見つかりません※1，またはファイルオープン数が限界に達しシェアードライブラリーが検索できません。
JEV_S_CONNECT_ERROR	イベントサービスに接続できません。
JEV_S_PARAM_ERROR	パラメーターが不正です。
JEV_S_MAXOPEN	ファイルオープン数が限界に達しました。
JEV_S_NOMEMORY	メモリー不足です。
JEV_S_IO_ERR	入出力エラーです。
JEV_S_SYSTEM_ERROR	システムエラーです（システムリソース不足です）。
JEV_S_NO_AUTHORITY	JP1 プログラムまたはユーザーアプリケーションにはイベントサーバに接続する権限がありません※2。
JEV_S_FILTER_ERROR	フィルターに誤りがあります（正規表現の誤り部分を除きます）。
JEV_S_REGEX_ERROR	フィルター中で指定している正規表現に誤りがあります。
JEV_S_REGEX_CANNOY_USED	正規表現ライブラリーが使用できません。

注※1　誤って必要なファイルを削除したか，またはコンパイルオプションに誤りがあるおそれがあります。必要なファイルを削除していた場合は，JP1/Base を再インストールしてください。コンパイルオプションが誤っていた場合は，再設定してください。

注※2　イベントサーバに接続する権限は，JP1/Base のイベントサーバ設定ファイル（conf）のusers パラメーターで設定してください。

lpszServer

接続先イベントサーバ名を表す「¥0」で終わる文字列へのポインターを指定します。NULL ポインターを指定すると，自ホスト名と同じ名称のイベントサーバに接続します。イベントサーバ名は，「¥0」を含めて256 バイト以下で指定してください。

lpszFilter

マニュアル「JP1/Base 運用ガイド」の，フィルターの文法の項で説明しているフィルターを示す「¥0」で終わる文字列へのポインターを指定します。NULL ポインターを指定した場合，すべての JP1 イベントが取得対象になります。

lPosition

JP1 イベントの取得開始位置をイベント DB 通し番号で指定します。

-1 が指定された場合は，この関数の発行以降に登録された JP1 イベントを取得できます。なお，この関数の実行中に発生したイベントは，取得対象にならないことがあります。そのため，取得が保証されるのは，この関数の完了以降に登録された JP1 イベントとなります。

戻り値

正常終了	JP1 イベント取得用ハンドルを返す。
異常終了	NULL ポインターを返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetProcessID

機能

JP1 イベントの基本属性(発行元プロセス ID)を取得します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-20　状態コードと意味（JevGetProcessID）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

戻り値

正常終了	指定されたハンドルで参照できる JP1 イベントの発行元プロセス ID を返す。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetRegistFactor

機能

JP1 イベントの基本属性(登録要因)を取得します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-21　状態コードと意味（JevGetRegistFactor）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

戻り値

正常終了	指定されたハンドルで参照できる JP1 イベントの登録要因を返す。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetRegistGroupID

機能

JP1 イベントの基本属性(発行元グループ ID)を取得します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-22　状態コードと意味（JevGetRegistGroupID）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

lplSize

発行元グループ ID を格納する領域へのポインターを指定します。

戻り値

正常終了	0 を返す。また，発行元グループ ID を lplSize で指定された領域に格納する。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetRegistGroupName

機能

JP1 イベントの基本属性(発行元グループ名)を取得します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-23　状態コードと意味（JevGetRegistGroupName）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

lppszValue

取得した発行元グループ名へのポインターを保存するための領域を示すポインターを指定します。該当するデータがない場合，NULL ポインターが設定されます。

戻り値

正常終了	0 を返す。また，発行元グループ名へのポインターを lppszValue で指定された領域に格納する。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetRegistTime（戻り値 long 型）

機能

JP1 イベントの基本属性(登録時刻)を取得します。登録時刻は UTC の 1970-01-01 00:00:00 からの通算秒で表されます。この関数は，2038 年以降は使用できません。取得した到着時刻が 2038/01/19 03:14:07 以降の場合，戻り値-1 を返します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-24　状態コードと意味（JevGetRegistTime）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。
JEV_S_EXTATTR_EOD	この JP1 イベントにはこれ以上拡張属性は含まれていません。 （2038/01/19 03:14:07 以降の日付の場合，この状態コードを返します。）

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

戻り値

正常終了	指定されたハンドルで参照できる JP1 イベントの登録時刻(long 型)を返す。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

注意事項

この関数は，2038 年以降は使用できません。JevGetRegistTimeT()関数またはJevGetRegistTime()関数（戻り値 time_t 型）を使用してください。

JevGetRegistTime（戻り値 time_t 型）

機能

JP1 イベントの基本属性(登録時刻)を取得します。登録時刻は UTC の 1970-01-01 00:00:00 からの通算秒で表されます。time_t 型の戻り値を返します。

この関数を使用する場合，コンパイル時のコンパイルオプションに「D_JEVTIME_T」を指定してください。

定義ヘッダー

JevApi.h

形式

```
time_t JevGetRegistTime(long* lplStatus,
                        JP1EVENT event);
```

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-25　状態コードと意味（JevGetRegistTime）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル（JevGetEvent()関数の戻り値）を指定します。

戻り値

正常終了	指定されたハンドルで参照できる JP1 イベントの登録時刻(time_t 型)を返す。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetRegistTimeT

機能

JP1 イベントの基本属性(登録時刻)を取得します。登録時刻（time_t 型）は UTC の 1970-01-01 00:00:00 からの通算秒で表されます。

定義ヘッダー

JevApi.h

形式

```
time_t JevGetRegistTimeT(long* lplStatus,
                         JP1EVENT event);
```

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-26　状態コードと意味（JevGetRegistTimeT）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル（JevGetEvent()関数の戻り値）を指定します。

戻り値

正常終了	指定されたハンドルで参照できる JP1 イベントの登録時刻(time_t 型)を返す。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetRegistUserID

機能

JP1 イベントの基本属性(発行元ユーザー ID)を取得します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-27　状態コードと意味（JevGetRegistUserID）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

lplSize

発行元ユーザー ID を格納する領域へのポインターを指定します。

戻り値

正常終了	0 を返す。また，発行元ユーザー ID を lplSize で指定された領域に格納する。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetRegistUserName

機能

JP1 イベントの基本属性(発行元ユーザー名)を取得します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-28　状態コードと意味（JevGetRegistUserName）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

lppszValue

取得した発行元ユーザー名へのポインターを保存するための領域を示すポインターを指定します。該当するデータがない場合，NULL ポインターが設定されます。

戻り値

正常終了	0 を返す。また，発行元ユーザー名へのポインターを lppszValue で指定された領域に格納する。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetSequenceNumber

機能

JP1 イベントの基本属性(イベント DB 内通し番号)を取得します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-29　状態コードと意味（JevGetSequenceNumber）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

戻り値

正常終了	指定されたハンドルで参照できる JP1 イベントのイベント DB 内通し番号を返す。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetSourceAddress

機能

JP1 イベントの基本属性(発行元 IP アドレス)を取得します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-30　状態コードと意味（JevGetSourceAddress）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

lpnSize

発行元 IP アドレス長を格納する領域へのポインターを指定します。なお，IPv6 環境で取得した JP1 イベントの発行元 IP アドレス長は，16 になります。

lppszValue

取得した発行元 IP アドレスへのポインターを保存するための領域を示すポインターを指定します。

戻り値

正常終了	0 を返し，発行元 IP アドレスへのポインターを lppszValue で指定された領域に格納する。また，発行元 IP アドレス長を lpnSize で指定された領域に格納する。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetSourceSequenceNumber

機能

JP1 イベントの基本属性(発行元別通し番号)を取得します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-31　状態コードと意味（JevGetSourceSequenceNumber）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

戻り値

正常終了	指定されたハンドルで参照できる JP1 イベントの発行元別通し番号を返す。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevGetSourceServer

機能

JP1 イベントの基本属性(発行元イベントサーバ名)を取得します。

定義ヘッダー

JevApi.h

形式

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

引数

lplStatus

この関数の異常終了時の状態コードを返す領域へのポインターを指定します。状態コードとその意味は次のとおりです。

表 3-32　状態コードと意味（JevGetSourceServer）

状態コード	意味
JEV_S_PARAM_ERROR	パラメーターが不正です。

event

JP1 イベントアクセス用ハンドル(JevGetEvent()関数の戻り値)を指定します。

lppszValue

取得した発行元イベントサーバ名へのポインターを保存するための領域を示すポインターを指定します。

戻り値

正常終了	0 を返す。また，発行元イベントサーバ名へのポインターを lppszValue で指定された領域に格納する。
異常終了	-1 を返す。また，失敗時の詳細エラーコードを lplStatus で指定された領域に格納する。

JevRegistEvent

機能

JP1/Base のイベントサーバに JP1 イベントを発行します。この関数が正常終了した時点で，自イベントサーバでの JP1 イベントの受付が完了していることが保証されます。

定義ヘッダー

JevApi.h

形式

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

引数

status

この関数の異常終了時の状態コードを返す領域のアドレスを指定します。状態コードとその意味は次のとおりです。

表 3-33　状態コードと意味（JevRegistEvent）

状態コード	意味
JEV_NO_LIBRARY	ライブラリーが見つかりません※，またはファイルオープン数が限界に達しシェアードライブラリーが検索できません。
JEV_S_CONNECT_ERROR	イベントサービスに接続できません。
JEV_S_INVALID_ID	イベント ID が不正です。
JEV_S_INVALID_SERVER	イベントサーバ名が不正です。
JEV_S_INVALID_EXT_NAME	拡張属性名が不正です。
JEV_S_OVER_EXT_COUNT	拡張属性数が上限を超えています。
JEV_S_OVER_EXT_SIZE	拡張属性の合計サイズが上限を超えています。
JEV_S_OVER_MESSAGE	メッセージの長さが上限を超えています。
JEV_S_PARAM_ERROR	パラメーターが不正です。
JEV_S_NOT_SUPPORT	サポートされていません。
JEV_S_MAXOPEN	ファイルオープン数の限界に達しました。

状態コード	意味
JEV_S_NOMEMORY	メモリー不足です。
JEV_S_IO_ERR	入出力エラーです。
JEV_S_SYSTEM_ERROR	システムエラーです。

注※　誤って必要なファイルを削除したか，またはコンパイルオプションに誤りがあるおそれがあります。必要なファイルを削除していた場合は，JP1/Base を再インストールしてください。コンパイルオプションが誤っていた場合は，再設定してください。

server

自ホストで稼働する接続先イベントサーバ名を表す「¥0」で終わる文字列へのポインターを指定します。NULL ポインターを指定すると，自ホスト名と同じ名称のイベントサーバに接続します。イベントサーバ名は，「¥0」を含めて 256 バイト以下で指定してください。

baseID

登録するイベント ID の基本部を表す数値を指定します。指定できる値の範囲は，0x00000000，0x00000001～0x00001FFF，または 0x7FFF8000～0x7FFFFFFF です。

message

登録するメッセージを表す「¥0」で終わる文字列へのポインターを指定します。メッセージは，「¥0」を含めて 1,024 バイト以下で指定してください。

extattrs

登録する拡張属性文字列を格納した文字列配列を指定します。配列の中の個々の拡張属性文字列は，「拡張属性名=拡張属性値」の形式の，「¥0」で終わる文字列です。

拡張属性名は，属性の意味を表す文字列です。拡張属性名には，英数字またはアンダーライン（_）から成る最大 32 バイトの文字列を指定できます。英字は大文字だけを使用できます。文字列の先頭は英字で指定してください。

拡張属性値は，属性の内容を表す 0～10,000 バイトの文字列です。

拡張属性は，最大 100 個指定できます。すべての属性値の長さの合計（「¥0」を含まない）は，10,000 バイトに制限されます。

引数として NULL ポインターを指定した場合，拡張属性は登録されません。

拡張属性の詳細については，「付録 A　JP1 イベントの属性の設定基準」を参照してください。

extcount

登録する拡張属性の個数を指定します。「拡張属性」に NULL ポインターが指定された場合，この値は無視されます。

戻り値

正常終了	0 を返す。
異常終了	-1 を返す。

注意事項

拡張属性を指定するときに，同一の拡張属性名を持つ文字列を複数指定した場合は，最後に指定した文字列が有効になります。

付録

付録 A　JP1 イベントの属性の設定基準

独自イベントを発行する際は，次の説明で示す基準に基づいてイベント属性を設定してください。また，JP1 イベントを取得する際は，次の説明で示す基準を参考にして，その属性を取得するかどうか検討してください。

付録 A.1　基本属性

独自イベントを発行する際に利用する基本属性は，イベント ID とメッセージです。そのほかの基本属性については，主に JP1 イベントを取得する際の参考にしてください。

表 A-1　JP1 イベントの基本属性

属性の項目	説明
イベント DB 内の通し番号	発行元によらないでこのイベントサーバに到達した順番（ローカルイベントも含む）。この属性は JP1 イベントのイベントサーバ間の転送時に保存されない。主に JP1 イベントをユーザーアプリケーションが取得したときやほかのイベントサーバへ転送したときの漏れ・重複の防止に用いる。
イベント ID	発行元のアプリケーションや事象の内容を表す 8 バイトの値。各日立プログラムおよびユーザーアプリケーションには，イベントの ID の範囲が割り当てられている。ユーザーアプリケーションに指定できる値の範囲は，0～0x1FFF および 0x7FFF8000～0x7FFFFFFF である。イベント ID は，システム全体でユニーク性を保持できるように割り当てる必要がある。なお，メッセージ ID の 8 バイトのうち上位 4 バイトが基本部，下位 4 バイトが拡張部となっている。
登録要因	JP1 イベントがこのイベントサーバに登録された要因。この属性は JP1 イベントのイベントサーバ間転送時に保存されない。登録要因を次に示す。 1：自イベントサーバから自イベントサーバあての発行 2：自イベントサーバから他イベントサーバあての発行 3：他イベントサーバから自イベントサーバあての発行 4：環境設定の指定による他イベントサーバから自イベントサーバへの転送
プロセス ID	発行元アプリケーションプログラムのプロセス ID。
登録時刻	発行元イベントサーバでの登録時刻（発行元ホストの時計に基づく。UTC の 1970-01-01 00:00:00 からの秒数）。
到着時刻	自イベントサーバでの登録時刻（UTC の 1970-01-01 00:00:00 からの秒数）。この属性は JP1 イベントのイベントサーバ間転送時に保存されない。
発行元ユーザー ID	発行元プロセスのユーザー ID。Windows と Java では環境設定による固定値（-1～65,535）。
発行元グループ ID	発行元プロセスのグループ ID。Windows と Java では環境設定による固定値（-1～65,535）。
発行元ユーザー名	発行元プロセスのユーザー名。
発行元グループ名	発行元プロセスのグループ名。Windows と Java では NULL 文字列。
発行元イベントサーバ名	発行元のイベントサーバ名。JP1 イベントが転送された場合でもこの JP1 イベントが発生したホストのイベントサーバ名が入る。

属性の項目	説明
送信先イベントサーバ名	発行元アプリケーションが他イベントサーバへの転送を明示して指定した場合に，他イベントサーバの名称が入る。
発行元 IP アドレス	発行元イベントサーバに対応する IP アドレス（NAT やプロキシを経由した場合および環境設定で転送した JP1 イベントについては正確な値ではない）。
送信先 IP アドレス	送信先イベントサーバに対応する IP アドレス（NAT やプロキシを経由した場合および環境設定で転送した JP1 イベントについては正確な値ではない）。
発行元別通し番号	発行元ホストでのイベント DB 内通し番号（転送によって値は変化しない）。
コードセット	メッセージ・詳細情報・拡張属性が記述されている文字コードセット名。
メッセージ	メッセージは次の規則に基づいて指定する。 • 事象の内容をわかりやすく説明する。 • 改行コードを含めないで 1 行で書く。
詳細情報	任意のデータ。

付録 A.2　拡張属性

JP1 イベントの拡張属性には，共通情報と固有情報があります。

(1)　共通情報

表 A-2　JP1 イベントの拡張属性（共通情報）

属性名	項目	内容
SEVERITY	重大度	次に示す重大度がある。 Emergency，Alert，Critical，Error，Warning，Notice，Information，Debug 重大度の詳細については，「表 A-3　重大度の意味」を参照のこと。
USER_NAME	ユーザー名	ユーザー名。業務を実行しているユーザーの名前。
PRODUCT_NAME	プロダクト名	プロダクト名。プロダクト名は，スラント「/」で区切られた半角英数字列となる。次のどちらかの形式に従い，会社ごとにユニークな文字列となるようにする。 /会社名/シリーズ名/プロダクト名 または /会社名/プロダクト名 なお，会社名「HITACHI」は予約されているため，使用できない。
OBJECT_TYPE	オブジェクトタイプ	オブジェクトタイプには，イベント発行元のオブジェクトの種類を指定する。基本的には，次のオブジェクトタイプが登録されている。このタイプを使用して JP1 イベントのフィルタリングおよび検索を実行するため，同じ意味を持つ JP1 イベントは，できるだけ同じオブジェクトタイプとして扱えるように指定する。

属性名	項目	内容
		新しいオブジェクトタイプを追加する場合は，ユニークな名称を指定する。オブジェクトタイプを追加した場合は，オブジェクトタイプ定義ファイルを作成する必要がある。 JOB：ジョブ JOBNET：ジョブネット ACTION：アクション ACTIONFLOW：アクションフロー PRINTJOB：印刷ジョブ PRINTQUEUE：プリントキュー PRINTER：プリンタ BATCHQUEUE：バッチキュー PIPEQUEUE：パイプキュー JOBBOX：ジョブボックス LOGFILE：ログファイル LINK：リンク（下位層の通信レイヤーの事象通知用） SERVICE：サービス（デーモンプロセスなど） PRODUCT：プロダクト（プログラム固有のそのほかの事象通知用） CONFIGRATION：構成定義 SERVER：サーバ BACKUP：バックアップ RESTORE：リストア MEDIA：メディア
OBJECT_NAME	オブジェクト名	オブジェクト名。オブジェクトタイプを特定するための名称を指定する。例えば，オブジェクトタイプが「JOB」の場合はジョブ名称などを指定する。
ROOT_OBJECT_TYPE	登録名タイプ	登録名タイプ。オブジェクトタイプの親オブジェクトのタイプを指定する。これは，オブジェクトが階層を持つ場合などに有効である。例えば，オブジェクトタイプが「JOB」の場合，ルートオブジェクトタイプは「JOBNET」となる。ルートオブジェクトタイプが存在しない場合は，オブジェクトタイプと同じ種別を指定すること。基本的には，オブジェクトタイプと同じ値が定義されている。
ROOT_OBJECT_NAME	登録名	登録名。ルートオブジェクトタイプを特定するための名称を指定する。例えば，ジョブネット名称などを指定する。
OBJECT_ID	オブジェクト ID	オブジェクト ID。 PRODUCT_NAME との組み合わせによってオブジェクトのインスタンスを統合システム内で一意に意識できる文字列（形式は他製品に依存する。この情報は JP1/IM の［統合機能メニュー］画面から各製品のモニターを呼び出すときに使用する）。
OCCURRENCE	事象種別	特定のオブジェクトについて，イベントの発行契機となる事象を設定する。基本的には，次の事象種別がある。この事象種別とオブジェクトタイプによってフィルター設定などをすることで，特定のオブジェクトの特定の事象を選択できるようになる。 ACTIVE：アクティブになった

属性名	項目	内容
		INACTIVE：非アクティブになった START：開始した END：終了した NOTSTART：開始できなかった CANCEL：キャンセルされた LATESTART：開始予定時刻を過ぎた LATEEND：終了予定時刻を過ぎた SUBMIT：サブミットされた UNSUBMIT：サブミットが取り消された ENQUEUE：キューに登録された DEQUEUE：キューから削除された PAUSE：一時停止（保留） RELEASE：一時停止の解除（保留解除） RESTART：再実行を開始した CREATE：作成された DELETE：削除された MODIFY：更新された RETRY：リトライを開始した STOP：停止中 MOVE：移動した COPY：コピーした NOTICE：通知した（オペレーターなどへの通知結果） REPLY：応答された CONNECT：接続した DISCONNECT：切り離した EXCEPTION：そのほかのエラーが発生した
START_TIME	開始時刻	実行開始または再実行開始の時刻。UTC 1970 年 1 月 1 日 00:00:00 からの秒数で指定する。事象種別が START，RESTART，PAUSE，RELEASE，または END の場合だけに指定する。
END_TIME	終了時刻	実行終了の時刻。UTC 1970 年 1 月 1 日 00:00:00 からの秒数で指定する。事象種別が END の場合だけに指定する。
RESULT_CODE	終了コード	10 進数文字列の終了コード。事象種別が END の場合だけに指定する。
PLATFORM※	－	イベント拡張属性定義ファイル，モニター画面呼び出し定義ファイルなどでプラットフォーム種別として指定する任意の文字列を指定する。省略した場合は，「base」が指定されたものとして扱われる。
ACTION_VERSION※	－	モニター画面呼び出しのときのバージョン情報として使用する。複数バージョンが混在していて，それぞれの場合に呼び出す画面などが異なる場合に必要である。指定しない場合は，モニター画面呼び出し定義でバージョン指定を使用しないこと。

注※

これらの属性は JP1/IM のイベント詳細情報には表示されません。ただし，JP1/IM でイベント拡張属性定義ファイルに定義した場合は JP1/IM のイベント詳細情報に表示されます。

「重大度」の意味

拡張属性の共通情報「重大度」の意味を次の表に示します。独自イベントの「重大度」を設定するときには，ここに示す基準を指針としてください。なお，発行時に「重大度」が指定されていないイベントは，JP1/IM の［イベントコンソール］画面に表示されません。

表 A-3　重大度の意味

重大度	表示名称	意味
Emergency	緊急	パニック状態。通常すべてのユーザーにブロードキャストするもの
Alert	警戒	システムやデータベースの破壊など，直ちに修復が必要な状態
Critical	致命的	ハードデバイスのエラーなどの危険な状態
Error	エラー	エラー
Warning	警告	警告メッセージ
Notice	通知	エラー状態ではないが，特別な扱いが必要な状態
Information	情報	通知メッセージ
Debug	デバッグ	通常，プログラムのデバッグ時だけに使用する情報を含むメッセージ。メッセージ量の問題が発生するので，JP1 イベント発行の対象としない

(2)　固有情報

共通情報のほかに，JP1 イベントに付加して利用価値のある情報をプログラムごとに固有情報として設定します。拡張属性の固有情報を設定した場合は，JP1/IM でイベント拡張属性定義ファイルに定義する必要があります。

固有情報を設定するときの規則を次に示します。

- 属性名は意味のない記号名でもよい。
- 属性名と意味は，同じ「PRODUCT_NAME」の拡張属性を持つ各プログラムで 1 対 1 で対応するようにする。

付録 B　サンプルソースファイル

JP1/Base では，C で作成したファイルを，次に示すサンプルソースファイルとして提供しています。

- sender.c
- receiver.c

これらのサンプルソースファイルは，次に示すディレクトリに格納されています。必要に応じてご利用ください。

Windows：インストール先フォルダ¥tools¥event¥

UNIX：/opt/jp1base/tools/event/

付録 B.1　サンプルソースファイルの詳細

サンプルソースファイルの詳細について説明します。

(1)　サンプルソースファイルで扱うイベント

表 B-1　サンプルソースファイルで扱うイベント

属性の種別	項目	属性名	内容
基本属性	イベント ID	−	0x00000001
	メッセージ	−	「Starts the SAMPLE application.」
拡張属性（共通情報）	重大度	SEVERITY	Notice
	ユーザー名	USER_NAME	アプリケーション実行ユーザー名
	プロダクト名	PRODUCT_NAME	/COMPANY/APP1/SAMPLE_PRODUCT(プロダクトの名称)
	オブジェクトタイプ	OBJECT_TYPE	SAMPLE
	オブジェクト名	OBJECT_NAME	SAMPLE_NAME
	登録名タイプ	ROOT_OBJECT_TYPE	ROOT_SAMPLE
	登録名	ROOT_OBJECT_NAME	ROOT_SAMPLE_NAME
	オブジェクト ID	OBJECT_ID	SAMPLE_ID
	事象種別	OCCURRENCE	START
	プラットフォーム	PLATFORM	NT
	バージョン情報	ACTION_VERSION	0600

属性の種別	項目	属性名	内容
拡張属性（固有情報）	SAMPLE 共通属性 1	COMMON_ATTR1	NATIVE
	SAMPLE 共通属性 2	COMMON_ATTR2	TRUE
	SAMPLE 開始属性 1	START_ATTR1	SAMPLE1
	SAMPLE 開始属性 2	START_ATTR2	SAMPLE2

(2) サンプルソースファイルのコーディング内容

(a) sender.c のコーディング内容

```
#include <stdio.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 */
      const char* extattrs[15];  /* 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";
      extattrs[9] = "PLATFORM=NT";
      extattrs[10] = "VERSION=0600";
      extattrs[11] = "COMMON_ATTR1=NATIVE";
      extattrs[12] = "COMMON_ATTR2=TRUE";
      extattrs[13] = "START_ATTR1=SAMPLE1";
      extattrs[14] = "START_ATTR2=SAMPLE2";

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

```
                        message,
                        extattrs,
                        15);
      if(rc < 0) {
          fprintf(stderr,
                  "JevRegistEvent() failed. status = %ld¥n",
                   status);
          return -1;
      }

      return 0;
}

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

(b) receiver.c のコーディング内容

```
#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();
}
```

索引

R

S

い

か

き

こ

さ

し

そ

と

り

株式会社 日立製作所

〒 100-8280　東京都千代田区丸の内一丁目 6 番 6 号