JFP Reference Manual 4 : File Formats

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

Part No: 817–0647–10
December 2002

021225@5115

Contents

Preface

Both novice users and those familar 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 secton 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.

DIAGNOSTICS	This section lists diagnostic messages with a brief explanation of the condition causing the error.
WARNINGS	This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.
NOTES	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.
BUGS	This section describes known bugs and, wherever possible, suggests workarounds.

JFP Reference Manual 4 : File Formats

NAME

Intro_jfp, intro_jfp – introduction to JFP file formats

DESCRIPTION

This section outlines the formats of JFP various files. The C structure declarations for the file formats are given where applicable. Usually, the headers containing these structure declarations can be found in the directories `/usr/include` or `/usr/include/sys`. For inclusion in C language programs, however, the syntax `#include <filename.h>` or `#include <sys/filename.h>` should be used.

Because the operating system now allows the existence of multiple file system types, there are several instances of multiple manual pages with the same name. These pages all display the name of the FSType to which they pertain, in the form *name_fstype* at the top of the page. For example, `fs_ufs`.

LIST OF FILES

Name	Description
`Intro_jfp`(4)	introduction to JFP file formats
`atok12wordlist`(4)	Text word file for ATOK12 dictionary utility
`css.conf`(4)	CS starting information file
`jserverrc`(4)	Initialization file for Wnn6 Kana-Kanji conversion server
`sdtudc_map`(4)	User defined character conversion map file
`uumkey`(4)	Wnn6 Kana-Kanji conversion key binding definition file
`uumrc`(4)	xjsi and uum initialization file
`wnnenvrc`(4)	Wnn6 Kana-Kanji conversion dictionary/conversion parameter configuration file
`wnnhosts`(4)	Wnn6 Kana-Kanji conversion server/dictionary lookup server access control file
`wnn_2A_CTRL`(4)	Change input conversion mode definition table
`wnn_2B_ROMKANA`(4)	Roman character Kana conversion definition table
`wnn_automaton`(4)	Automaton
`wnn_cvt_key_tbl`(4)	Kana-Kanji conversion front end processor (uum) key code conversion table file
`wnn_cvt_xim_tbl`(4)	Key conversion table for xjsi
`wnn_hinsi.data`(4)	Wnn6 part of speech administration file
`wnn_mode`(4)	Mode definition table
`wnn_serverdefs`(4)	Wnn6 Kana-Kanji conversion server connection parameter configuration file
`wnn_ximrc`(4)	xjsi configuration file

NAME

atok12wordlist – Text word file for ATOK12 dictionary utility

DESCRIPTION

`atok12wordlist` is a word file of text format for ATOK12 dictionary maintenance. It is used by some functions of the `atok12`(1) dictionary utilitiy for input and output.

The format of the word file is defined as follows:

First line of the file

The first line must begin with:

```
!ATOK12
```

In this case, ! must be a half-size character. ATOK12 can be half-size characters and corresponding full-size characters.

Comment line

Lines begin with ! (Both half-size and full-size can be used) are ignored as a commented out line, except for the first line.

Specifying words using a part_of_speech

Words are specified by the following notation.

```
Reading,notation,part_of_speech
```

Either a comma or touten (Japanese comma) can be used as a delimiter. When notation contains a delimiter, enclose the notation in double- or single-quotation marks. Either a half-size characters or a full-size characters can be used for delimiters and quotation marks. For the part_of_speech type, see the description of "Possible part_of_speech". The part_of_speech entry must be in Japanese.

Specifying a word with a part_of_speech_number

Words can also be specified by the following notation.

```
Reading,notation,part_of_speech_number
```

This format is the same as above, except that the part_of_speech is specified by a number. For the part_of_speech_number, see the description of "Possible part_of_speech".

Possible reading

Length: Up to 16 characters. Dakuten (sonant sound mark) and han-dakuten (p-sound mark) are counted as a single character.

Character: The following half-size and full-size characters can be used. However, when stored in the dictionary, the characters for the reading are converted to the corresponding half-size characters.

- Full-size Hiragana
- The following half-size and full-size characters

```
Katakana
Alphabet
Numerals
Dakuten
Han-dakuten
-
+
*
/
_
#
$
%
&
=
@
:
;
<
>
```

The following characters cannot be used as the first character of the reading.

- The following half-size and full-size characters

```
Hiragana/Katakana "wo"
Hiragana/Katakana "n"
Chouon (prolonged sound mark)
Hiragana/Katakana Youon small "a", "i",
"u", "e", "o",  "ya", "yu", "yo"
Hiragana/Katakana Sokuon "small tsu"
Dakuten (sonant sound mark)
Han-dakuten (p-sound mark)
```

Order of reading: In the dictionary, the reading is stored after conversion to the corresponding half-size characters. The order of the reading is same as the order of the JIS-X0201 character set definition code. When list display or reading range specification is made using the dictionary maintenance tool, this order is used.

Possible notation

Length: Up to 50 characters. A half-size character is counted as 1 and a full-size character 2.

Character: All half-size and full-size characters

Possible part_of_speech

There are a total of 33 part_of_speech types. The part_of_speech entries must be in Japanese.

Part_of_speech_number	Part_of_speech
1	General nouns
4	Proper nouns (person name)
5	Proper nouns (location name)
6	Proper nouns (organization name)
8	Proper nouns (general)
9	Nouns (sa-gyou irregular)
10	Nouns (za-gyou irregular)
11	Nouns (adjective verbs)
13	Numerals
14	Aeverbs
15	Rentaishi (non-conjugative adjectives)
16	Conjunctions
17	Interjections
18	Independent words
19	Prefixes
21	Noun suffixes
23	Ka-gyou godan katsuyou (consonant-stem) verbs
24	Ga-gyou godan katsuyou (consonant-stem) verbs
25	Sa-gyou godan katsuyou (consonant-stem) verbs
26	Ta-gyou godan katsuyou (consonant-stem) verbs
27	Na-gyou godan katsuyou (consonant-stem) verbs
32	Ha-gyou godan katsuyou (consonant-stem) verbs
28	Ba-gyou godan katsuyou (consonant-stem) verbs

Part_of_speech_number	Part_of_speech
29	Ma-gyou godan katsuyou (consonant-stem) verbs
30	Ra-gyou godan katsuyou (consonant-stem) verbs
31	Wa-gyou godan katsuyou (consonant-stem) verbs
33	Ichidan katsuyou verbs
34	Ka-gyou irregular verbs
35	Sa-gyou irregular verbs
36	Za-gyou irregular verbs
37	Adjectives
39	Adjective verbs
41	Tankanji (single Kanji)

USAGE

See Japanese locale man pages for examples of word file entries.

SEE ALSO

`atok12`(1)

ATOK12 User's Guide

NAME

css.conf – CS starting information file

SYNOPSIS

```
/etc/css.conf
```

DESCRIPTION

`css.conf` is the editable text file and specifies CS starting information directories which contain some CS starting script for `cssd`(1M)

Any line beginning with a # is treated as comment line.

Each line shows a pathname of a CS starting information directory. The primary contents of this file are as follows.

```
/etc/css.d
/usr/lib/css.d
```

FILES

`/etc/css.conf`	CS starting information file by default

SEE ALSO

`cssd`(1M)

NAME

jserverrc – Initialization file for Wnn6 Kana-Kanji conversion server

SYNOPSIS

`/etc/lib/locale/ja/wnn/ja/jserverrc`

DESCRIPTION

`jserverrc` is the initialization file for Wnn6 Kana–Kanji conversion server (`jserver`), which is read when `jserver` starts. The followings can be set in the file.

`readfile` *dictionary_filename*
: *dictionary_filename* is a dictionary file name to be read when the server starts. The dictionary file resides on the server until the server process is terminated. This can save time for each client to read the dictionary file when it starts. Add a colon (`:`) just before the file name to allow `jserver` to read the file when `wnnds` is used. For example:

```
readfile  :iwanami/fisd
```

`max_client` *number*
: *number* is the maximum number of clients that can connect to the server. The default is `64` .

`max_sticky_env` *number*
: *number* is the maximum number of environments that can be set (fixed). The fixed environment maintains its configuration even when connection to the client is terminated and the environment can be freed. This provides a time-saving startup because configuration tasks are omitted when the environment is used next time. The default is `10` .

`jserver_dir` *path*
: *path* is the path name through which the server controls the dictionaries. User's frequency files and dictionaries are controled under the specified directory. The default is `/usr/local/lib/dic/`. `@LIBDIR` and `@LANG` notation can be used.

  `@LIBDIR`
  : Is the default directory path name (`/usr/lib/locale/ja/wnn`) for the `uum` environment file.

  `@LANG`
  : Should be ja for Japanese Solaris releases.

`def_param` *number_0. . .number_16*
: Specifies Kana–Kanji conversion parameters and pseudo-part of speech frequency. The default values are shown in parentheses.

  number_0
  : N for N (long) phrase analysis (5)

  number_1
  : Maximum number of short phrases in a long phrase (10)

  number_2
  : Main word frequency parameter (2)

	number_3	Short phrase length parameter (45)
	number_4	Main word length parameter (0)
	number_5	Bit parameter to indicate most recent usage (80)
	number_6	Dictionary parameter (5)
	number_7	Short phrase evaluation parameter (1)
	number_8	Long phrase length parameter (20)
	number_9	Number of short phrases parameter (0)
	number_10	Pseudo–part of speech "number" frequency (400)
	number_11	Pseudo–part of speech "Kana" frequency (–100)
	number_12	Pseudo–part of speech "alphanumerics" frequency (400)
	number_13	Pseudo–part of speech "symbol" frequency (80)
	number_14	Pseudo–part of speech "closing parentheses" frequency (200)
	number_15	Pseudo–part of speech "auxillary word" frequency (2)
	number_16	Pseudo–part of speech "opening parentheses" frequency (200)
	Specify integers for the above parameters.	
`max_param` *number_0. . .number_16*	Specifies the upper limits of the Kana–Kanji conversion parameters. The meaning and order of items are the same as `def_param` specifications. The default value will be set if the upper limit of autotuned parameters is smaller than the default value. If the upper limit is specified two times or more, the value set last will be used.	
`min_param` *number_0. . .number_16*	Specifies the lower limits of the Kana–Kanji conversion parameters. The meaning and order of items are the same as `def_param` specifications. The default value will be set if the lower limit of autotuned parameters is	

larger than the default value. If the lower limit is specified two times or more, the value set last will be used.

`set_giji_eisuu` *character. . .*

Specifies character codes, which can be used, in addition to alphanumerics in pseudo–phrase conversion, as the "alphanumerics" pseudo–part of speech. *character* should be specified in any of the following notations:

Character	Notation
CTRL_A	^A
' '(SPACE)	0x20 \x20 32 040 \o40 040 \o40 : octal number 32 : decimal number 0x20 \x20: hexadecimal number
' _'	' _'

`default_wnnds_list` *wnnds_info1 wnnds_info2 wnnds_info3*

Specifies the default `wnnds` to be used if no default `wnnds` options (`-ds` and `+ds`) are specified when `jserver` starts.

`wnnds` is specified in the following format.

hostname	`wnnds` that uses the well-known port number (26208) on host *hostname*.
hostname:no	`wnnds` that uses the port number obtained by adding *no* to the well-known port number on host *hostname*.
hostname/port_no	`wnnds` that uses *port_no* as the port number on host *hostname*.

Up to three `wnnds` can be specified. Attempts are made for connection on the order specified. The one successfully connected first is used as default `wnnds` .

Example: `default_wnnds_list wnnds1/26209 wnnds2 wnnds3:1`

SEE ALSO | `wnnenvutil`(1), `jserver`(1M), `wnnds`(1M)

NAME

sdtudc_map – User defined character conversion map file

SYNOPSIS

`/usr/dt/config/$LANG/sdtudc_map`

DESCRIPTION

`sdtudc_map` is the default character conversion map file accessed by `sdtudc_convert`(1) and `sdtudc_extract`(1) when moving from the environment implemented for user–defined characters under Solaris 2.5.1 and earlier to the slightly different environment used beginning under Solaris 2.6 and later. This file is a standard text file that can be edited with a regular text editor.

Any line beginning with a # is interpreted as a comment line and not read.

Each line lists a character code to be converted and the character code it should be converted to. Each character code to be converted must be placed ',' at the beginning and the end, and a character code to be converted and the character code it should be converted to must be separated by '\t'.

Initial default values for `ja` locale are as follows. This example below is how to convert code point of codeset 1 9 ku – 15 ku to codeset 1 85 ku – 91 ku.

a9a1,a9ff f5a1
aaa1,aaff f6a1
aba1,abff f7a1
aca1,acff f8a1
ada1,adff f9a1
aea1,aeff faa1
afa1,afff fba1

For example, `sdtudc_extract` would extract user–defined characters registered in the region `0xa9a1` to `0xa9ff` from the specified font file and convert them to values beginning from `0xf5a1` in the first line. Also `sdtudc_convert` would convert `0xa9a1` to `0xa9ff` code points in text file to code points beginning from `0xf5a1`.

FILES

`/usr/dt/config/$LANG/sdtudc_map`

SEE ALSO

`sdtudc_convert`(1), `sdtudc_extract`(1), `sdtudctool`(1)

NOTES

Be sure to specify `f5a1` or larger for the code point of conversion target.

NAME

uumkey – Wnn6 Kana-Kanji conversion key binding definition file

SYNOPSIS

`/usr/lib/locale/ja/wnn/ja/uumkey`

DESCRIPTION

The `uumkey` file defines the key binding for Japanese input. Each user can define own `uumkey` file.

Entries Style

Entries is set in the following style:

`include`	`uumkey` file name
`unset`	^Function entries
Function entries	^keycode [keycode...]

***include* and *unset* are called "other entries".**

entries and *setting_value* are separated by space character or tab.

Lines beginning with semicolons (`;`) or colons (`:`) are comments.

keycode is specified in octal, decimal, or hexadecimal notation as in the C programming language. For example, notation "^A" is available for control key input.

Up to 10 codes can be written for each function entry.

If the same entry appears more than once, the last one will be used. To specify more than one *keycode* to the single function, they must be set at the same time.

However, only one function entry can be set to a single key in the same Kana-Kanji conversion mode.

Integers from 0 through 511 can be taken as key code. Numbers that cannot be generated from the keyboard (e.g., integers over 127) may be converted in key code (`cvt_xim_tbl/cvt_key_tbl`) or must be generated through the Automaton.

Other Entries

`include` *uumkey_filename* — Reads the specified `uumkey` file for key binding.

`unset` *entry* — Cancels key binding for the specified entry.

Kana-Kanji Conversion Operation Mode

Each function entry can be used in specifed operate mode.

Operation Mode	Contents
0	Correction condition for conversion results; the contition of inspect.
1	Text input condition before conversion.
2	Phrase length adjustment condition after conversion.

Operation Mode	Contents
3	No strings specified condition (input buffer is empty).
4	Candidate selection condition. Including part of speech or dictionary registration condition in `uum`

Function Entries

The function entries followed by `_e` can operate with the operation mode 3 (input buffer is empty), adding the operation mode of the same function as those of the function entries followed by no `_e`.

- Standard function entries

Entry Name	Operation Mode	Function
`henkan_on`	`01234`	Turns the conversion mode ON and OFF. This function can be used in all modes.
`send_string`	`012`	Binds the text string in the conversion line with input keycode that is allocated to `send_string`, and sends it to the application (Final operation).
`kakutei`	`012`	Sends the text string in the conversion line to the application (Final operation).
`forward_char`	`1`	Moves the cursor one character to the right.
`backward_char`	`1`	Moves the cursor one character to the left.
`goto_top_of_line`	`1`	Moves the cursor to the first character on the line.
`goto_end_of_line`	`1`	Moves the cursor to the last character on the line.
`delete_char_at_cursor`	`1`	Deletes the character at the cursor.
`fiwnn`	`012`	Opens "GAKUSYUU/HENKAN/HYOUJI" window.
`fiwnn_e`	`0123`	Opens "GAKUSYUU/HENKAN/HYOUJI" window. It operates even when the buffer is empty.

Entry Name	Operation Mode	Function
hindo	012	Saves frequency information.
hindo_e	0123	Saves frequency information. It operates even when the buffer is empty.
kensaku	012	Opens "TANGO_SAKUJO,HENSYUU" window.
kensaku_e	0123	Opens "TANGO_SAKUJO,HENSYUU" window. It operates even when the buffer is empty.
keybind	012	Opens "NYUURYOKU_SUTAIRU" window.
keybind_e	0123	Opens "NYUURYOKU_SUTAIRU" window. It operates even when the buffer is empty.

Entry Name	Operation Mode	Function
kaijo	02	Restores the converted string after the phrase the cursor is positioned to the pre-conversion state.
henkan	1	Converts continuous phrases.
tan_henkan	1	Converts a single phrase of a short phrase as one phrase.
tan_henkan_dai	1	Converts a single phrase of a long phrase as one phrase.
nobi_henkan	2	When adjusting the phrase length, performs single phrase conversion for the highlighted section as a short phrase and performs continuous phrase conversion thereafter.

Entry Name	Operation Mode	Function
nobi_henkan_dai	2	When adjusting the phrase length, performs single phrase conversion for the highlighted section as a long phrase and performs continuous phrase conversion thereafter.
jikouho	0	Displays the next candidate as a short phrase.
jikouho_dai	0	Displays the next candidate as a long phrase.
zenkouho	0	Displays the previous candidate as a short phrase.
zenkouho_dai	0	Displays the previous candidate as a long phrase.
select_jikouho	0	Displays the candidate list as short phrases.
select_jikouho_dai	1	Displays the candidate list as long phrases.
kana_henkan	1	Performs Kana–Kanji conversion. However, only reverse lookup dictionary (Kanji to Kana coversion is possible in registerable format) is effective.
kill	1	Deletes the text string at and after the cursor and places it in the kill buffer.
yank	1	Inserts the contents of kill buffer to the cursor position.
yank_e	13	Inserts the contents of kill buffer to the cursor position. It operates even when the buffer is empty.

Entry Name	Operation Mode	Function
bunsetu_nobasi	0	Increases the phrase length by one character.
bunsetu_chijime	0	Reduces the phrase length by one character.

Entry Name	Operation Mode	Function
jisho_utility	012	Opens Wnn6 menu.
jisho_utility_e	0123	Opens Wnn6 menu. It operates even when the buffer is empty.
touroku	012	Opens "TANGO_TOUROKU" window.
touroku_e	0123	Opens "TANGO_TOUROKU" window. It operates even when the buffer is empty.
sainyuuryoku	1	Replaces the contents of the input buffer with the previously input Kana string.
sainyuuryoku_e	13	Replaces the contents of the input buffer with the previously input Kana string. It operates even when the buffer is empty.
kuten	1	Enters the Kuten code input mode.
kuten_e	13	Enters the Kuten code input mode. It operates even when the buffer is empty.
jis	1	Enters the JIS code input mode.
jis_e	13	Enters the JIS code input mode. It operates even when the buffer is empty.
redraw_line	0124	Redraw the conversion line.
redraw_line_e	01234	Redraw the conversion line. It operates even when the buffer is empty.

Entry Name	Operation Mode	Function
previous_history	1	Replaces the contents of the input buffer with the previous text string recorded in the history.
previous_history_e	13	Replaces the contents of the input buffer with the previous text string recorded in the history. It operates even when the buffer is empty.

Entry Name	Operation Mode	Function
next_history	1	Replaces the contents of the input buffer with the next text string recorded in the history.
next_history_e	13	Replaces the contents of the input buffer with the next text string recorded in the history. It operates even when the buffer is empty.
all_history	1	Replaces the contents of the input buffer with all the text string recorded in the history.
all_history_e	13	Replaces the contents of the input buffer with all the text string recorded in the history. It operates even when the buffer is empty.
bushu	1	Radical input. It is not used for uum.
bushu_e	13	Radical input. It is not used for uum. It operates even when the buffer is empty.
com_entry		Adds comments to word and dictionary for uum.

Entry Name	Operation Mode	Function
greek	13	Inputs the Greek characters list. It is not used for uum.
hankaku	012	Converts to the half-width characters. It is not used for uum.
hiragana	012	Converts to the Enters the Hiragana characters. It is not used for uum.
jis2	13	Enters the auxiliary Kanji hexadecimal code input mode. It is not used for uum.
katakana	012	Converts to the Katakana. It is not used for uum.

Entry Name	Operation Mode	Function
russian	13	Enters the Cyrillic characters (Russian) list input. It is not used for uum.
select_ikeiji_dai	0	Converts the long phrase special characters.
tankan_henkan	01	Performs single Kanji conversion. It is not used for uum.
kuten2	13	Inputs auxiliary Kanji Kuten code. It is not used for uum.
next_page	4	Displays the next candidates list. It is not used for uum.
tankan_nobi_henkan	2	Performs single Kanji conversion. It is not used for uum.
tel_henkan	01	Converts telephone number. It is not used for uum.
tel_nobi_henkan	2	Converts telephone number. It is not used for uum.
zip_henkan	01	Converts zip code. It is not used for uum.
zip_nobi_henkan	2	Converts zip code. It is not used for uum.
one_char_kakutei	012	Commits one character. It is not used for uum.
one_char_no_henkan	01	Does not convert N characters. It is not used for uum.
previous_page	4	Displays the previous candidate list. It is not used for uum.
kigou	13	Enters the symbol input mode. It is not used for uum.
eisuu	012	Converts to alphanumerics. It is not used for uum.

Entry Name	Operation Mode	Function
change_to_insert_mode	0	Enables editing of a converted text string. Converted Kanji will not be returned to Kana.

Entry Name	Operation Mode	Function
quote	1	Includes the next input character to input buffer directly if it is not allocated to henkan_on character. In this case, inputed characters are not converted for Roman-Kana conversion and allocated function is invalid.
quote_e	13	Equivalent to quote. It operates even when the buffer is empty.
forward_select	4	Moves to the next (right) candidate. The next candidates is displayed (if necessary).
backward_select	4	Moves to the previous (left) candidate. The previous candidates is displayed (if necessary).
next_select	4	Moves to the next (under) line in candidates. The next candidates is displayed (if necessary).
previous_select	4	Moves to the previous (above) line in candidates. The previous candidates is displayed (if necessary).
linestart_select	4	Moves to the left side candidate.
lineend_select	4	Moves to the right side candidate.
select_select	4	Select candidates and shifts to operation mode 0.
send_ascii_char	01234	Prevents to store in the input buffer when the input buffer is empty and ASCII characters are input (Sends for ASCII characters).
not_send_ascii_char	01234	Store in the input buffer when the input buffer is empty and ASCII characters are input (Prevents to send for ASCII characters).

Entry Name	Operation Mode	Function
pop_send_ascii_char	01234	Restores the previous sending for ASCII operation status when the buffer is empty and ASCII characters are input.
togle_send_ascii_char	01234	Reverses the ASCII operation status when the buffer is empty and ASCII characters are input for uum.
quote_send_ascii_char	3	Stores the next inputed ASCII character into the buffer when the input buffer is empty in the ASCII operation status.
reconnect_jserver	01234	Reconnects to the Kana-Kanji conversion server (jserver).
inspect	0	Displays the candidates information.
sakujo_kouho	0	Deletes a candidate from the Kana-Kanji conversion dictionary.
forward_bunsetsu	0	Moves forward one phrase. If not defined key binding explicitly, the key for forward_char is allocated. It is not used for uum.
henkan_forward	2	Performs single phrase conversion for the phrase adjusting the length, performs continuous phrase conversion thereafter, and moves to the forward one phrase. If not defined key binding explicitly, the same key code as forward_char is used.
backward_bunsetsu	0	Moves bakward one phrase. If not defined key binding explicitly, the key for backward_char is allocated. It is not used for uum.

Entry Name	Operation Mode	Function
henkan_backward	2	Performs single phrase conversion for the phrase adjusting the length, performs continuous phrase conversion thereafter, and moves to the backward one phrase. If not defined key binding explicitly, the same key code as backward_char is used.
top_bunsetsu	0	Moves top one phrase. If not defined key binding explicitly, the key for goto_top_of_line is allocated. It is not used for uum.
end_bunsetsu	0	Moves end one phrase. If not defined key binding explicitly, the key for goto_end_of_line is allocated. It is not used for uum.

Entry Name	Operation Mode	Function
jmptijime	2	Moves to the first character. If not defined key binding explicitly, the key for goto_top_of_line is allocated. It is not used for uum.
c_end_nobi	2	Moves to the last character. If not defined key binding explicitly, the key for goto_end_of_line is allocated. It is not used for uum.
forward	2	Increases the length of the phrase by one character. If not defined key binding explicitly, the key for bunsetu_nobasi is allocated. It is not used for uum.
chijime	2	Decreases the length of the phrase by one character. If not defined key binding explicitly, the key for bunsetu_chijime is allocated. It is not used for uum.

Entry Name	Operation Mode	Function
rubout	1	Deletes the character to the left of the cursor. Also used in Roman-Kana conversion. The key code must be the range from 0 to 255. It is not used for uum.
next_ku_kuten	4	Displays candidates in next ku on the kuten number input candidate list window. It is not used for uum.
previous_ku_kuten	4	Displays candidates in previous ku on the kuten number input candidate list window. It is not used for uum.

- Function entries in ATOK8 input style

Entry name	Operation Mode	Function
bubun_kakutei	02	Minimum decision. Decides first one phrase. Phrases are available to operate continuously. It is not used for uum.
ichi_oto_kakutei	1	ICHIOTO decision. Decides first one KANA. Phrases are available to operate continuously. It is not used for uum.
bunsetu_top_kakutei	02	One character decision (top) . Decides first one character of the serected phrase (target phrase). Any strings except this character are deleted. It is not used for uum.
bunsetu_end_kakutei	02	One character decision (last) . Decides last one character of the serected phrase (target phrase). Any strings except this character are deleted. It is not used for uum.
bunsetu_kakutei	012	Target phrase decision. Decides first phrase to the serected phrase (target phrase). Phrases are available to operate continuously. It is not used for uum.

Entry name	Operation Mode	Function
atok_jikouho	4	Next candidate. Displays next candidates in the converting line and selects it in the candidate list selecting window. It is not used for uum.
hankaku_space_input	0123	Inputs hankaku space. It is not used for uum.
sjis	1	Enters the PC Kanji hexadecimal code input mode. It is not used for uum.
sjis_e	13	Enters the PC Kanji hexadecimal code input mode. It operates even when the input buffer is empty. It is not used for uum.
atok_zenkouho	4	Previous candidate. Displays previous candidates in the converting line and selects it in the candidate list selecting window. It is not used for uum.
atok_jikouho_gun	4	Displays next candedate. It is not used for uum.
atok_jikouho_gun	4	Displays previous candedate. It is not used for uum.
top_kigou_kuten	4	Displays first symbol ku on the kuten number input candidate list window. It is not used for uum.
top_gaiji_kuten	4	Displays first ku in the storage to available for user definition character on the kuten number input candidate list window. It is not used for uum.
atok_kill	0124	Delete. Deletes any undefined strings for conversion. It is not used for uum.
atok_kaijo	012	Reset. Resets strings converted after target phrase to input mode. It is not used for uum.
zen__kaijo	0	Resets all conversion (entire). Resets all undefined characters to input mode. It is not used for uum.

Entry name	Operation Mode	Function
atok_rubout	1	Deletes the character to the previous of the cursor. If converted (undefined) phrase is before cursor, resets the phrase to input mode. It is not used for uum.
new_backward_char	1	Moves to the previous one character. If converted (undefined) phrase is before cursor, resets the phrase to input mode. It is not used for uum.
zenkaku_space_input	0123	Mode 0 : Displays next candidate. Mode 13 : Inputs zenkaku space. Mode 2 : Converts in extend phrase. It is not used for uum.
atok_select_jikouho	0	Displays next candidate list of ATOK style. It is not used for uum.

- Function entries in ATOK7 input style

Entry name	Operation Mode	Function
atok_bubun_kakutei	01	Minimum decision. It is not used for uum.

- Function entries in cs00 input style

Entry name	Operation Mode	Function
send_string_off	0123	Defined the undefined characters, and sets convert OFF. It is not used for uum.

- Function entries in EGBRIDGE input style

Entry name	Operation Mode	Function
eg_zenkaku_eisuu	012	Converts to zenkaku numeric characters. It is not used for uum.
eg_hankaku_katakana	012	Converts to hankaku katakana. It is not used for uum.
eg_hankaku_eisuu	012	Converts to hankaku numeric character. It is not used for uum.

Entry name	Operation Mode	Function
eg_Aa_henkan_big_loop	012	Converts to the following order, Hiragana, katakana, hankaku katakana, zenkaku numeric, and hankaku numeric. It is not used for uum.
eg_Aa_henkan_small_loop	012	Converts to the following order, Hiragana, katakana, and hankaku katakana. It is not used for uum.
code_convert	012	Code reconversion. Converts to hexadecimal code. It is not used for uum.
undetermined_henkan	1	Conversion in undeternined. Modifies inputed YOMI without conversion. It is not used for uum.
delete_one_kanji	0	Deletes last one character of the target phrase, and then does not delete the character in the phrase when the target phrase consists of only one character. It is not used for uum.

EXAMPLE

```
; include file
include /usr/lib/locale/ja/wnn/ja/uumkey
; Commands                          Codes
unset sjis_e
atok_select_jikouho               0x20 0x9E 0x118 ^W
```

- The first and third lines are comments.
- Includes the standard key binding definition file in the second line.
- Resets the key binding for sjis_e function entry in the forth line.
- Binds the key for atok_select_jikouho function entry in the fifth line.

SEE ALSO

uum(1), wnnenvutil(1), xjsi(1), uumrc(4), wnn_automaton(4), wnn_cvt_key_tbl(4),wnn_cvt_xim_tbl(4),wnn_mode(4)

BUGS

It cannot be allocated key code generated by Automaton set as henkan_on entries.

NAME

uumrc – xjsi and uum initialization file

SYNOPSIS

`/usr/lib/locale/ja/wnn/ja/uumrc`

DESCRIPTION

`uumrc` is used to set the Wnn6 Kana–Kanji conversion standard interface for each user.

Entries is set in the following style:

entry setting_value ...entry and *setting_value* are separated by space character or tab. Lines beginning with semicolons (`;`) are comments. If the same entry that may not specified only once, appears more than once, the last one will be used.

`include` *uumrc_filename*
Reads another `uumrc` file. Used to add personal settings on the another `uumrc` file. If `@DEFAULT` is specified, `uumrc` is read in the following order.

1. `@HOME/.Wnn6/uumrc`
2. `/etc/lib/locale/ja/wnn/ja/uumrc`
3. `/usr/lib/locale/ja/wnn/ja/uumrc` This entry can be specified more than once.

`setconvenv` *wnnenvrc_filename*

`setconvenv` *wnnenvrc_filename* `sticky`

`setconvenv` *jserver wnnenvrc_filename*

`setconvenv` *jserver wnnenvrc_filename* `sticky`
Specifies the Kana-Kanji conversion configuration file.

Specifies the connected Kana–Kanji conversion server by `jserver`. It is specified in the following order.

hostname	Kana–Kanji conversion server that uses the standart port number (22273) on *hostname*
hostname:offset	Kana–Kanji conversion server that uses the standart port number plus *offset* as port number on *hostname*
hostname/port_number	Kana–Kanji conversion server that uses *port_number* as port number on *hostname*

If `sticky` is specified, the current environment settings are stored even after `xjsi/uum` exits. This eliminates the need to initialize the environment the next time `xjsi/uum` starts and increases startup speed.

If this setting is omitted, path name for `wnnenvrc` is determined in the following order.

1. `@HOME/.Wnn6/wnnenvrc`
2. `/etc/lib/locale/ja/wnn/ja/wnnenvrc`
3. `/usr/lib/locale/ja/wnn/ja/wnnenvrc`

`setkankanaenv` *wnnenvrc_filename*

`setkankanaenv` *wnnenvrc_filename* `sticky`

`setkankanaenv` *server_hostname wnnenvrc_filename*

`setkankanaenv` *server_hostname wnnenvrc_filename* `sticky`
Specifies the Kanji–Kana conversion configuration file. By standard setting, Kanji cannot be converted to Kana. Kanji–Kana conversion cannot be done using the Wnn6 system dictionary.

The connected Kana–Kanji conversion server can be specified by `jserver`. The style of Kana–Kanji conversion server is the same of `setconvenv`.

If `sticky` is specified, the current environment settings are stored even after `xjsi/uum` exits. This eliminates the need to initialize the environment the next time `xjsi/uum` starts and increases startup speed.

`setmaxchg` *number*
Specify the maximum number of convertable characters. The default value will be used if 0 or a number below 0 is specified. The default is 100.

`setmaxbunsetsu` number
Specify the maximum number of convertable phrases. The upper limit is 400. The default value will be used if 0 or a number below 0 is specified. The default is 80.

`setmaxichirankosu` *number*
Specify the maximum number of candidates that can be displayed on the candidate list for `uum`. The default value will be used if 0 or a number below 0 is specified. The default is 36.

`setmaxhistory` *number*
Specify the maximum number of entries to be saved in the history. The default value will be used if 0 or a number below 0 is specified. The default is 11.

`excellent_delete`
Deletes the individual alphabetic characters that were input when deleting confirmed characters for the Automaton (roman character–Kana conversion) (default setting).

`simple_delete`
Deletes individual Japanese language characters when deleting confirmed characters for the Automaton (roman character–Kana conversion).

`flow_control_on`
Turns ON tty flow control (default setting) for `uum`.

`flow_control_off`
Turns OFF tty flow control for `uum`.

`convkey_not_always_on`
Disables key code conversion when conversion is turned OFF (default setting) for `uum`.

`convkey_always_on`
Enables key code conversion when conversion is turned OFF for `uum`.

`not_send_ascii_char`
Places ASCII characters in the Kana–Kanji conversion buffer when the buffer is empty (default setting).

`send_ascii_char`
Does not place ASCII characters in the Kana–Kanji conversion buffer when the buffer is empty.

`waking_up_in_henkan_mode`
Boots with the conversion mode turned ON (default setting).

`waking_up_no_henkan_mode`
Boots with the conversion mode turned OFF.

`henkan_off_kuten`
Does not convert Japanese language period (default setting).

`henkan_on_kuten`
Converts Japanese language period.

`setuumkey` *uumkey_filename*
Specifies the key binding configuration file.

If this setting is omitted, the path name for `uumkey` file is determined in the following order.

1. `/etc/lib/locale/ja/wnn/ja/uumkey`
2. `/usr/lib/locale/ja/wnn/ja/uumkey`

`setrkfile` *roman_character-Kana_conversion_file_name*
Specifies the mode defined table file name for the roman–Kana conversion. The direcroty including the mode defined table file (`mode`) can be also specified. The default is `/usr/lib/locale/ja/wnn/ja/rk/mode`.

`setconvkey` *convert–key_filename*

`setconvkey` *termtype convert–key_filename*
Specifies the key code conversion table file that is used to resolve the differences in key binding between machines (terminals) in `uum`. The default is `/usr/lib/locale/ja/wnn/cvt_key_tbl`. The entry that is specified *termtype* and it matches the value of the TERM environment variable, the entry is in effect. A wildcard character (*) can be used. This entry can be specified more than once.

`setjishopath` *pathname*
Specify the initial value of the dictionary file name input buffer that is used in adding dictionaries function of `uum`. The default is an empty string.

`sethindopath` *pathname*
Specify the initial value of the frequency file name input buffer that is used in adding dictionaries function of `uum`. The default is an empty string.

`setfuzokugopath` *pathname*
Specify the initial value of the auxiliary word file name input buffer that is used in adding dictionaries function of `uum`. The default is an empty string.

`touroku_comment`
Accepts comment input for word registration for `uum`.

`touroku_no_comment`
Does not accepts comment input for word registration for `uum` (default setting).

The following can be used at file name.

~	Value of the HOME environmental variable.
~*user*	*user* home directory.
`@HOME`	Value of the HOME environmental variable.
`@LIBDIR`	`/usr/lib/locale/ja/wnn`
`@USR`	user name
`@LANG`	`ja`

The first @USR in the arguments for `setdic`, `setjishopath`, or `sethindopath` is expanded into the startup user name.

SEE ALSO `uum`(1), `wnnenvutil`(1), `xjsi`(1), `jserver`(1M), `uumkey`(4), `wnn_automaton`(4), `wnn_cvt_key_tbl`(4),`wnn_mode`(4), `wnnenvrc`(4)

NAME

wnnenvrc – Wnn6 Kana-Kanji conversion dictionary/conversion parameter configuration file

SYNOPSIS

`/usr/lib/locale/ja/wnn/ja/wnnenvrc`

DESCRIPTION

`wnnenvrc` is a file that sets the Kana–Kanji conversion dictionary, attributes files, and conversion parameters.

Entries is set in the following style:

entry setting_value ...entry and *setting_value* are separated by space character or tab. Lines beginning with semicolons (`;`) are comments. If the same entry except for `setdic` appears more than once, the last one will be used. Up to 30 files can be specified as entry setting dictionaries (`setdic`, `set_fi_system_dic`, `set_fi_user_dic`, `muhenkan_gakusyuu`, `bunsetsugiri_gakusyuu`).

- `include` *wnnenvrc_filename*

  Reads another `wnnenvrc` file, and is used to add individual user settings to another `wnnenvrc` file. The following can be used for file name.

`~`	Value of `HOME` environmental variable
~user	The home directory of *user*
`@HOME`	Value of `HOME` environmental variable
`@LIBDIR`	`/usr/lib/locale/ja/wnn`
`@USR`	user name

- `setdic` *dictionary_file_name frequency_file_name dictionary_priority dictionary_read–only_setting frequency_read–only_setting dictionary_file_password_file_name frequency_file_password_file_name conversion*

  The above dictionaries are set.

  dictionary_file_name frequency_file_name
  Specifies the name of the dictionary file and the frequency file. The frequency in the dictionary file will be used if "–" is specified. The following specification for file position can be used at the beginning of file names.

`!`	For a client file
`:`	For a `jserver` file
hostname`:`	For a *hostname* `wnnds` file

  If a file position is not specified and the dictionary server (`wnnds`) set by `jserver` is specified, the `wnnds` file will be used. Otherwise, the `jserver` file is used.

  dictionary_priority
  Specifies the dictionary priority as a decimal value.

  dictionary_read–only_setting
  Specifies the dictionary read–only setting.

1 Read–only. Does not update the dictionary file.

2 Temporary learning. Learns temporarily, but does not update the dictionary file.

3 Group dictionary. Shares one dictionary among multiple users. Updates the dictionary file.

4 Merge dictionary. Shares one dictionary among multiple users. Does not updates the dictionary file.

0 Others.

frequency_read–only_setting
Specify the dictionary read–only setting.

`1` Read–only. Does not update the frequency file.

`2` Temporary learning. Learns temporarily, but does not update the frequency file.

`0` Others.

dictionary_file_password_file_name
Specifies the file in which the dictionary file password is described

frequency_file_password_file_name
Specifies the file in which the frequency file password is described

conversion
Specify `0` for Kana–Kanji conversion and `1` for Kanji–Kana conversion.

If the specification is omitted setting after *frequency_file_name*, the following value is used.

```
- 5 0 0 - - 0
```

- `set_fi_system_dic` *FI–related_system_dictionary_file_name FI–related_frequency_file_name FI–related_frequency_read–only_setting FI–related_frequency file_password_file_name*

  FI–related_system_dictionary_file_name FI–related_frequency_file_name
  Specifies the FI–related user dictionary file name. If *FI–related_system_dictionary_file_name* is specified "-", FI related frequency is not updated. The following specification for file position can be used at the beginning of *FI–related_system_dictionary_file_name* and *FI–related_frequency_file_name*.

  `!` For a client file

  `:` For a `jserver` file

  hostname`:` For a *hostname* `wnnds` file

If a file position is not specified and the dictionary server (`wnnds`) set by `jserver` is specified, the `wnnds` file will be used. Otherwise, the `jserver` file is used.

FI–related_frequency_read–only_setting/fP
Specify the FI-related frequency read–only setting.

`1`	Read–only. Does not update the FI-related frequency file.
`2`	Temporary learning. Learns the FI-related frequency file temporarily, but does not update the FI-related frequency file.
`3`	Group dictionary. Shares one FI-related frequency dictionary among multiple users. Updates the FI-related frequency file.
`0`	Others.

FI–related_frequency file_password_file_name
Specifies the file in which the FI-related frequency file password is described

`set_fi_user_dic` *FI–related_user_dictionary_file_name*
FI–related_user_dictionary_read–only_setting
FI–related_user_dictionary_file_password_file_name

FI–related_user_dictionary_file_name
Specifies the FI-related user dictionary file. The following specification for file position can be used at the beginning of the file.

`!`	For a client file
`:`	For a `jserver` file
hostname`:`	For a *hostname* `wnnds` file

If a file position is not specified and the dictionary server (`wnnds`) set by `jserver` is specified, the `wnnds` file will be used. Otherwise, the `jserver` file is used.

FI–related_user_dictionary_read–only_setting
Specify the FI-related user dictionary read–only setting.

`1`	Read–only. Does not update the FI-related user dictionary file.
`2`	Temporary learning. Learns temporarily, but does not update the FI-related user dictionary file.
`3`	Group dictionary. Shares one dictionary among multiple users. Updates the FI-related user dictionary file.
`0`	Others.

FI–related_user_dictionary_file_password_file_name
Specifies the file in which the FI-related user dictionary file name password is described.

`muhenkan_gakusyuu` *no-conversion_learning_dictionary_file_name no–conversion_learning_dictionary_read–only_setting no–conversion_learning_dictionary_priority no–conversion_learning_dictionary_file_password_file_name conversion*

Sets the no-conversion learning. No–conversion learning is used to automatically record candidates in the no–conversion learning dictionary when candidates are confirmed in Hiragana, Katakana, or roman characters.

no-conversion_learning_dictionary_file_name

Specifies no-conversion learning dictionary file name. The following specification for file position can be used at the beginning of the file.

`!`	For a client file
`:`	For a `jserver` file
hostname`:`	For a *hostname* `wnnds` file

If a file position is not specified and the dictionary server (`wnnds`) set by `jserver` is specified, the `wnnds` file will be used. Otherwise, the `jserver` file is used.

no–conversion_learning_dictionary_read–only_setting

Specify the no-conversion learning dictionary read–only setting.

`1`	Read–only. Does not update the no-conversion learning dictionary file.
`2`	Temporary learning. Learns temporarily, but does not update the no-conversion learning dictionary file.
`3`	Group dictionary. Shares one no-conversion learning dictionary among multiple users. Updates the no-conversion learning dictionary file.
`0`	Others.

no–conversion_learning_dictionary_priority

Specifies the no-conversion learning dictionary priority in a decimal number.

no–conversion_learning_dictionary_file_password_file_name

Specifies the file in which the no-conversion learning dictionary file password is decribed.

conversion

Specify `0` for Kana–Kanji conversion and `1` for Kanji–Kana conversion.

- `bunsetsugiri_gakusyuu` *rephrasing_learning_dictionary_file_name rephrasing_learning_dictionary_read–only_setting rephrasing_learning_dictionary_priority rephrasing_learning_dictionary_ file_password_file_name conversion*

  Set the rephrasing learning. Rephrasing learning is used to record one phrase in the rephrasing learning dictionary, which is made from two phrases before and after a rephrasing point when confirming phrases.

rephrasing_learning_dictionary_file_name
Specifies the rephrasing learning dictionary file. The following specification for file position can be used at the beginning of the file.

`!`
For a client file

`:`
For a `jserver` file

hostname`:`
For a *hostname* `wnnds` file

If a file position is not specified and the dictionary server (`wnnds`) set by `jserver` is specified, the `wnnds` file will be used. Otherwise, the `jserver` file is used.

rephrasing_learning_dictionary_read–only_setting
Specify the rephrasing learning dictionary setting.

`1`	Read–only. Does not update the rephrasing learning dictionary file.
`2`	Temporary learning. Learns temporarily, but does not update the rephrasing learning dictionary file.
`3`	Group dictionary. Shares one rephrasing learning dictionary among multiple users. Updates the rephrasing learning dictionary file.
`0`	Others.

rephrasing_learning_dictionary_priority
Specifies the rephrasing learning dictionary priority in a decimal number.

rephrasing_learning_dictionary_ file_password_file_name
Specifies the file in which the rephrasing learning dictionary file password is decribed.

conversion
Specifies `0` for Kana–Kanji conversion and `1` for Kanji–Kana conversion.

- `setfuzokugo` *auxiliary_word_file*

  `setgrammar` *auxiliary_word_file*

  Specifies the auxiliary word file name.

- `setparam param0. . . param9 hindo0. . . hindo6`

  Specifies conversion parameters and pseudo–part of speech frequencies as integers. Defaults are given in parentheses.

param0	N for N (long) phrase analysis (5)
param1	Maximum number of short phrases in a long phrase (10)
param2	Main word frequency parameter (2)
param3	Short phrase length parameter (45)
param4	Main word length parameter (0)

param5	Bit parameter to indicate immediate previous usage (80)
param6	Dictionary parameter (5)
param7	Short phrase evaluation parameter (1)
param8	Long phrase length parameter (20)
param9	Number of short phrases parameter (0)
hindo0	Pseudo–part of speech "number" frequency (400)
hindo1	Pseudo–part of speech "Kana" frequency (–100)
hindo2	Pseudo–part of speech "alphanumerics" frequency (400)
hindo3	Pseudo–part of speech "symbol" frequency (80)
hindo4	Pseudo–part of speech "closing parentheses" frequency (200)
hindo5	Pseudo–part of speech "auxillary word" frequency (2)
hindo6	Pseudo–part of speech "opening parentheses" frequency (200)

- `confirm`

  Asks the user to confirm whether or not to create a dictionary or frequency file if a dictionary or frequency file does not exist after this entry.

- `confirm1`

  Asks the user once to confirm whether or not to create a dictionary or frequency file if a dictionary or frequency file does not exist after this entry and then continues using the confirmation thereafter.

- `create_without_confirm`

  Creates dictionary or frequency file unconditionally if a dictionary or frequency file does not exist after this entry (initial setting).

- `no_create`

  Does not create dictionary or frequency file if a dictionary or frequency file does not exist after this entry.

- `saisyu_siyou TRUE|FALSE`

  Last–useage top–priority processing provides the last–used homophone as the first candidate and then lists the remaining candidates in the order of previous confirmation (up to six candidates). Specify `TRUE` to enable this processing and `FALSE` to disable it.

- `fukugou_yuusen TRUE|FALSE`

  Composite–word priority conversion gives priority to candidates that do not contain auxilliary words. Specify `TRUE` to enable this processing and `FALSE` to disable it.

- `okuri_kijun REGULATION|YES|NO`

Okurigana processing is used to provide the first candidate according to the specified rules when converting words that can have different Okurigana. Specify `REGULATION` to take precedence candidates according to the basic rule, specify `YES` to take precedence the long Okurigana, `NO` to take precedence the short Okurigana.

- `settou_kouho HIRAGANA|KANJI`

  Specify `HIRAGANA` to provide Hiragana candidates first, `KANJI` to provide Kanji `candidates` first.

- `rendaku TRUE|FALSE`

  Euphonic change processing does not provide candidates with euphonic changes to voiced sounds first. Specify `TRUE` to enable this processing and `FALSE` to disable it.

- `yuragi TRUE|FALSE`

  Long vowel/alternate spelling processing enables the conversion of long vowels and alternate spellings. Specify `TRUE` to enable this processing and `FALSE` to disable it.

- `okuri_gakusyu TRUE|FALSE`

  Okurigana processing learning enables learning the previously confirmed Okurigana processing rule and using it for the next conversion. Specify `TRUE` to enable this processing and `FALSE` to disable it.

- `settou_gakusyu TRUE|FALSE`

  Prefix learning enables learning the previously confirmed prefix information (Hiragana or Kanji) and using it for the next converstion. Specify `TRUE` to enable this processing and `FALSE` to disable it.

- `setubi_gakusyu TRUE|FALSE`

  Suffix learning enables learning the previously confirmed suffix information (provide or do not provide) and using it for the next converstion. Specify `TRUE` to enable this processing and `FALSE` to disable it.

- `hanyou_gakusyu TRUE|FALSE`

  General learning enables learning of the previously confirmed general word candidate information (Hiragana, Katakana, or Kanji) and using it for the next conversion. Specify `TRUE` to enable this processing and `FALSE` to disable it.

- `hindo_kakuritu NOT|LOW|NORMAL|HIGH|ALWAYS`

  Specifies the frequency learning extent. Specify `HIGH` to learn quickly, specify `LOW` to learn slowly, specify `ALWAYS` to always learn, and specify `NOT` to never learn.

- `fi_hindo_kakuritu NOT|LOW|NORMAL|HIGH|ALWAYS`

  FI Specifies the FI –related frequency learning extent. Specify `HIGH` to learn quickly, specify `LOW` to learn slowly, specify `ALWAYS` to always learn, and specify `NOT` to never learn.

- `use_hinsi` *hinsi. . .*

Specifies the list of part of speech to be used in conversion.

- `unuse_hinsi` *hinsi. . .*

  Specify the list of part of speech not to be used in conversion.

- `giji_number HAN|ZEN|HANCAN|ZENCAN|KAN|KANSUUJI|KANOLD`

  Specify the first candidate to be given for converting pseudo–numbers. Kanji numbers with unit: `KANSUUJI` Old Kanji numbers with unit: `KANOLD` Half–width numbers with commas: `HANCAN` Full–width number with commas: `ZENCAN` Half–width numbers without commas: `HAN` Full–width number without commas: `ZEN` Kanji without unit: `KAN`

- `giji_eisuu HAN|ZEN`

  Specifies `HAN` to provide half–width conversion candidates first for pseudo–alphabet characters. Specify `ZEN` to provide full–width candidates first.

- `giji_kigou HAN|JIS|ASC`

  Specify `HAN` to provide half–width conversion candidates first for pseudo–symbols, specify `ZEN` to provide full–width candidates first, and specify `ASC` to provide ASCII candidates first.

- `kutouten TRUE|FALSE`

  Specify `TRUE` to input periods as full-width period and full-width comma `FALSE` to input periods as ".".

- `kakko TRUE|FALSE`

  Specify `TRUE` to input full-width parentheses and `FALSE` to input bracket parentheses.

- `kigou TRUE|FALSE`

  Specify `TRUE` to set the symbol input and `FALSE` to set it to "/".

- `autosave` *number*

  Specifies the confirmation operation times to save automatically the learned information. If the confirmation operation is done specified times, the learned information is automatically saved. The default is 50.

The first @`USR` used in arguments in `setdic`, `setjishopath`, or `sethindopath` will be expanded into the startup user name.

EXAMPLES

EXAMPLE 1

```
confirm1
setfuzokugo iwanami/kougo.fzk
set_fi_system_dic iwanami/fisd    usr/@USR/fisd.h        0 -
set_fi_user_dic                       usr/@USR/fiud            0 -
setdic iwanami/kihon.dic            usr/@USR/kihon.h        6 1 0 - - 0
setdic iwanami/symbol.dic            usr/@USR/symbol.h        1 1 0 - - 0
setdic iwanami/tankan.dic                      -                    1 1 1 - - 0
setdic iwanami/tankan2.dic                      -                    1 1 1 - - 0
setdic iwanami/tankan3.dic                      -                    1 1 1 - - 0
setdic iwanami/tel.dic                          -                    1 1 1 - - 0
```

EXAMPLE 1 *(Continued)*

```
setdic iwanami/zip.dic                          -                    1 1 1 - - 0
setdic iwanami/ikeiji.dic                      -                   1 1 0 - - 0
setdic usr/@USR/ud                              -                    15 0 0 - - 0
muhenkan_gakusyuu                          usr/@USR/muhenkan         0 15 - 0
bunsetsugiri_gakusyuu                  usr/@USR/bunsetsu    0     15 - 0
okuri_kijun REGULATION
settou_kouho    KANJI
setubi_gakusyu    TRUE
hanyou_gakusyu    TRUE
hindo_kakuritu    NORMAL
giji_number HAN
unuse_hinsi single Kanji  zip code    telephone number
;;      N nsho hindo len jiri flag jisho sbn dbn_len sbn_cnt
suuji kana eisuu kigou toji_kakko fuzokogo kaikakko
setparam 5 10   2   45  0    80   5     1   20    0    400
-100   400   80    200    2    200
```

SEE ALSO uum(1), xjsi(1), jserver(1M), uumrc(4)

NAME

wnnhosts – Wnn6 Kana-Kanji conversion server/dictionary lookup server access control file

SYNOPSIS

`/etc/lib/locale/ja/wnn/wnnhosts`

DESCRIPTION

`wnnhosts` specifies users who can use the Wnn6 Kana–Kanji conversion server (`jserver`) and the Kana–Kanji conversion server that can be connected to the Wnn6 dictionary lookup server (`wnnds`).

The following is the format of the access control file. Place a space character before "{".

```
jserver        ja    <Kana-Kanji conversion server>    {
<access control data>
    :
}
wnnds        ja    <dictionary lookup server>    {
<access data>
    :
}
```

<Kana–Kanji conversion server> is specified in the following format.

hostname	The Kana–Kanji conversion server that uses the well-known port number (22273) on host *hostname*.
hostname/port_no	The Kana–Kanji conversion server that uses *port_no* as port number on host *hostname*.

<dictionary lookup server> is specified in the following format.

hostname	The dictionary lookup server that uses the well-known port number (26208) on host *hostname*.
hostname/port_no	The dictionary lookup server that uses *port_no* as port number on host *hostname*.

<access control data> for `jserver` is specified in the following format.

hostname	All the users on the host can use data.
hostname:*username_list*	*username_list* contains a list of *username* separated with ",". Users listed in the list on this host can use data.
@username	This user can use data from any host.

<access control data> for `wnnds` is specified in the following format.

hostname `jserver` on this host can be connected.

`jserver` and `wnnds` use the access control information with the host name and port number matched.

Lines beginning with ";" are comments.

EXAMPLES

EXAMPLE 1

```
jserver ja_JP hostA {
;hostC:usr1,usr2,usr3
hostA:usr1,usr4
hostB
hostC:usr5
@usrA
;usrB
}

wnnds ja_JP hostA {
hostA
hostD
}

jserver ja_JP hostA/22273 {
hostB
hostE
@usrA
}

wnnds ja_JP hostA/22385 {
hostA
hostD
}
```

SEE ALSO

jserver(1M), wnnaccess(1M), wnnds(1M)

NAME

wnn_2A_CTRL – Change input conversion mode definition table

SYNOPSIS

```
/usr/lib/locale/ja/wnn/ja/rk/2A_CTRL
```

DESCRIPTION

`2A_CTRL` sets conversion keys that change input conversion mode for `xjsi`(1) and `uum`(1).

Refer to the `wnn_automaton`(4) manpage for `2A_CTRL` description.

EXAMPLES

EXAMPLE 1

```
'\x81'  (switch katakana)    ;PF1 key
'\x82'  (switch zenkaku)    ;PF2 key
'\x83'  (switch romkan)    ;PF3 key
```

SEE ALSO

`uum`(1), `xjsi`(1), `uumkey`(4), `wnn_automaton`(4), `wnn_cvt_key_tbl`(4), `wnn_cvt_xim_tbl`(4),`wnn_mode`(4)

NOTES

`2A_CTRL` is applied for the code processed by `cvt_xim_tbl` or `cvt_key_tbl`.

NAME

wnn_2B_ROMKANA – Roman character Kana conversion definition table

SYNOPSIS

`/usr/lib/locale/ja/wnn/ja/rk/2B_ROMKANA`

DESCRIPTION

`2B_ROMKANA` sets the roman character–Kana conversion rules for `xjsi`(1) and `uum`(1).

Refer to the `wnn_automaton`(4) manpage for `2A_ROMKANA` description.

SEE ALSO

`uum`(1), `xjsi`(1), `wnn_automaton`(4), `wnn_mode`(4)

NAME

wnn_automaton – Automaton

DESCRIPTION

Automaton performs `xjsi` and `uum` roman character–Kana conversion by referring to the entries mapped in a table (called a conversion table). Automaton can replace tables to enable versatile conversion.

Automaton performs three conversions in series according to conversion tables (in the order of preprocessing, main processing, and postprocesing) and outputs the final results. Processing is handled according to conversion tables for each of the three conversions. Atomaton also has a mode function. The mode can be switched to dynamically change the combinations of the three processing stages. Setting the mode and the switchover codes is also performed using conversion tables.

Because the conversion tables are text files, they can be replaced easily. You can use a backspace to return to the previous status after a conversion has been completed and before the next conversion is completed.

Although `xjsi` performs roman character–Kana conversion only between uppercase alphbets and Hiragana in the main processing stage, the preprocessing and postprocessing stages can handle various types of inputs and outputs. For example, the preprocessing can convert lowercase alphabets to uppercase alphabets. The postprocessing stage can convert Hiragana to Katakana or Hiragana to half–width Katakana.

Automaton proceeds with the operation as follows.

1. Input. Upper/lowercase of alphabet (half–width)
2. Preprocessing. Converts lower case characters to uppercase characters.
3. Main processing. Converts uppercase alphabets to Hiragana according to the conversion table.
4. Postprocessing. Converts Hiragana to Katakana or half–width Kana as required.
5. Output

Conversion Tables

Automaton uses the following conversion tables.

- Mode definition table

  Declares the mode and the correspondence tables to use. The file name is `mode`.
- Correspondence tables
  - Preprocessing tables

    The correspondence table used for preprocessing. The file name begins with "1".
  - Main processing tables

    The correspondence table used for main processing. The file name begins with "2".
  - Postprocessing tables

The correspondence table used for postprocessing The file name begins with "3".

The mode definition table contains the mode declaration, the correspondence tables to be used for each mode, and table usage rules for them.

The correspondence tables contain lists of corresponding input codes and output codes. The correspondence tables are separated into those for preprocessing, main processing and postprocessing and any number of correspondence tables can be used for each processing.

`xjsi` searches for the mode definition table in the following order.

1. Specification by `setrkfile` entry in the `xjsi` initialization file `uumrc`
2. File name `/usr/lib/locale/ja/wnn/ja/rk/mode`

The following table entires can be used in the following mode definition table and correspondence tables.

- . . . Indicates repeating 0 or more times.
- . . . . . Indicates one or more times.
- [ ] Indicates optional.

Mode Definition Table

The mode definition table contains the mode declaration, the correspondence tables to be used for each mode, the determination standards for them, and the mode display text strings.

The mode definition table consists of the following items 1, 2, 3, and 4. The remainder of a line is treated as a comment if a semicolon (;) appears at the beginning of the line or follows leading space(s) (including tabs) unless the semicolon is escaped.

1. Special path specification
   The followings are special strings to represent path names.

   `@HOME` Indicates the environment variable `HOME`

   `@MODEDIR` Indicates the directory containing the mode definition table.

   `@LIBDIR` `/usr/lib/locale/ja/wnn/`

   ~user If *user* is a user name, it indicates the user's home directory.

   ~ Indicates your home directory.

2. Mode Declaration
   The mode is declared as follows:

   `defmode` *mode_name* `[on|off]` *mode_name* `[r|nr]`

   specifies alphanumerics. Specifies the initial status. The default is `off`. is a flag to allow Automaton to recognize roman character–Kana conversion mode. In a mode in which `r` is set, it is used regardless of the current mode when converting Hiragana by F6 key, etc. The default is `nr`.

The mode declaration is made before the mode is used.

3. Search Specifications for Correspondence Tables
Search specifications are made for correspondence tables using the following format.

`search` *directory...* Specify the directory name(s) to be searched when the correspondence tables specified in the mode definition table are not in the same directory as the mode definition table. Multiple directory names can be specified by separating them with spaces. The `search` directory name must be specified before the correspondence tables.

`path` *directory...* Overrides any directory names previously stored to search for correspondence tables and specifies to search the directory name(s) specified as the argument. Multiple directory names can be specified by separating them with spaces. The `path` directory name must be specified before the correspondence tables.

4. Specifications for Correspondence Tables and Mode Display Text Strings
There are three ways for the specifications.

(1) Correspondence table file names or mode display text string

(2) if Conditional_expression. Correspondence_table_specifications or Mode_display_text_string

(3) when Conditional_expression. Correspondence_table_specifications or Mode_display_text_string

File names for correspondence tables must begin with (1), (2) or (3). Path names can also be specified. Mode display text strings are text strings enclosed with quotation marks used to display the current mode.

(a) `"`*string*`"`

Indicates the mode display text string when conversion is ON.

(b) (`on_dispmode "`*string*`"`)

Indicates the mode display text string when conversion is ON.

(c) (`off_dispmode "`*string*`"`)

Indicates the mode display text string when conversion is OFF.

(d) (`on_unchg`)

Indicates the same mode display text string as was used before the mode was changed be used when conversion is ON.

(e) (`off_unchg`)

Indicates the same mode display text string as was used before the mode was changed be used when conversion is OFF.

This text string is used by `xjsi` to display the mode.

(2) and (3) are used to change the correspondence table depending on specified conditions. If the condition in the if statement in (2) is true, then the specification in the if statement is referenced and the specification following the if statement is not referenced. If the condition is false, the if statement is exited and the specification following the if statement is referenced.

If the condition in the when statement in (3) is true, the specification in the statement is referenced. The specification following the when statement, however, is referenced regardless of whether or not the condition is true or false.

(2) or (3) can be used recursively to specify correspondence tables.

Any one of the following can be used for the conditional statement.

mode_name	True when the mode is ON
(and conditional_statement conditional_statement)	True when both of the conditional statements are true
(or conditional_statement conditional_statement)	True when either of the conditional statements is true
(not conditional_statement)	True when the conditional statement is false
(false)	Always false
(true)	Always true

For example, when (`defmode kana`) and (`defmode romajikana`) are both in the mode definition, (`and kana romajikana`) is true when both modes are ON.

Where conditional statements are represented by @, #, and *, and conversion table names are represented by A, B, and C, assume the following statement.

```
(when @ A (if # B ) C ) (if * D ) E
```

Also assume that conditional statements @, #, and * have been met. Examine the statement from the beginning. First comes (when @ A (if # B) C). Because @ has been met, "A (if # B) C" is examined and table A is selected.

Next comes (if # B) and # has been met so table B is selected. Because this is an if statement and the conditional statements have been met, the rest of the current series "A (if # B)C" need not be examined. Although this ends examination of "A (if # B)C," this series is contained in a when statement, so the remainder of "(when @ A (if # B)C) (if * D) E" is examined.

The next portion is (if * D). Table D is selected because the condition statement * has been met. Because this is an if statement, the rest of "(when @ A (if # B) C) (if * D) E" is not examined. As a result, tables A, B, and D are selected.

Next we'll use the mode definition tables used by `xjsi` as an example.

Three modes are defined in the mode definition table. There are specifications for correspondence table and mode display text string to be used from `2A_CTRL` to the end. This table is referenced each time the mode changes and the tables to be used are selected as described above.

```
(defmode romkan)
(defmode katakana)
(defmode zenkaku)
2A_CTRL (if romkan
         1B_TOUPPER
         2B_ROMKANA 2B_JIS
          (if (not katakana) "[Ar]")
          (if zenkaku 3B_KATAKANA "[Ar]")
          3B_HANKATA "[AIr]")    ; "A" and "I" are half-width Katakana.
2B_DAKUTEN
(if (not katakana)
         1B_ZENHIRA
         (if zenkaku 3B_ZENKAKU "[A ]")
         "[AA]")
(if zenkaku
         1B_ZENKATA
         3B_ZENKAKU
         "[A ]")
         "[AIA]"    ; "A" and "I" are half-width Katakana.
```

Initially `romkan` , `katakana`, and `zenkaku` are all OFF. `2A_CTRL` is selected as the table at this point. Because romkan is OFF, the following if statement is not referenced, and `2B_DAKUTEN` is selected. The conditional statement for the next if statement, (not katakana), is true because katakana is OFF. The inside of the if statement is referenced and `1B_ZENHIRA` is selected. Next the if statement inside the if statement is referenced. Because zenkaku is OFF, the conditional statement is false. The if statement is thus not referenced.

Next the mode display text string "[A[hiragana-A]]" is selected and the rest of the conversion table series is not examined.

Correspondence Tables

The correspondence tables contain the conversion data (input codes and corresponding output codes) for preprocessing, main processing, and postprocessing.

Preprocessing and postprocessing play supplemental roles for main processing. The following restrictions thus apply to preprocessing and postprocessing correspondence tables.

Preprocessing Table	Item (2), below, is not possible. Also, there can only be one form for each input and output code in item (1) that results in a character when evaluated. The buffer remainder cannot be entered.

Postprocessing Table	Item (2), below, is not possible. Also, there can only be one form for each input code in item (1) that results in a character when evaluated. The buffer remainder cannot be entered.

All lines in the correspondence table must contain one of the following items (1) to (3) or must be empty. Lines of this form are repeated to form the correspondence table.

(1) input_code [output_code [buffer_remainder]]

(2) input_code function

(3) Variable declaration

Each entry must occupy no more than one line. The remainder of a line is treated as a comment if a semicolon (;) appears at the beginning of the line or follows leading space(s) (including tabs) unless the semicolon is escaped.

The output code or buffer remainder will be treated as a null string if omitted. Input codes, output codes, and buffer remainders must contain strings of the following without intervening spaces: forms that evaluate to characters and forms that evaluate to text strings.

Forms are considered to evaluate to characters or text strings if the form is replaced by the character or text string.

The following types of forms evalutate to characters.

(1) Character Notation — Character notations are shown below (these differ from character notations treated as forms that evaluate to text strings).

Character	Characters excluding "(", ")", "'", """, "\", ";", and space characters
'Characters	Characters excluding "'", "\", and "^" characters.
'^Characters	Represents control characters. The characters can be ASCII code 32 to 126. ^? indicates a DEL code.
'\Characters	Characters do not include numerics, "o", "d", and "x". "\n", "\t", "\b", "\r", "\f" indicates the characters same as escaped codes in the C language. "\e", and "\E" indicate escape characters. The other characters are literally interpreted.
'\octal code. . . . .'	Indicates a character corresponding to the specified octal code.
'\ooctal code. . . . .'	Indicates a character corresponding to the specified octal code.
'\ddecimal code. . . . .'	Indicates a character corresponding to the specified decimal code.
'\xhexadecimal code. . . . .'	Indicates a character corresponding to the specified hexadecimal code.

(2) Function Name with Form that Evaluates to a Character

Function name	Description
`toupper`	If the augument is a lowercase alphabet ASCII character, it is changed to an uppercase character. Example: (`toupper a`) -> `A`
`tolower`	If the augument is an uppercase alphabet ASCII character, it is changed to a lowercase character. Example: (`tolower A`) -> `a`
`toupdown`	The case of the alphabet ASCII character is changed from upper to lower or from lower to upper. Example: (`toupdown a`) -> `A` (`toupdown A`) -> `a`
`tozenalpha`	If the argument is an ASCII character, it is converted to a full–width Japanese roman character. Example: (`tozenalpha A`) -> `A`
`tohira`	If the argument is full–width Katakana, it is converted to Hiragana. Example: (`tohira` A) -> A
`tokata`	If the argument is Hiragana, it is converted to full–width Katakana. Example: (`tokata` A) -> A
`tozenhira`	If the argument is half–width Katakana, it is converted to Hiragana. Example: (`tozenhira` A) -> A ; "A" is half-width Katakana.
`tozenkata`	If the argument is half–width Katakana, it is converted to full–width Katakana. Example: (`tozenkata` A) -> A ; "A" is half-width Katakana.
`value`	Converts a character code to its actual numeric value. Example: `value 0` -> `'\x0' value A` -> `'\xA' value F` -> `'\xf'`

(3) Function Name with Two Forms that Evaluate to Characters

Function name	Description
+	Finds the sum of the arguments. Example: (+ A '\d256') -> A (+ 0 (value 3)) → 3
–	Finds the difference of the arguments.
*	Finds the product of the arguments.
/	Finds the quotient of the arguments.

(4) Variable Names — Variable names are any alphanumeric text strings beginning with an alphabet that do not correspond to function names, functions, and declarations (defvar). Where, an underscore '_' is considered as an alphabet.

The following types of forms evalutate to characters.

(1) "Character Notation. . ." — Character notations are shown below (these differ from character notations treated as forms that evaluate to text strings).

Character	Characters excluding """, "^", and "\".
^ character	Indicates a control character. It can be an ASCII code 32 to 126 character. ^? indicates a DEL code.
\Character	Characters do not include numerics, "o", "d", and "x". "\n", "\t", "\b", "\r", "\f" indicates the characters same as escaped codes in the C language.
\octal code. . . . . .[;]	Indicates a character corresponding to the specified octal code. Specify a semicolon (;) if number(s) follow.
\ooctal code. . . . . .[;]	Indicates a character corresponding to the specified octal code. Specify a semicolon (;) if number(s) follow.

\ddecimal code. . . . . .[;] Indicates a character corresponding to the specified decimal code. Specify a semicolon (;) if number(s) follow.

\xhexadecimal code. . . . . .[;] Indicates a character corresponding to the specified hexadecimal code. Specify a semicolon (;) if number(s) follow. " " indicates an empty character string.

(2) Function Name with Form that Evaluates to a Character

Function name	Description
`tohankata`	If the augument is full–width Hiragana or full–width Katakana, it is converted to half–width Katakana. Example: `tohankata` [hiragana–GA]) to [hankakukana–GA]
`last=`	Examines the argument (a form that evaluates to a character) to see if it matches the last character of the last–matched text string. If the character matches, an empty text string is returned. Example: `last=` A → [A] `last=` can only be entered for input codes.
`todigit`	Converts the code given as the first argument to a number in the number base code given as the second argument.
`dakuadd`	Adds a Dakuten (voiced constant mark) after the argument.
`handakuadd`	Adds a Handakuten (semivoiced constant mark) after the argument.

(3) Function Name with Mode Name

The mode name must be defined in the mode definition table.

Function name	Description
`if`	If the mode given as the argument is ON, (`if` *mode_name*) returns an empty string. Example: `(if katakana)VU` [katakana–VU]
`unless`	If the mode given as the argument is OFF, (`unless` *mode_name*) returns an empty string. Example: `(unless katakana)VU` [hiragana–BU]

Function name	Description
`on`	Turns ON the mode given as the argument. Example: `(on katakana)`
`off`	Turns OFF the mode given as the argument. Example: `(off katakana)`
`switch`	Switches the state of the mode given as the argument, i.e., ON to OFF or OFF to ON. Example: `(switch katakana)`

However, `if` and `unless` can only be entered for `on`, `off` and `switch` can only be entered for output codes in the main processing table.

(4) Function Names Only

The following function names can only be entered for output codes in the main processing table.

Function name	Description
`allon`	Turns ON all modes.
`alloff`	Turns OFF all modes.

Precautions on functin usage

Because functions are forms that evaluate to characters or text string, it can be represented as `(toupper (tolower Y))`. However, if evaluated as follows, a function that evaluates to text string cannot be used as arguments for other functions.

```
(toupper (tohankata [Hiragana KA]))
```

Functions

The following functions can be used. These functions can be used independently.

`(error)` An error will be generated if the corresponding input code is received.

`(restart)` The previous mode definition table is read again to reset conversion. If there is an error in the new conversion table, an error message is displayed and the settings in the previous (original) conversion table are used.

Variable Declarations

`(defvar` variable_notation `(list` character_notation. . . . . `))`
`list` uses its arguments as the variable range.

`(defvar` variable_notation `(all))`
`all` uses all the characters as the variable range.

`(defvar` variable_notation `(between` character_notation1 character_notation2))
`between` uses characters between character_notation1 and character_notation2 (both inclusive), when sorted in the code order, as the variable range.

Variables that can be used as forms that evaluate to characters and the range of the variables is defined. Variables are declared in the table that uses it.

Variable notations are given as variable names or as (variable_name. . . . .). Character notations are the same as forms that evaluate to characters.

The variable definitions are effective on the entire table. The same variable cannot be declared twice by `defvar` in a table.

You can define a variable `a1` in a table, and define it in another table with different specifications. Two variables of `a1` are processed independently.

Variables

Variables can be used effectively when the same patterns appear many times in conversions, such as in the following example.

```
(defvar    a1    (list  K S T H Y R W G Z
D B P))
(a1)(a1)    [small tsu]    (a1)
```

The above two lines achieve the same conversions as the following lines. Both show methods of handling assimulated sounds (Sokuon) in roman character–Kana conversion.

KK	[small tsu]	K
SS	[small tsu]	S
TT	[small tsu]	T
. . .		
(omitted)		
PP	[small tsu]	P

The variables declared in the variable declaration are processed.

`(between A E)` and `(list A B C D E)` are the same.

Precautions on variables

Variables to be used must be defined by variable declaration in the table.

You can define the variable `a1` in two different tables as required and the `a1` will be treated as two separate variables. You cannot, however, define the same variable twice in one table. Variable definitions are valid anywhere within the table. You can define the variable `a1` within two different tables as required and the `a1` will be treated as two separate variables. The definitions of variables are effective in the entire table. You cannot, however, define the same variable twice within any one table.

A variable always has the same value within a single line in a correspondence table.

```
(defvar a1 (list  A  B))
    (a1)(tolower (a1)) 3
```

The text strings "`Aa`" and "`Bb`" will be converted to "`3`" in the above example and not to "`Ab`" and "`Ba`".

Input code is matched with input codes in the tables starting at the left. Thus, when examining input codes from the left in the tables, a variable must not be used where it will be treated as the argument of a function before it is matched to specific characters, such as in the following example.

```
(defvar a1 (list a b))
    (toupper (a1))(a1) 3
```

"Aa" will not be converted to "3", because the argument of `(toupper(a1))` is the variable `a1`, which does not yet have a value. This type of setting is checked when tables are read into the system.

In this case, if you make changes as follows, the result will be as expected.

```
(defvar a1 (list a b))
    (a1)(toupper (a1)) 3
```

If "`aA`" and "`bB`" are input, they are converted to "`3 `".

```
(defvar a1 (list A B))
    (a1)(toupper (a1)) 3
```

If "`Aa`" and "`Bb`" are input, they are converted to "`3 `".

Any variable appearing in the output codes or buffer remainder section must appear in the input code section, i.e., must have been assigned a value when matched to an input code.

```
(defvar  a1  (list  K  S))
    (defvar  a2  (list  a))
    (a1)(a1)   (a2)    (a1)
```

The above programming is not correct because the variable `a2` is not matched to an input code, but appears for an output code.

Conversion Method by Correspondence Table

Preprocessing
: First, the code that is input is grouped into character units (characters of 2–byte codes are also treated as one character). This is called the input code.

In preprocessing, each input code corresponds to one output code. The output code from preprocessing becomes the input code for main processing.

Input codes in the preprocessing table currently used are examined in the order from the beginning. When a match is found for the input code, the corresponding output code (i.e., the output code written on the same line as the input code) is output.

If there is more than one table specified in the mode definition table, they are examined in the same order as listed in the mode definition table. If no matching input code is found in a table (including when no table is specified), the input code is output unchanged. This is also true for main processing and postprocessing.

Main processing

In main processing, input code is continuously added to the buffer as long as there is still a chance that a longer match will be found in the input codes in the table (i.e., when some number of characters from the beginning of the current section of input code have already been matched sometwhere in the table).

Each time more input code is added to the buffer, comparisons are again done in the order from the beginning of the input codes listed in the main processing table. As long as there is a chance of the input code in the buffer matching with the longest entry in the table (i.e., when some number of characters from the beginning of the current section of input code have already been matched somewhere in the table) a conversion is not finalized and more input code is awaited. The code in the buffer is, however, output as nonfinalized characters to enable displaying and other processing.

Codes for input errors and mode changes are also output. These codes are differentiated from normal output codes

and do not undergo postprocessing. When the contents of the buffer matches the longest possible input code in the table (if more than one match is made, then the first one in the table is used), the corresponding output code is output. If no buffer remainder has been specified, the part of the buffer that was matched is deleted from the buffer. If a buffer remainder was specified, it replaces the portion in the buffer that was matched and the above operation is repeated.

If no possibility of a match is found in the table, the first character in the buffer is output unchanged. If the output code for a matched input code is a function that changes the mode (`on`, `off`, `switch`, etc.), the correspondence table is changed according to the specifications in the mode definition table. The functions that change the mode should be placed in the tables where they are required regardless of the status of the modes.

If a match is found for the input code corresponding to the function (`restart`), the mode definition table will be reread. However, the same file as the one for the previous mode definition table will be used. This function can be used to change to an edited version of the conversion tables (including the mode definition table) while the Automaton is running without having to stop the Automaton.

Postprocessing

In postprocessing, more than one output code can be output for one input code as the final output. In all the other ways, postprocessing is the same as preprocessing.

In the following example "`ls -la` (carriage_return)" is output when "`Ls`" or "`LS`" is input.

Preprocessing table

```
(defvar   a1   (list s))
(a1)  (toupper (a1))
```

Main processing table

```
LS        "LS -la\n"
```

Postprocessing table

```
(defvar   a1   (all)) (a1)
(tolower (a1))
```

NAME

wnn_cvt_key_tbl – Kana-Kanji conversion front end processor (uum) key code conversion table file

SYNOPSIS

`/usr/lib/locale/ja/wnn/cvt_key_tbl`

DESCRIPTION

`cvt_key_tbl` defines the conversion table for `terminfo` entry and key code. `uum`(1) converts the input strings to key code by using of `terminfo` and `cvt_key_tbl`. If each escape sequence character is input for more than 1 second, `uum` converts it as each separate character.

Syntax

terminfo_entry code

A space character is necessary between *terminfo_entry* and *code*. Lines beginning with a semicolon (`;`) are comments.

terminto_entry The following `terminfo` entries are converted.

kf0	kf1	kf2	kf3	kf4	kf5	kf6
kf7	kf8	kf9	kf10	kf11	kf12	kf13
kf14	kf15	kf16	kf17	kf18	kf19	kf20
kf21	kf22	kf23	kf24	kf25	kf26	kf27
kf28	kf29	kf30	kf31	kbs	ktbc	kclr
kctab	kdch1	kdl1	kcud1	krmir	kel	ked
khome	kich1	kil1	kcub1	kll	knp	kpp
kcuf1	kind	kri	khts	kcuu1		

code

One character excluding space, "\", and "^"	Indicates literal character.
^ *character*	*character* indicates `@`, `A (a)`, `B (b)`, `C (c)`, `D (d)`, `E (e)`, `F (f)`, `. . .`, `Z (z)`, `[`, `\`, `]`, `^`, and _, ^@ indicates control + space (0x00), ^A indicates control + A (0x01), . . ., and ^_ indicates control + _ (0x1f).

\octal number, \ooctal number, \ddecimal number, \xhexadecimal number	Directly specifies character code.
\n, \t, \b, \r, \f, \e, \E	\n indicates newline, \t indicates tab, \b indicates backspace, \r indicates return (RETURN), \f indicates form feed, \e indicates escape (ESC), and \E indicates escape (ESC).
character	Possible *characters* are any characters except the following: `0` through `7`, `o`, `d`, `x`, `n`, `t`, `b`, `r`, `f`, `e`, `E`. \ itself is repreasented by "\\".

EXAMPLES

EXAMPLE 1

```
kf1         \x81
kf2         \x82
kf3         \x83
kf4         \x84
kcud1         \x92
kcub1         \x91
kcuf1         \x90
kcuu1         \x93
```

SEE ALSO

`uum`(1), `uumkey`(4), `wnn_2A_CTRL`(4)

NOTES

Code converted by this table is evaluated by the roman character–Kana conversion Automaton table `2A_CTRL` (default) and then by `uumkey`.

NAME

wnn_cvt_xim_tbl – Key conversion table for xjsi

SYNOPSIS

`/usr/lib/locale/ja/wnn/cvt_xim_tbl`

DESCRIPTION

`cvt_xim_tbl` defines the conversion table for keyboard input and key code.`xjsi`(1) converts keyboard input (*KeySym*) to code (*Wnn_code*) by using of `cvt_key_tbl`.

SYNTAX

State-or-KeySym Wnn_code

State-or-KeySym = [*States*]*KeySym-name*

States = *State-name* | [*States*]

State-or-KeySym and *Wnn_code* must be separated with a space character or tab. Lines beginning with a semicolon (`;`) are comments.

CODE DESCRIPTION

Octal number	`0??`
Decimal number	`??`
Hexadecimal number	`0x??` or `0X??`

EXAMPLES

EXAMPLE 1

```
Meta|Left               0x9A
Meta|Up                0x99
Meta|F11                0x95
Meta|minus             0x81
Meta|asciicircum    0x82
Kanji                    0x81
F1                        0x91
F2                        0x90
Meta|Shift|F1        0x91
```

SEE ALSO

`xjsi`(1), `uumkey`(4), `wnn_2A_CTRL`(4)

NOTES

Code converted by `cvt_xim_tbl` is evaluated by the Automaton table `2A_CTRL` (default) and then by `uumkey`.

NAME

wnn_hinsi.data – Wnn6 part of speech administration file

SYNOPSIS

```
/usr/lib/locale/ja/wnn/ja/hinsi.data
```

DESCRIPTION

`hinsi.data` is a file that contains information required to administer the main parts of speech.

Numbers are allocated in the order of the parts of speech and composite parts of speech defined in `hinsi.data`. These numbers are used when creating dictionary files and part of speech files, when looking up part of speech names by numbers from the client and server, and looking up the parts of speech in composite parts of speech. Numbers are assigned in ascending order starting at 0.

Only the following operations are allowed for this file: appending new parts of speech or composite parts of speech to the end of the file and replacing lines consisting of only "@" with definitions of parts of speech or composite parts of speech. NEVER DELETE ENTRIES. "@" is used to reserve lines in the file in advance when part of speech names have not yet been determined.

The part of speech formats for lines in this file are as follows:

```
part_of_speech
composite_part_of_speech $ part_of_speech : part_of_speech: . . . :part_of_speech
```

All parts of speech appearing in definitions of composite parts of speech must be defined before the composite part of speech can be defined. There must not be more than one part of speech or composite part of speech with the same name.

Everything on a line following a semicolon (`;`) is treated as comment and ignored.

Information on this file (looking up part of speech names from part of speech numbers and looking up the structural elements of composite parts of speech) are provided by the library and can thus be referenced through the client process.

EXAMPLES

EXAMPLE 1

```
[SENTOU]        ;Beginning of a sentence
[MEISHI]        ;Indicates a noun.
[ICHIDAN]
[ICHIDANMEI]$[ICHIDAN]:[MEISHI]    ;A composite part of speech
@
@
```

NOTES

Information on the main parts of speech must be consistent between all dictionaries and connection information files. Do not edit and change `hinsi.data`. (If the file is changed, the meaning of the part of speech numbers in dictionaries and connection information files created with the old part of speech administration file will change.)

NAME

wnn_mode – Mode definition table

SYNOPSIS

```
/usr/lib/locale/ja/wnn/ja/rk/mode
```

DESCRIPTION

`mode` specifies the input definition mode for `xjsi`(1) and `uum`(1), and sets the combinations of correspondence tables used in each input conversion mode and the rules for determining their usage. `mode` correspondes to a "table of contents" for Automaton conversion tables.

Refer to the `wnn_automaton`(4) manpage for `wnn_mode` description.

SEE ALSO

`uum`(1), `uumrc`(1), `xjsi`(1), `wnn_automaton`(4)

NAME

wnn_serverdefs – Wnn6 Kana-Kanji conversion server connection parameter configuration file

SYNOPSIS

```
/etc/lib/locale/ja/wnn/serverdefs
```

DESCRIPTION

`serverdefs` configures the connection between clients and the Wnn6 Kana–Kanji conversion server. The Wnn6 library refers to this file to connect the conversion server. Lines beginning with semicolons (`;`) are comments.

SYNTAX

<LANG_name> <server_host_name> <UNIX_DOMAIN_socket_name> <service_name> <port_number> <environment_variable_name>

<LANG_name>	Specifies the language. `xjsi` and `uum` refers the line containing "`ja`".
<server_host_name>	Specifies the host name of the conversion server.
<UNIX_DOMAIN_socket_name>	Specifies the socket's terminal to use if connection is made to a unix domain socket.
<service_name>	Specifies the name of the tcp service to use if connection is made to an inet domain socket.
<port_number>	Specifies the tcp port number to use if connection is made to an inet domain socket. If the service name is not found, this port number will be used. This port number may be omitted.
<environmental_variable_name>	Specifies the environment variable name set to the host name of the conversion server.

EXAMPLES

EXAMPLE 1

```
ja jserver  /tmp/jd_sockV6   wnn6    22273   JSERVER
```

SEE ALSO

`uum`(1), `xjsi`(1), `jserver`(1M)

NAME

wnn_ximrc – xjsi configuration file

SYNOPSIS

`/usr/lib/locale/ja/wnn/ximrc`

DESCRIPTION

`ximrc` file sets the environment for `xjsi`. Different settings are possible for each user. Entries is set in the following style:

entry setting_value ...entry and *setting_value* are separated by space character or tab. Lines beginning with semicolons (`;`) are comments. If the same entry appears more than once, the last one will be used. Defaults are used for any *setting_value*s that are not set.

`setuumrc` *language uumrc*
Specifies the `uumrc` file that `xjsi` refers to for each language. For Japanese, specify "`ja`". `uumrc` defaults to `@LIBDIR/@LANG/uumrc`.

`preloadrkfile` *language_name*
Specifies the language that loads the Automaton table when starting. `xjsi` ignores this entry.

`setbackspacechar` *backspace_char*
Specifies the backspace character. If it is omitted, `0x7f` is used.

`setposition` *location*
Specifies the position to display the candidate list window.

`over`	window bottom in which input operation is executed
`spot`	inserted position for input characters
`center`	center of the screen

The default is `over`.

`setlayout` *format*
Specifies the format of the candidate list window.

`multi`	Displays the candidate in vertical and horizontal
`vert`	Displays the candidate in vertical
`horiz`	Displays the candidate in horizontal

If ATOK input style is used, the default is `horiz`. Otherwise, `multi` is used as default value. If `multi` is specified in ATOK input style, `horiz` is used. The default is `multi`.

EXAMPLES

EXAMPLE 1

```
;;    sample    ximrc
setposition            spot
setlayout            vert
```

SEE ALSO `xjsi`(1)