Contents

Preface 5

JFP Reference Manual 3 : Library Routines 11
Intro_ifp(3) 12
wctrans_ja(3C) 15
wctype_ja(3C) 16
Preface

Both novice users and those familiar with the SunOS operating system can use online man pages to obtain information about the system and its features. A man page is intended to answer concisely the question “What does it do?” The man pages in general comprise a reference manual. They are not intended to be a tutorial.

Overview

The following contains a brief description of each man page section and the information it references:

- Section 1 describes, in alphabetical order, commands available with the operating system.
- Section 1M describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes.
- Section 2 describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value.
- Section 3 describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2.
- Section 4 outlines the formats of various files. The C structure declarations for the file formats are given where applicable.
- Section 5 contains miscellaneous documentation such as character-set tables.
- Section 6 contains available games and demos.
- Section 7 describes various special files that refer to specific hardware peripherals and device drivers. STREAMS software drivers, modules and the STREAMS-generic set of system calls are also described.
Section 9 provides reference information needed to write device drivers in the kernel environment. It describes two device driver interface specifications: the Device Driver Interface (DDI) and the Driver/Kernel Interface (DKI).

Section 9E describes the DDI/DKI, DDI-only, and DKI-only entry-point routines a developer can include in a device driver.

Section 9F describes the kernel functions available for use by device drivers.

Section 9S describes the data structures used by drivers to share information between the driver and the kernel.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the intro pages for more information and detail about each section, and man(1) for more information about man pages in general.

NAME

This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS

This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full path name is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ] Brackets. The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.

... Ellipses. Several values can be provided for the previous argument, or the previous argument can be specified multiple times, for example, "filename ...".

| Separator. Only one of the arguments separated by this character can be specified at a time.

{ } Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.
PROTOCOL
This section occurs only in subsection 3R to indicate the protocol description file.

DESCRIPTION
This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES. Interactive commands, subcommands, requests, macros, and functions are described under USAGE.

IOCTL
This section appears on pages in Section 7 only. Only the device class that supplies appropriate parameters to the ioctl(2) system call is called ioctl and generates its own heading. ioctl calls for a specific device are listed alphabetically (on the man page for that specific device). ioctl calls are used for a particular class of devices all of which have an io ending, such as mtio(7I).

OPTIONS
This section lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.

OPERANDS
This section lists the command operands and describes how they affect the actions of the command.

OUTPUT
This section describes the output – standard output, standard error, or output files – generated by the command.

RETURN VALUES
If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.

ERRORS
On failure, most functions place an error code in the global variable errno indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than
one condition can cause the same error, each condition is described in a separate paragraph under the error code.

**USAGE**

This section lists special rules, features, and commands that require in-depth explanations. The subsections listed here are used to explain built-in functionality:

- Commands
- Modifiers
- Variables
- Expressions
- Input Grammar

**EXAMPLES**

This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command-line entry and machine response is shown. Whenever an example is given, the prompt is shown as example%, or if the user must be superuser, example#. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS, and USAGE sections.

**ENVIRONMENT VARIABLES**

This section lists any environment variables that the command or function affects, followed by a brief description of the effect.

**EXIT STATUS**

This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion, and values other than zero for various error conditions.

**FILES**

This section lists all file names referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.

**ATTRIBUTES**

This section lists characteristics of commands, utilities, and device drivers by defining the attribute type and its corresponding value. See attributes(5) for more information.

**SEE ALSO**

This section lists references to other man pages, in-house documentation, and outside publications.
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIAGNOSTICS</td>
<td>This section lists diagnostic messages with a brief explanation of the condition causing the error.</td>
</tr>
<tr>
<td>WARNINGS</td>
<td>This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.</td>
</tr>
<tr>
<td>NOTES</td>
<td>This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.</td>
</tr>
<tr>
<td>BUGS</td>
<td>This section describes known bugs and, wherever possible, suggests workarounds.</td>
</tr>
</tbody>
</table>
NAME
Intro_jfp, intro_jfp – introduction to JFP functions and libraries

DESCRIPTION
This section describes JFP functions found in various libraries, other than those functions that directly invoke UNIX which are described in Section 2 of man pages section 1: User Commands. Function declarations can be obtained from the #include files indicated on each page.

DEFINITIONS
A character is any bit pattern able to fit into a byte on the machine.

The null character is a character with value 0, conventionally represented in the C language as \0. A character array is a sequence of characters. A null-terminated character array (a string) is a sequence of characters, the last of which is the null character. The null string is a character array containing only the terminating null character. A NULL pointer is the value that is obtained by casting 0 into a pointer. C guarantees that this value will not match that of any legitimate pointer, so many functions that return pointers return NULL to indicate an error. The macro NULL is defined in <stdio.h>. Types of the form size_t are defined in the appropriate headers.

MT-Level of Libraries
See attributes(5) for descriptions of library MT-Levels.

FILES
INCDIR usually, /usr/include

LIBDIR usually, /usr/lib (32-bit) or /usr/lib/sparcv9 (64-bit)

LIBDIR/libci.so
LIBDIR/libci.a
LIBDIR/libcics.so
LIBDIR/libcics.a
LIBDIR/libci.so.1
LIBDIR/libcics.so.1

SEE ALSO
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)

Linker and Libraries Guide
For functions that return floating-point values, error handling varies according to compilation mode. Under the -Xt (default) option to cc these functions return the conventional values 0, ±HUGE or NaN when the function is undefined for the given arguments or when the value is not representable. In the -Xa and -Xc compilation modes, ±HUGE_VAL is returned instead of ±HUGE. (HUGE_VAL and HUGE are defined in math.h to be infinity and the largest-magnitude single-precision number, respectively.)

When compiling a multithreaded application, either the _POSIX_C_SOURCE, _POSIX_PTHREAD_SEMANTICS, or _REENTRANT flag must be defined on the command line. This enables special definitions for functions only applicable to multithreaded applications. For POSIX.1c-conforming applications, define the _POSIX_C_SOURCE flag to be >= 199506L:

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

For POSIX behavior with the Solaris fork() and fork1() distinction, compile as follows:

```bash
cc [flags] file... -D_POSIX_PTHREAD_SEMANTICS -lthread
```

For Solaris behavior, compile as follows:

```bash
cc [flags] file... -D_REENTRANT -lthread
```

When building a singlethreaded application, the above flag should be undefined. This generates a binary that is executable on previous Solaris releases, which do not support multithreading.

Unsafe interfaces should be called only from the main thread to ensure the application’s safety.

MT-Safe interfaces are denoted in the NOTES section of the functions and libraries man pages. If a man page does not state explicitly that an interface is MT-Safe, the user should assume that the interface is unsafe.

Be sure to have set the environment variable LD_BIND_NOW to a non-NULL value to enable early binding. Refer to the “When Relocations are Performed” chapter in Linker and Libraries Guide for additional information.

None of the functions, external variables, or macros should be redefined in the user’s programs. Any other name may be redefined without affecting the behavior of other library functions, but such redefinition may conflict with a declaration in an included header.
The headers in INCDIR provide function prototypes (function declarations including the types of arguments) for most of the functions listed in this manual. Function prototypes allow the compiler to check for correct usage of these functions in the user’s program.

The lint program checker may also be used and will report discrepancies even if the headers are not included with #include statements. Definitions for Sections 2, 3C, and 3S are checked automatically. Other definitions can be included by using the -l option to lint. (For example, -lm includes definitions for libm.) Use of lint is highly recommended. See the lint chapter in Profiling Tools.

Users should carefully note the difference between STREAMS and stream. STREAMS is a set of kernel mechanisms that support the development of network services and data communication drivers. It is composed of utility routines, kernel facilities, and a set of data structures. A stream is a file with its associated buffering. It is declared to be a pointer to a type FILE defined in <stdio.h>.

In detailed definitions of components, it is sometimes necessary to refer to symbolic names that are implementation-specific, but which are not necessarily expected to be accessible to an application program. Many of these symbolic names describe boundary conditions and system limits.

In this section, for readability, these implementation-specific values are given symbolic names. These names always appear enclosed in curly brackets to distinguish them from symbolic names of other implementation-specific constants that are accessible to application programs by headers. These names are not necessarily accessible to an application program through a header, although they may be defined in the documentation for a particular system.

In general, a portable application program should not refer to these symbolic names in its code. For example, an application program would not be expected to test the length of an argument list given to a routine to determine if it was greater than {ARG_MAX}.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intro_jfp(3)</td>
<td>introduction to JFP functions and libraries</td>
</tr>
<tr>
<td>wctrans_ja(3C)</td>
<td>Wide character conversion for the Japanese locale</td>
</tr>
<tr>
<td>wctype_ja(3C)</td>
<td>Define a character class for the Japanese locale</td>
</tr>
</tbody>
</table>
NAME

wctrans_ja – Wide character conversion for the Japanese locale

SYNOPSIS

#include <wchar.h>

wctrans_t wctrans(const char *property);

descrIption

wctrans() builds values in wctrans_t data type according to the specification with
the property argument to allow conversion between wide characters. towctrans() is
used for actual conversion. wctrans() returns arguments that towctrans() needs
to use.

The following character class names are defined in every locale.

tolower
toupper

In addition to the above, the Japanese locale (ja, ja_JPPCK and ja_JP.UTF-8) defines the
following character classes specific to the Japanese locale.

tojhira
tojkata
tojisx0208
tojisx0201

These can be also used as property arguments to wctrans(). However, the use of
these classes are limited to applications for the Japanese locale only.

tolower
Specifications conversion to lowercase alphabet wide characters.
toupper
Specifications conversion to uppercase alphabet wide characters.
tojhira
Specifications conversion of JIS X 0208 Katakana to Hiragana.
tojkata
Specifications conversion of JIS X 0208 Hiragana to Katakana.
tojisx0208
Specifications conversion of JIS X 0201 Roman character graphic set or
Katakana character graphic set to the associated JIS X 0208
characters.
tojisx0201
Specifications conversion of JIS X 0208 characters to the associated JIS X
0201 Roman character graphic set or Katakana character graphic
set.

Example

The following shows an example to convert a wide character wc to Hiragana.

towctrans(wc, wctrans("tojhira"))

See Also
towctrans(3C), wctrans(3C), wctype_ja(3C), PCK(5), eucJP(5)
NAME
wctype_ja – Define a character class for the Japanese locale

SYNOPSIS
#include <wchar.h>

wctype_t wctype(const char *charclass);

DESCRIPTION
wctype() builds values in wctype_t data type according to the specification with
the charclass argument to determine wide character classes. iswctype() is used for
actual determination. wctype() returns arguments that wctype() needs to use.

The following character class names are defined in every locale.

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

In addition to the above, the Japanese locale (ja, ja_JP.eucJP, ja_JP.PCK and ja_JP.UTF-8)
defines the following character classes specific to the Japanese locale.

jkanji jkata hira jdigit
jparen line jisx0201r jisx0208
jisx0212 udc vdc

The following character classes are supported in ja and ja_JP.eucJP locales only.

jalpha jspecial jgreek jrussian
junit jsci jgen jpunct

The following character classes are supported in ja_JP.eucJP and ja_JP.UTF-8 locale
only.

ascii paren jisx0201
gaiji jhankana jspace

These can be also used as charclass arguments to wctype(). However, the use of these
classes are limited to applications for the Japanese locale only.

upper Character class that represents any uppercase letter

JIS X Alphabet uppercase letters (C/1–D/10)
0201 Roman character
0208 Greek character uppercase letters (6/1–24)

JIS X Roman character uppercase letters (3/33–3/58)
0208 Greek character uppercase letters (6/1–24)

JIS X Greek alphabet uppercase letters with diacritical marks
0212 (6/65–69, 71, 73, 74, 76)
<table>
<thead>
<tr>
<th>lower</th>
<th>Character class that represents any lowercase letter</th>
</tr>
</thead>
<tbody>
<tr>
<td>JIS X 0201</td>
<td>Roman character lowercase letters (3/65–90)</td>
</tr>
<tr>
<td></td>
<td>Greek character lowercase letters (6/33–56)</td>
</tr>
<tr>
<td>JIS X 0208</td>
<td>Russian character lowercase letters (7/49–81)</td>
</tr>
<tr>
<td>JIS X 0212</td>
<td>Greek alphabet lowercase letters with diacritical marks (6/81–92)</td>
</tr>
<tr>
<td></td>
<td>Cyrillic alphabet lowercase letters (7/82–94)</td>
</tr>
<tr>
<td></td>
<td>Latin alphabet lowercase letters (9/33–48)</td>
</tr>
<tr>
<td></td>
<td>Latin alphabet lowercase letters with diacritical marks (11/1–27, 29–35, 37–87)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>digit</th>
<th>Class that determines the numbers 0 to 10 for decimal representation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>JIS X 0201</td>
<td>Numbers (B/0–9)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>space</th>
<th>Class that determines a space.</th>
</tr>
</thead>
<tbody>
<tr>
<td>JIS X 0201</td>
<td>Space (A/9–13)</td>
</tr>
<tr>
<td></td>
<td>Space characters</td>
</tr>
<tr>
<td>JIS X 0208</td>
<td>Space (1/1)</td>
</tr>
</tbody>
</table>
### punct

Class that determines symbols and special characters.

<table>
<thead>
<tr>
<th>JIS X 0201 Roman character graphic set</th>
</tr>
</thead>
<tbody>
<tr>
<td>A/1–15, B/10–C/0, D/11–E/0, F/11–14</td>
</tr>
</tbody>
</table>

### cntrl

Class that determines control characters.

<table>
<thead>
<tr>
<th>JIS X 0201 Control character set</th>
</tr>
</thead>
<tbody>
<tr>
<td>All characters</td>
</tr>
</tbody>
</table>

| Kill characters                  |
| C1 All characters               |
| control characters              |

### blank

Class that determines field delimiters.

<table>
<thead>
<tr>
<th>JIS X 0201 Control character set</th>
</tr>
</thead>
<tbody>
<tr>
<td>A/9 Space characters</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JIS X 0208 Space (1/1)</th>
</tr>
</thead>
</table>

### xdigit

Class that determines alphanumerics used for hexadecimal representation.

<table>
<thead>
<tr>
<th>JIS X 0201 Roman character graphic set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numbers (B/0–9)</td>
</tr>
<tr>
<td>A–E, a–f (C/1–6, E/1–6)</td>
</tr>
</tbody>
</table>

### alpha

Class that determines alphabets.

| upper class and lower class letters |

### print

Class that determines printable characters.

<p>| JIS X 0201 Roman character Space characters graphic set |</p>
<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JIS X 0201 Katakana</td>
<td>All the characters except in character undefined areas</td>
</tr>
<tr>
<td>JIS X 0208</td>
<td>All the characters except in character undefined areas</td>
</tr>
<tr>
<td>JIS X 0212</td>
<td>All the characters except in character undefined areas</td>
</tr>
<tr>
<td>Vendor-defined character</td>
<td>All the characters except in character undefined areas in Class vdc.</td>
</tr>
<tr>
<td>User-defined character</td>
<td>All the characters including character undefined areas in Class udc.</td>
</tr>
<tr>
<td>graph</td>
<td>Class that determines graphic characters.</td>
</tr>
<tr>
<td></td>
<td>All the characters in Class print except those in Class space.</td>
</tr>
<tr>
<td>jkanji</td>
<td>Class that determines Kanji (symbol or ideographic characters used for Kanji representation).</td>
</tr>
<tr>
<td>JIS X 0208</td>
<td>Character defined areas from Ku 16 to Ku 84.</td>
</tr>
<tr>
<td></td>
<td>Character defined areas from Ku 16 to Ku 77.</td>
</tr>
<tr>
<td>jkata</td>
<td>Class that determines Katakana.</td>
</tr>
<tr>
<td>JIS X 0208</td>
<td>5/1–86, 1/11, 12, 19, 20</td>
</tr>
<tr>
<td>jhira</td>
<td>Class that determines Hiragana.</td>
</tr>
<tr>
<td>JIS X 0208</td>
<td>4/1–83, 1/11, 12, 21, 22, 26</td>
</tr>
<tr>
<td>jdigit</td>
<td>Class that determines numbers except in digit.</td>
</tr>
<tr>
<td>JIS X 0208</td>
<td>3/16–25</td>
</tr>
<tr>
<td>jparen</td>
<td>Class that determines characters such as parentheses.</td>
</tr>
<tr>
<td>JIS X 0208</td>
<td>1/38–59</td>
</tr>
<tr>
<td>line</td>
<td>Class that determines ruled line primitives.</td>
</tr>
<tr>
<td>JIS X 0208</td>
<td>8/1–32</td>
</tr>
<tr>
<td>jisx0201r</td>
<td>Class that determines characters included in JIS X 0201 Katakana character graphic set.</td>
</tr>
</tbody>
</table>
JIS X 0201 Katakana  All the characters from A/1 to character graphic set  D/15.

jisx0208  Class that determines characters included in JIS X 0208.

All the characters including those in JIS X 0208 character undefined areas: From Ku 1 to Ku 84 (Ku 13 Vendor-defined character area is included).

jisx0212  Class that determine characters included in JIS X 0212.

All the characters including those in JIS X 0212 character undefined areas: From Ku 1 to Ku 84 (Ku 83 and 84 Vendor-defined character areas are also included). No characters in ja_JP.PCK locale are included in this class.

udc  Class that determines user-defined characters.

All the characters including those in character undefined areas in the user-defined character area.

ja locale

User-defined characters (Ku 1–20)  0xf5a1–0xfefe

0x8ff5a1–0x8ffefe

ja_JP.PCK locale

User-defined characters (Ku 1–20)  0xf040–0xf9fc

ja_JP.UTF-8 locale

User-defined characters (6400 characters)  0xe000–0xf8ff

vdc  Class that determines vendor-defined characters.

All the characters including those in character undefined areas in the vendor-defined character area.

ja and ja_JP.eucJP locale  JIS X 0208 Ku 13: Special symbols

JIS X 0212 Ku 83 – 84

IBM Extended characters not included in JIS X 0212.

ja_JP.PCK locale  JIS X 0208 Ku 13: Special symbols

NEC-selective IBM Extended characters  0xed40–0xfffc
**wctype_ja(3C)**

IBM Extended characters: 0xfa40–0xfcfc

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ja_JP.UTF-8 locale</strong></td>
<td>Not defined</td>
</tr>
<tr>
<td><strong>jalpha</strong></td>
<td>Class that determines alphabet letters.</td>
</tr>
<tr>
<td>JIS X</td>
<td>3/33–58, 3/65–90</td>
</tr>
<tr>
<td></td>
<td>0208</td>
</tr>
<tr>
<td><strong>jspecial</strong></td>
<td>Class that determines special symbol characters.</td>
</tr>
<tr>
<td>JIS X</td>
<td>1/2–94, 2/1–14, 2/26–33, 2/42–48, 2/60–74, 2/82–89, 94</td>
</tr>
<tr>
<td></td>
<td>0208</td>
</tr>
<tr>
<td></td>
<td>2/15–25, 2/34–36, 2/75–81</td>
</tr>
<tr>
<td></td>
<td>0212</td>
</tr>
<tr>
<td>JIS X</td>
<td>IBM Extended characters</td>
</tr>
<tr>
<td>0208 Ku13:</td>
<td>Special characters defined by NEC-selective IBM</td>
</tr>
<tr>
<td></td>
<td>Extended characters</td>
</tr>
<tr>
<td></td>
<td>Special symbols</td>
</tr>
<tr>
<td><strong>jgreek</strong></td>
<td>Class that determines Greek characters.</td>
</tr>
<tr>
<td>JIS X</td>
<td>6/1–24, 6/33–56</td>
</tr>
<tr>
<td></td>
<td>0208</td>
</tr>
<tr>
<td><strong>jrussian</strong></td>
<td>Class that determines Russian characters.</td>
</tr>
<tr>
<td>JIS X</td>
<td>7/1–7/33, 7/49–81</td>
</tr>
<tr>
<td></td>
<td>0208</td>
</tr>
<tr>
<td><strong>junit</strong></td>
<td>Class that determines unit symbols.</td>
</tr>
<tr>
<td>JIS X</td>
<td>1/75–83, 2/82, 83</td>
</tr>
<tr>
<td></td>
<td>0208</td>
</tr>
<tr>
<td></td>
<td>2/80</td>
</tr>
<tr>
<td></td>
<td>0212</td>
</tr>
<tr>
<td><strong>jsci</strong></td>
<td>Class that determines scientific symbols.</td>
</tr>
<tr>
<td>JIS X</td>
<td>1/60–74, 2/26–33, 2/42–48, 2/60–74</td>
</tr>
<tr>
<td></td>
<td>0208</td>
</tr>
<tr>
<td><strong>jgen</strong></td>
<td>Class that determines general symbols.</td>
</tr>
<tr>
<td>JIS X</td>
<td>1/84–94, 2/1–14, 2/84–89, 94</td>
</tr>
<tr>
<td></td>
<td>0208</td>
</tr>
<tr>
<td></td>
<td>2/35, 75, 2/79–81</td>
</tr>
<tr>
<td></td>
<td>0212</td>
</tr>
<tr>
<td><strong>jpunct</strong></td>
<td>Class that determines punctuation symbols.</td>
</tr>
</tbody>
</table>

JFP Reference Manual 3 : Library Routines  

21
ascii Class that determines JIS X 0201 Functional character set, Space characters, Roman character graphic set, and Kill characters.

paren Class that determines characters such as parentheses.

jisx0201 Class that determines characters included in JIS X 0212.

gaiji Class that determines implementer defined characters. udc and vdc classes are included.

jhankana Class that determines characters used for Japanese representation included in JIS X 0212.

jspace Class that determines space characters included in JIS X 0208 and JIS X 0212.

XX/YY in JIS X 0201 Functional character set, Roman character graphic set, and Katakana character graphic set denotes Column XX and Row YY. XX/YY in JIS X 0208 and JIS X 0212 denotes Ku XX and Point YY.

In case of JIS X 0212 characters, this rule only applies to ja, ja_JP.eucJP, or ja_JP.UTF-8 locale.

**EXAMPLES**

The following example shows how to determine if the wide character wc is included in Class udc.

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

**SEE ALSO**

iswctype(3C), wctype(3C), wctransJa(3C), eucJP(5), PCK(5)