JFP リファレンスマニュアル 3 : C ライブラリ関数

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 817–0188–10
2002 年 12 月

Adobe PostScript

021031@4879

目次

はじめに

概要

SunOS リファレンスマニュアルは、初めて SunOS を使用するユーザーやすでにある程度の知識を持っているユーザーのどちらでも対応できるように解説されています。このマニュアルを構成するマニュアルページは一般に参照マニュアルとして作られており、チュートリアルな要素は含んでいません。それぞれのコマンドを実行すると、どのような結果が得られるかについて、詳しく説明されています。なお、各マニュアルページの内容はオンラインでも参照することができます。

このマニュアルは、マニュアルページの内容によっていくつかのセクションに分かれています。各セクションについて以下に簡単に説明します。

- セクション 1 は、オペレーティングシステムで使えるコマンドを説明します。
- セクション 1M は、システム保守や管理用として主に使われるコマンドを説明します。
- セクション 2 は、すべてのシステムコールについて説明します。ほとんどのシステムコールに 1 つまたは複数のエラーがあります。エラーの場合、通常ありえない戻り値が返されます。
- セクション 3 は、さまざまなライブラリ中の関数について説明します。ただし、UNIX システムプリミティブを直接呼び出す関数については、セクション 2 で説明しています。
- セクション 4 は、各種ファイルの形式について説明します。また、ファイル形式を宣言する C 構造体を適用できる場合には随時説明しています。
- セクション 5 は、文字セットテーブルなど他のセクションには該当しないものについて説明します。
- セクション 7 は、特殊なハードウェア周辺装置またはデバイスドライバに関するさまざまな特殊ファイルについて説明します。STREAMS ソフトウェアドライバ、モジュール、またはシステムコールの STREAMS 汎用セットについても説明します。

以下に、このマニュアルの項目を表記されている順に説明します。ほとんどのマニュアルページが下記の項目からなる共通の書式で書かれていますが、必要でない項目については省略されています。たとえば、記述すべきバグがコマンドにない場合などは、「使用上の留意点」という項目はありません。各マニュアルページの詳細は各セクションの `intro` を、マニュアルページの一般的な情報については `man`(1) を参照してください。

名前　コマンドや関数の名称と概略が示されています。

形式　コマンドや関数の構文が示されています。標準パスにコマンドやファイルが存在しない場合は、フルパス名が示されます。字体は、コマンド、オプションなどの定数にはボールド体 (bold) を、引数、パラメータ、置換文字などの変数にはイタリック体 (Italic) または <日本語訳> を使用しています。オプションと引数の順番は、アルファベット順です。特別な指定が必要な場合を除いて、1 文字の引数、引数のついたオプションの順に書かれています。

以下の文字がそれぞれの項目で使われています。

- [ ]　このかっこに囲まれたオプションや引数は省略できます。このかっこが付いていない場合には、引数を必ず指定する必要があります。
- . . .　省略符号。前の引数に変数を付けたり、引数を複数指定したりできることを意味します (例: ‘ `filename` . . .’)。
- |　区切り文字 (セパレータ)。この文字で分割されている引数のうち 1 つだけを指定できます。
- { }　この大かっこに囲まれた複数のオプションや引数は省略できます。かっこ内を 1 組として扱います。

プロトコル　この項が使われているのは、プロトコルが記述されているファイルを示すサブセクション 3R だけです。パス名は常にボールド体 (bold) で示されています。

機能説明　コマンドの機能とその動作について説明します。実行時の詳細を説明していますが、オプションの説明や使用例はここでは示されていません。対話形式のコマンド、サブコマンド、リクエスト、マクロ、関数などに関しては「使用法」で説明します。

IOCTL　セクション 7 だけに使用される項です。`ioctl`(2) システムコールへのパラメータは `ioctl` と呼ばれ、適切なパラメータを持つデバイスクラスのマニュアルページだけに記載されています。特定のデバイスに関する `ioctl` は、(そのデバイスのマニュアルページに) アルファベット順に記述されています。デバイスの特定のクラスに関する `ioctl` は、`mtio`(7I) のように `io` で終わる名前が付いているデバイスクラスのマニュアルページに記載されています。

オプション	各オプションがどのように実行されるかを説明しています。「形式」で示されている順に記述されています。オプションの引数はこの項目で説明され、必要な場合はデフォルト値を示します。
オペランド	コマンドのオペランドを一覧表示し、各オペランドがコマンドの動作にどのように影響を及ぼすかを説明しています。
出力	コマンドによって生成される出力 (標準出力、標準エラー、または出力ファイル) を説明しています。
戻り値	値を返す関数の場合、その値を示し、値が返される時の条件を説明しています。関数が 0 や -1 のような一定の値だけを返す場合は、値と説明の形で示され、その他の場合は各関数の戻り値について簡単に説明しています。void として宣言された関数はこの項では扱いません。
エラー	失敗の場合、ほとんどの関数はその理由を示すエラーコードを `errno` 変数の中に設定します。この項ではエラーコードをアルファベット順に記述し、各エラーの原因となる条件について説明します。同じエラーの原因となる条件が複数ある場合は、エラーコードの下にそれぞれの条件を別々のパラグラフで説明しています。
使用法	この項では、使用する際の手がかりとなる説明が示されています。特定の決まりや機能、詳しい説明の必要なコマンドなどが示されています。組み込み機能については、以下の小項目で説明しています。 コマンド 修飾子 変数 式 入力文法
使用例	コマンドや関数の使用例または使用方法を説明しています。できるだけ実際に入力するコマンド行とスクリーンに表示される内容を例にしています。例の中には必ず `example%` のプロンプトが出てきます。スーパーユーザーの場合は `example#` のプロンプトになります。例では、その説明、変数置換の方法、戻り値が示され、それらのほとんどが「形式」、「機能説明」、「オプション」、「使用法」の項からの実例となっています。
環境	コマンドや関数が影響を与える環境変数を記述し、その影響について簡単に説明しています。
終了ステータス	コマンドが呼び出しプログラムまたはシェルに返す値と、その状態を説明しています。通常、正常終了には 0 が返され、0 以外の値はそれぞれのエラー状態を示します。

ファイル	マニュアルページが参照するファイル、関連ファイル、およびコマンドが作成または必要とするファイルを示し、各ファイルについて簡単に説明しています。
属性	属性タイプとその対応する値を定義することにより、コマンド、ユーティリティ、およびデバイスドライバの特性を一覧しています。詳細は `attributes`(5) を参照してください。
関連項目	関連するマニュアルページ、当社のマニュアル、および一般の出版物が示されています。
診断	エラーの発生状況と診断メッセージが示されています。メッセージはボールド体 (bold) で、変数はイタリック体 (Italic) または <日本語訳> で示されており、C ロケール時の表示形式です。
警告	作業に支障を与えるような現象について説明しています。診断メッセージではありません。
注意事項	それぞれの項に該当しない追加情報が示されています。マニュアルページの内容とは直接関係のない事柄も参照用に扱っています。ここでは重要な情報については説明していません。
使用上の留意点	すでに発見されているバグについて説明しています。可能な場合は対処法も示しています。

JFP リファレンスマニュアル 3 : C ライブラリ関数

Intro_jfp(3)

名前 | Intro_jfp, intro_jfp – JFP 関数およびライブラリの序章

機能説明 | 本セクションでは、さまざまなライブラリにある関数のうち、直接 UNIX システムのプリミティブを呼び出す関数 (セクション 2 のマニュアルページで説明) 以外の、JFP で提供する特殊ライブラリ について説明します。 関数宣言は、各マニュアルページに示してある `#include` ファイルから取り込むことができます。

定義 | 文字とは、マシンのバイトに適合できる ビットパターンを指します (ただし、言語によっては“文字” は 1 バイト以上を必要とし、 複数バイトで表現されます)。

`NULL` 文字とは、値が `0` の文字を指し、 C 言語では便宜上 `\0` と表されます。 文字配列とは、文字の並び (シーケンス) を指します。`NULL` 終了文字配列 (文字列) とは、最後の文字が `NULL` 文字である文字の並びを指します。`NULL` 文字列とは、終了 `NULL` 文字だけを含む文字配列を指します。 `NULL` ポインタとは、`0` をポインタにキャストすることによって 得られる値を指します。C 言語では、 `NULL` の値は有効なポインタとマッチしません。このため、ポインタを返す関数のほとんどは、`NULL` を返すことによってエラーを示します。`NULL` マクロは `<stdio.h>` で定義されています。`size_t` の型は適切なヘッダーの中に定義されています。

MT レベルのライブラリ | MT レベルのライブラリの詳細については `attributes`(5) を参照してください。

ファイル

INCDIR	通常、`/usr/include`
LIBDIR	通常、`/usr/lib` (32 ビット) または `/usr/lib/sparcv9` (64 ビット)

LIBDIR`/libci.so`

LIBDIR`/libci.a`

LIBDIR`/libcics.so`

LIBDIR`/libcics.a`

LIBDIR`/libci.so.1`

LIBDIR`/libcics.so.1`

関連項目 | `ar`(1), `cc`(1B), `ld`(1), `nm`(1),

`intro`(2),

`intro`(3), `stdio`(3C)

`libadm`(3LIB), `libc`(3LIB), `libelf`(3LIB), `libdl`(3LIB), `libkvm`(3LIB), `libmapmalloc`(3LIB), `libmp`(3LIB), `libnsl`(3LIB), `librac`(3LIB), `libresolv`(3LIB) , `librpcsvc`(3LIB), `libsocket`(3LIB), `libpthread`(3LIB), `libthread`(3LIB), `libxfn`(3LIB), `libxnet`(3LIB)

`attributes`(5), `standards`(5)

『リンカーとライブラリ』

『プロファイリングツール』

ANSI C Programmer's Guide

診断

浮動小数点の値を返す関数の場合、エラー処理はコンパイルのモードによって変わります。cc に -Xt オプションを指定した場合 (デフォルト)、指定された引数が使用する関数が定義されていないとき、または値を表現できないとき、これらの関数は 0、±HUGE、または NaN という一定の値を返します。-Xa および -Xc コンパイルモードでは、±HUGE の代わりに ±HUGE_VAL が返されます (HUGE_VAL および HUGE は math.h の中で定義されています。HUGE_VAL は無限を表し、 HUGE は単精度小数点の最大値を表します)。

マルチスレッドアプリケーションに関する注意事項

マルチスレッドアプリケーションをコンパイルする際は、_POSIX_C_SOURCE フラグ、_POSIX_PTHREAD_SEMANTICS フラグ、_REENTRANT フラグのいずれかのフラグを、コマンド行で定義しなければなりません。これにより、マルチスレッドのアプリケーションだけに適用される関数の定義が特別に有効になります。POSIX.1003.1c に適合するアプリケーションに対しては、_POSIX_C_SOURCE フラグに 199506L 以上の値を定義してください。

cc [*flags* **]***file...* **-D_POSIX_C_SOURCE=199506L -lpthread**

Solaris の fork() と fork1() 関数を使用した POSIX では、 次のようにコンパイルしてください。

cc [*flags* **]***file...* **-D_POSIX_PTHREAD_SEMANTICS -lthread**

Solaris では、 次のようにコンパイルしてください。

cc [*flags* **]***file...* **-D_REENTRANT -lthread**

シングルスレッドのアプリケーションを作成する際、上記のフラグを定義してはいけません。定義しないことにより、 マルチスレッドをサポートしていない旧リリースの Solaris 上で実行可能なバイナリが生成されます。

アプリケーションを安全に実行するために、Unsafe インタフェースはメインスレッド以外から呼び出さないでください。

MT-Safe インタフェースは、各関数やライブラリのマニュアルページの「属性」の項に記述されています (attributes(5) を参照)。インタフェースが MT-Safe であることがマニュアルページ中に明示されていないときは、そのインタフェースは Unsafe であるとみなしてください。

リアルタイムアプリケーション

早期のバインディングを可能にするためには、非 NULL 値に対して、環境変数 LD_BIND_NOW を設定してください。詳細については、『リンカーとライブラリ』中の再配置に関する説明を参照してください。

注意事項

ここで説明する関数、外部変数、マクロは、ユーザープログラムの中で再定義しないでください。他の名前を再定義してもこれらのライブラリの動作に影響を与えませんが、同じ名前を再定義すると、インクルードされているヘッダーの宣言と衝突してしまいます。

INCDIR の中にあるヘッダーは、 本マニュアルページで説明するほとんどの関数のプロトタイプを提供します (関数のプロトタイプとは、引数の型を含む関数宣言のことです)。関数のプロトタイプを使用すると、ユーザーのプログラムの中で使用されている上記関数の使用法が間違っていないか、コンパイラでチェックすることができます。

`lint` プログラムチェッカを使用すると、ヘッダーが `#include` 文によってインクルードされていないという矛盾をチェックすることができます。セクション 2、3C、3S の定義は自動的にチェックされます。その他の定義をチェックする場合、`lint` に `-l` オプションを指定します (たとえば、`libm` の定義をチェックするには `-lm` を指定します)。できるだけ `lint` を使用することをお勧めします。詳細は、『プロファイリングツール』の `lint` に関する章を参照してください。

STREAMS と *stream* の違いには十分気を付けてください。 STREAMS とは、カーネルの機構の一種であり、 ネットワークサービスとデータ通信ドライバの開発をサポートします。 STREAMS は、ユーティリティルーチン、カーネル機能、データ構造体で構成されています。*stream* とは、STREAMS に関連するバッファリングで使用されるファイルを指します。*stream* は `<stdio.h>` の中で `FILE` 型へのポインタとして定義されています。

各要素の詳細な定義では、実装に特有なシンボリック名を参照しなければならない場合があります。ただし、必ずしもアプリケーションプログラムで使用できるようにする必要はありません。これらのシンボリック名の多くは、境界の条件やシステムの制限を記述するものです。

本セクションでは、分かりやすくするために、このような実装固有の値をシンボリック名にしています。このようなシンボリック名は必ず中括弧 ({ }) で囲まれ、ヘッダーを使ってアプリケーションからアクセスできる他の実装に固有の定数のシンボリック名と区別されます。 中括弧で囲まれているシンボリック名は、特定のシステムの文書中で定義されていることもありますが、ヘッダーを使用することによってアプリケーションプログラムから必ずしもアクセスできるわけではありません。

一般には、移植可能なアプリケーションプログラムは、コードの中で、このようなシンボリック名を参照してはいけません。たとえば、あるアプリケーションプログラムは、あるルーチンに提供された引数リストの長さが {`ARG_MAX`} より大きいかどうかをテストしてはなりません。

ライブラリ一覧

名前	説明
`Intro_jfp`(3)	JFP 関数およびライブラリの序章
`wctrans_ja`(3C)	日本語ロケール用のワイド文字変換
`wctype_ja`(3C)	日本語ロケールの文字クラスの定義

名前 | wctrans_ja – 日本語ロケール用のワイド文字変換

形式 |
```
#include <wchar.h>

wctrans_t wctrans(const char *property);
```

機能説明 | `wctrans()` 関数は、*property* 引数によって指示した内容に従って、ワイド文字間の変換を可能にするための `wctrans_t` 型の値を構築します。実際の変換は `towctrans()` 関数を使用します。`wctrans()` は、`towctrans()` 関数で必要な引数を返します。

次の文字クラス名は、すべてのロケールに定義されています。

```
tolower
toupper
```

この他に、日本語ロケール (ja、ja_JP.eucJP、ja_JP.PCK、および ja_JP.UTF-8) では、次の日本語ロケール専用文字クラスを定義しています。

```
tojhira
tojkata
tojisx0208
tojisx0201
```

これらもまた、`wctrans()` 関数の *property* 引数として利用できます。ただし、これらを使用した場合は、日本語ロケール専用のアプリケーションになります。

`tolower` アルファベットの小文字を表すワイド文字への変換を指示します。

`toupper` アルファベットの大文字を表すワイド文字への変換を指示します。

`tojhira` JIS X 0208 で定義されるカタカナ文字に対して、ひらがな文字への変換を指示します。

`tojkata` JIS X 0208 で定義されるひらがな文字に対して、カタカナ文字への変換を指示します。

`tojisx0201` JIS X 0201 ローマ文字用図形キャラクタ集合または片仮名用図形キャラクタ集合の文字に対して、対応する JIS X 0208 文字への変換を指示します。

`tojisx0208` JIS X 0208 の文字に対して、対応する JIS X 0201 ローマ文字用図形キャラクタ集合または 片仮名用図形キャラクタ集合の文字への変換を指示します。

使用例 | ワイド文字 `wc` をひらがな文字へ変換する例を示します。

```
towctrans(wc, wctrans("tojhira"))
```

関連項目 | `towctrans`(3C), `wctrans`(3C), `wctype_ja`(3C), `PCK`(5), `eucJP`(5)

名前 wctype_ja – 日本語ロケールの文字クラスの定義

形式

```
#include <wchar.h>

wctype_t wctype(const char *charclass);
```

機能説明 `wctype()` 関数は、*charclass* 引数によって指示した内容に従って、ワイド文字クラスを判定するための `wctype_t` 型の値を構築します。実際の判定は `iswctype()` 関数を使用します。`wctype()` は、`wctype()` 関数で必要な引数を返します。

次の文字クラス名は、すべてのロケールに定義されています。

```
alnum       alpha       blank       cntrl
digit       graph       lower       print
punct       space       upper       xdigit
```

この他に、日本語ロケール (ja、ja_JP.eucJP、ja_JP.PCK、および ja_JP.UTF-8) では、次の日本語ロケール専用文字クラスを定義しています。

```
jkanji       jkata       hira           jdigit
jparen       line           jisx0201r       jisx0208
jisx0212       udc           vdc
```

以下は ja、ja_JP.eucJP ロケールでのみ使用できます。

```
jalpha          jspecial       jgreek       jrussian
junit           jsci           jgen          jpunct
```

以下は、ja_JP.eucJP、ja_JP.UTF-8 ロケールでのみ使用できます。

```
ascii           paren           jisx0201
gaiji           jhankana           jspace
```

これらもまた、`wctype()` 関数の *charclass* 引数として利用できます。ただし、これらを使用した場合は、日本語ロケール専用のアプリケーションになります。

`upper` 任意の大文字を表す文字クラス

JIS X 0201 ローマ文字用図形キャラクタ集合	アルファベット大文字 (C/1–D/10)
JIS X 0208	ローマ字大文字 (3/33–3/58)
	ギリシャ文字大文字 (6/1–24)
	ロシア文字大文字 (7/1–33)
JIS X 0212	ダイアクリティカルマーク付きギリシャアルファベット文字大文字 (6/65–69、71、73、74、76)
	キリル系アルファベット大文字 (7/34-46)

		ラテン系アルファベット大文字 (9/1、2、4、6、8、9、11、12、13、15、16)
		ダイアクリティカルマーク付きラテン系アルファベット大文字 (10/01–24、26–87)
lower	任意の小文字を表す文字クラス	
	JIS X 0201 ローマ文字用図形キャラクタ集合	アルファベット小文字 (E/1–F/10)
	JIS X 0208	ローマ字小文字 (3/65–90)
		ギリシャ文字小文字 (6/33–56)
		ロシア文字小文字 (7/49–81)
	JIS X 0212	ダイアクリティカルマーク付きギリシャアルファベット文字小文字 (6/81–92)
		キリル系アルファベット小文字 (7/82–94)
		ラテン系アルファベット小文字 (9/33–48)
		ダイアクリティカルマーク付きラテン系アルファベット小文字 (11/1–27、29–35、37–87)
digit	10 進表現に用いる 0 から 9 までの数字を判別するクラス	
	JIS X 0201 ローマ文字用図形キャラクタ集合	数字 (B/0–9)
space	空白を判別するクラス	
	JIS X 0201 制御キャラクタ集合	空白 (A/9–13) 間隔文字
	JIS X 0208	間隔 (1/1)
punct	記号、特殊文字などを判別するクラス	

JIS X 0201 ローマ文字用図形キャラクタ集合　A/1–15、B/10–C/0、D/11–E/0、F/11–14

`cntrl`　制御文字を判別するクラス

JIS X 0201 制御キャラクタ集合　すべての文字

抹消文字

C1 制御文字　すべての文字

`blank`　フィールド区切り文字を判別するクラス

JIS X 0201 制御キャラクタ集合　A/9 間隔文字

JIS X 0208　間隔 (1/1)

`xdigit`　16 進表現に用いる英数字を判別するクラス

JIS X 0201 ローマ文字用図形キャラクタ集合　数字 (B/0–9)、A–F、 a–f (C/1–6、 E/1–6)

`alpha`　アルファベットを判別するクラス

`upper` クラスと `lower` クラスの文字

`print`　表示可能文字を判別するクラス

JIS X 0201 ローマ文字用図形キャラクタ集合　間隔文字

	JIS X 0201 片仮名用図形キャラクタ集合　文字未定義領域以外のすべての文字
	JIS X 0208　文字未定義領域以外のすべての文字
	JIS X 0212　文字未定義領域以外のすべての文字
	ベンダー定義文字領域　vdc クラスのうち、文字未定義領域以外のすべての文字
	ユーザー定義文字領域　udc クラスのうち、文字未定義領域も含むすべての文字
graph	図形文字を判別するためのクラス
	print クラスから space クラスに含まれる文字を除いたすべての文字
jkanji	漢字 (漢字表記のために使用する記号、 表意文字) を判別するクラス
	JIS X 0208　16 区 から 84 区の文字定義領域
	JIS X 0212　16 区 から 77 区の文字定義領域
jkata	片仮名を判別するクラス
	JIS X 0208　5/1–86、1/11、12、19、20
jhira	平仮名を判別するクラス
	JIS X 0208　4/1–83、1/11、12、21、22、26
jdigit	digit に含まれる以外の数字を判別するクラス
	JIS X 0208　3/16–25
jparen	括弧などに用いるための文字を判別するクラス
	JIS X 0208　1/38–59
line	罫線素片を判別するためのクラス
	JIS X 0208　8/1–32

`jisx0201` JIS X 0201 片仮名用図形キャラクタ集合に属する文字を判別するクラス

JIS X 0201 片仮名用図形キャラクタ集合	A/1 から D/15 までのすべての文字	

`jisx0208` JIS X 0208 に属する文字を判別するクラス

JIS X 0208 文字未定義領域の文字を含むすべての文字。ただし、1 区から 84 区まで (13 区ベンダー定義文字領域もここに含まれる)

`jisx0212` JIS X 0212 に属する文字を判別するクラス

JIS X 0212 文字未定義領域の文字を含むすべての文字。ただし、1 区から 84 区まで (83、84 区ベンダー定義文字領域もここに含まれる)。ja_JP.PCK ロケールではどの文字もこのクラスに属さない

`udc` ユーザー定義文字を判別するクラス

ユーザー定義文字領域の文字未定義領域も含むすべての文字

ja、ja_JP.eucJP ロケール	ユーザー定義文字 (1–20 区)	0xf5a1–0xfefe 0x8ff5a1–0x8ffefe
ja_JP.PCK ロケール	ユーザー定義文字 (1–20 区)	0xf040–0xf9fc
ja_JP.UTF-8 ロケール	ユーザー定義文字 (6400 文字)	0xe000–0xf8ff

`vdc` ベンダー定義文字を判別するクラス

ベンダー定義文字領域の文字未定義領域も含むすべての文字

ja、ja_JP.eucJP ロケール	JIS X 0208 13 区: 特殊記号 JIS X 0212 83 区–84 区 JIS X 0212 に含まれない IBM 拡張文字
ja_JP.PCK ロケール	JIS X 0208 13 区: 特殊記号 NEC 選定 IBM 拡張文字 0xed40–0xeffc

		IBM 拡張文字: 0xfa40–0xfcfc
	ja_JP.UTF-8 ロケール	定義なし
jalpha	英字を表す文字を判別するクラス	
	JIS X 0208	3/33–58、 3/65–90
jspecial	特殊文字を表す文字を判別するクラス	
	JIS X 0208	1/2–94、 2/1–14、 2/26–33、 2/42–48、 2/60–74、 2/82–89、 94
	JIS X 0212	2/15–25、 2/34–36、 2/75–81
	JIS X 0208 13 区: 特殊記号	IBM 拡張文字 NEC 選定 IBM 拡張文字で定義される特殊文字
jgreek	ギリシャ文字を判別するクラス	
	JIS X 0208	6/1–24、 6/33–56
jrussian	ロシア文字を判別するクラス	
	JIS X 0208	7/1–7/33、7/49–81
junit	単位記号を判別するクラス	
	JIS X 0208	1/75–83、2/82、83
	JIS X 0212	2/80
jsci	学術記号を判別するクラス	
	JIS X 0208	1/60–74、2/26–33、2/42–48、2/60–74
jgen	一般記号を判別するクラス	
	JIS X 0208	1/84–94、2/1–14、2/84–89、94
	JIS X 0212	2/35、75、2/79–81
jpunct	記述記号を判別するクラス	

JIS X 0208	1/2–37
JIS X 0212	2/34、36

`ascii` JIS X 0201 機能キャラクタ集合、間隔文字、ローマ文字用図形キャラクタ集合、抹消文字を判別するクラス

`paren` 対にして用いる記号を判別するクラス

`jisx0201` JIS X 0201 に属する文字を判別するクラス

`gaiji` 実装者定義文字を判別するクラス。`udc` と `vdc` クラスを含む

`jhankana` JIS X 0201 に属する文字のうち日本語の表記に使用する文字を判別するクラス

`jspace` JIS X 0208、JIS X 0212 に属する文字のうち、空白文字を判別するクラス

JIS X 0201 機能キャラクタ集合、ローマ文字用図形キャラクタ集合、片仮名用図形キャラクタ集合での *XX/YY* は *XX* 列 *YY* 行を意味します。JIS X 0208、JIS X 0212 での *XX/YY* は *XX* 区 *YY* 点を意味します。

JIS X 0212 の文字に関しては、ja、ja_JP.eucJP、または ja_JP.UTF-8 ロケールのみ該当します。

使用例

ワイド文字 `wc` が `udc` クラスに属するかどうかを判定する例を示します。

```
iswctype(wc, wctype("udc"))
```

関連項目

`iswctype`(3C), `wctype`(3C), `wctrans_ja`(3C), `eucJP`(5), `PCK`(5)