CTR

iWnn Programming Manual

2013/04/11

Version 1.0.9

The content of this document is highly confidential and should be handled accordingly.

CTR-06-0160-001-D
Released: May 8, 2013

Table of Contents

Code

Tables

Figures

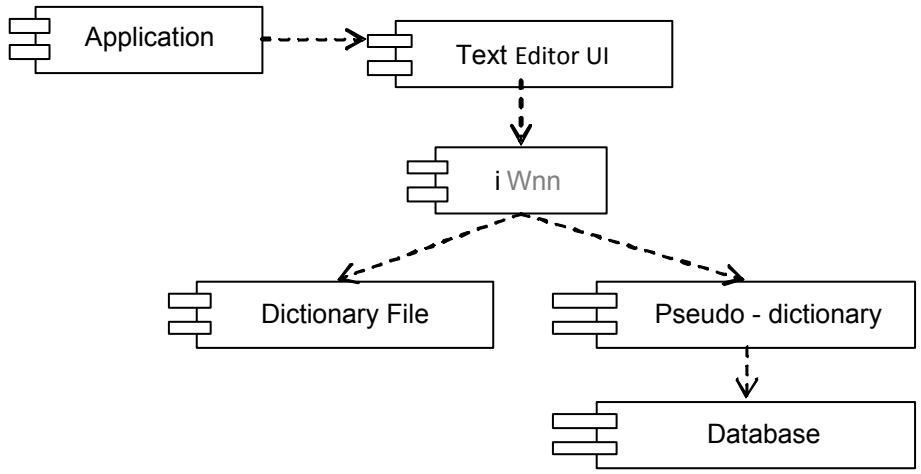
1 What is iWnn?

iWnn is integrated language processing middleware for embedded devices. It provides a full line of language processing features, such as those listed below.

- Multilingual prediction conversion (Japanese, English, Chinese, Korean, and so forth).
- Japanese kana-kanji conversion.
- Japanese morphological analysis (split words, attached readings).
- Dictionary search capabilities.

The following typical relationships apply when embedding iWnn.

Figure 1-1 Typical Relationships When Embedding iWnn



Basically, the iWnn engine provides a conversion feature from a generic text string representing a phonetic value, to a standard text string with the correct surface characters. Various languages can be supported by switching the dictionary. In addition, data in built-in terminal databases (such as a call log) can be used as a conversion dictionary (pseudo-dictionary).

However, a user interface (UI) is not included. A separate module (text editor UI block) must be provided for getting key events and displaying conversion candidates.

2 Glossary

2.1 Readings

The reading is the string input to iWnn. Although the reading is treated as the input string by functions besides those for morphological analysis, the reading has the significance of the attached reading string when using functions for morphological analysis.

- String to be converted using kana-kanji conversion.
  Reading “わたしは” Conversion Candidate “私は”
- Words registered in the dictionary.
  Reading “おむろん” Candidate “オムロン”
- Morphological Analysis.
  Input string “私は田中です” Split words “私は””田中です”
  Readings: “わたしは””たなかです”

2.2 Candidates

This is a list obtained by iWnn, a word registered in the dictionary, or a list obtained by getting predicted candidates.

- List of all candidates obtained from kana-kanji conversion.
  Reading “わたしは” Conversion Candidate “私は”
  Get all candidates Candidate “私は”
  Candidate “わたしは”
  Candidate “渡しは”
  Candidate “ワタシハ”
- Word registered in dictionary.
  Reading “おむろん” Candidate “オムロン”
- List obtained by getting predicted candidates.
  Reading “あ” Search/Conversion Candidate “明日”
  Candidate “亜”

2.3 Phrases

A phrase is a unit made up of an independent word, or an independent word and an ancillary word.

For example, the Japanese sentence below:

- 今日は良い天気ですね

comprises the following phrases.

Table 2-1 Example Configuration of Independent Words and Ancillary Words

Phrase	Independent Word	Ancillary Word
Phrase 1	今日	は
Phrase 2	良	い
Phrase 3	天気	ですね

In order to increase conversion accuracy during kana-kanji conversion and predictive conversion by iWnn, certain ancillary words may be registered as independent words in the dictionary, as a matter of convenience.

Even an ancillary word by itself sometimes creates a phrase during morphological analysis by iWnn.

2.4 Parts of Speech

The part of speech is an attribute used to classify the nature or behavior of a word. This includes parts of speech such as nouns, proper nouns, and so forth.

2.5 Compound Words

A compound word is a word that consists of more than one independent word. If a compound word is registered in the dictionary, it is handled as a single independent word by iWnn.

- Dictionary Registered Word Example 1
  Reading Candidate
  “とうきょうと” “東京都”
  Conversion Result Example
  Phrase 1 “東京都”
  Dictionary Registered Word Example 2
- Reading Candidate
  “とうきょう” “東京”
  “と” “都”

Conversion Result Example

Phrase 1 “東京”

Phrase 2 “都”

2.6 Additional Information

This is additional information, given in text format, which is added to candidates obtained by iWnn.

- Example 1: Chinese Language Example
  Reading “nihao” Candidate “你好” Additional Information “Ni3Hao3”
- Example 2: Phonetic Data Example
  Reading “きょう” Candidate “器用” Additional Information “up1-1, キヨー”

Additional information includes both learnable additional information and unlearnable additional information. Whether or not additional information is learnable can be specified at the time a dictionary is created. (Additional information in a user dictionary or learning dictionary is always learnable.)

2.7 Complete Match Searches (Forward/Reverse Lookup), Prefix Match Searches (Forward Lookup) and Derived Searches

Forward lookup refers to searching for candidates using the reading as a key. Reverse lookup refers to searching for a reading using a candidate as a key.

Forward lookup Reading “わたし” Candidate “私”

Reverse lookup Reading “私” Candidate “わたし”

Complete match search refers to searching for candidates that have the exact same reading string as a specified reading string.

- Complete match search Reading “かく” Candidates “書く” “描く” “各”

Prefix match search refers to searching for candidates that have a reading string that starts with a specified reading string.

- Prefix match search Reading “か” Candidates “課” “科” “回” “傘”

Derived search refers to searching for candidates that follow a specified word.

- Relationship information: “田中”-“太郎”, “田中”-“花子”
- Specified word: “田中” Derived search result: “太郎” “花子”

Table 2-2 Supported Dictionary Formats and Search Functions Available With Each Dictionary

Dictionary Format	Feature	Dictionary
Dictionary for forward lookup complete match search	Forward lookup complete match search feature	Single kanji dictionary (`njtan.a`) Ancillary word dictionary (`njfzk.a`)
Dictionary for forward lookup prefix match searches	Forward lookup complete match search feature Forward lookup prefix match search feature	Customized dictionary
Dictionary for derived searches	Forward lookup complete match search feature Forward lookup prefix match search feature Connection search feature	Learning dictionary
Dictionary for reverse lookup complete match searches	Forward lookup complete match search feature Forward prefix match search feature Reverse lookup complete match search feature	Single kanji dictionary (`njtan.a`)* User dictionary Customized dictionary *
Integrated dictionary	Reverse lookup complete match search feature Reverse lookup prefix match search feature Reverse lookup complete match search feature Reverse lookup prefix match search feature Derived search feature	Single kanji dictionary (`njubase1.a`) Integrated dictionary (`njubase2.a`)

* For versions of iWnn that support morphological analysis.

Note: The rule dictionary (`njcon.a`) and no reading prediction dictionary (`njyomi.a`) are provided in formats other than those listed above.

2.8 Various Dictionaries (Integrated, Single Kanji, Ancillary Word, User, Learning, Rule, Customized, No Reading Prediction)

2.8.1 Integrated Dictionary

This dictionary stores words used in conversion, prediction (standard prediction and connection prediction), and morphological analysis.

The integrated dictionary is provided in two integrated dictionary files (`njubase1.a` and `njubase2.a`).

2.8.2 Single Kanji Dictionary

This dictionary is a collection of single kanji.

The single kanji dictionary is provided as a dictionary binary file (`njtan.a`), for forward lookup complete match or reverse lookup complete match searches.

2.8.3 Ancillary Word Dictionary

This dictionary is a collection of ancillary words.

The ancillary word dictionary is provided as a dictionary binary file (`njfzk.a`), for forward lookup complete match searches.

2.8.4 User Dictionary

The user dictionary allows words to be added using a function for registering words.

The user dictionary is in a form that allows reverse lookup complete match searches. User dictionaries are created using a function for creating a dictionary region in memory.

2.8.5 Learning Dictionaries

Learning dictionaries save learning results.

Learning dictionaries are in a form that allows derived searches. They are created using a function for creating a dictionary region in memory. There are two types of learning dictionaries (AWnn or iWnn). AWnn supports only single word searches.

2.8.6 Rule Dictionary

The rule dictionary saves connection rules for words.

The rule dictionary is provided as a binary file (`njcon.a`).

2.8.7 Customized Dictionaries

Customized dictionaries are for distribution.

There are two types of customized dictionaries: uncompressed dictionaries that allow connection learning and compressed dictionaries.

A compressed dictionary can be used as a dictionary for forward lookup complete match, forward lookup prefix match, or reverse lookup complete match searches. Uncompressed dictionaries are in a form that allows derived searches.

Customized dictionaries can be created using a dictionary binary file creation tool.

2.8.8 No Reading Prediction Dictionary

A no reading prediction dictionary is a collection of words used in situations where no reading has been input.

The no reading prediction dictionary is provided as a binary file (`njyomi.a`).

2.9 Kana-Kanji Conversion

Kana-kanji conversion is a process where independent and ancillary words are searched based on the reading input to the device and the result is converted into a well-formed Japanese sentence.

Table 2-3 Example of Kana-Kanji Conversion

Reading	Kana-Kanji Conversion Result (Multiple Phrase Conversion)
“きょうはよいてんきですね”	“今日は良い天気ですね”

2.10 Morphological Analysis

Morphological analysis takes the input text and parses it into the smallest possible units in Japanese (morpheme).

The morphological analysis capability of iWnn parses the input text into phrase units, and then further parses specified phrases into independent and ancillary words.

Table 2-4 Example of Morphological Analysis

Input Text	Morphological Analysis Result	
“今日は良い天気ですね”	Phrase 1	“今日は”
	Phrase 2	“良い”
	Phrase 3	“天気ですね”

2.11 Pseudo-Dictionaries

In addition to dictionaries formed from standard data files, a program module that returns conversion candidates for an input string can be treated as a pseudo-dictionary.

This type of program module is called a pseudo-dictionary.

2.12 Pseudo-Candidates

Candidates created by the pseudo-candidate creation dictionary module (NjexPseudo) are called pseudo-candidates.

A candidate can be identified as a pseudo-candidate (created through pseudo-candidate creation) whenever a value other than zero results from specifying the operationId member of the processing result structure (`IWNN_RESULT`), to be described later, to the `NJ_GET_PSEUDO_BIT` macro.

However, once a pseudo-candidate has been learned, it cannot be identified as a pseudo-candidate because it will be treated as registered in a dictionary.

2.13 Dictionary Handles

A dictionary handle is the start address of each dictionary used by iWnn, or an interface function pointer to a pseudo-dictionary.

In the case of standard dictionaries, load the dictionary file into a memory region for dictionaries and specify a start address for that memory region.

2.14 Predictions

Prediction is an operation where a search is made for candidates having a reading string that starts with the reading input.

Table 2-5 Example of Prediction

Input Reading	Prediction Candidates
“あ”	“明日” “明後日” “後で”

3 Using iWnn

3.1 Defined Values Set at Compile Time

Specify the following parameters when compiling iWnn from source code.

The parameters are defined in `nj_lib.h`.

Note: The only character encoding supported by all iWnn functions is Unicode (UTF-16BE).

Data types (`u8`, `u16`, `u32`, `s8`, `s16`, `s32`, `wchar_t`, `void`)

Defines 8-bit, 16-bit, and 32-bit data types.

If `int` is a 32-bit type, `s32` is defined as `int` and `u32` is defined as `unsigned int`.

Be sure to specify unsigned short (two bytes) for `wchar_t` when using Unicode (UTF-16BE).

Note: For information on whether or not a data type can be changed, see D.1 Defined Values Set at Compile Time (Data Types).

Table 3-1 Data Types

Macro	Bit Width	C Language Type in a 32-bit Environment (example)
`s8`	Signed 8-bit	char
`s16`	Signed 16-bit	short
`s32`	Signed 32-bit	int OR long
`u8`	Unsigned 8-bit	unsigned char
`u16`	Unsigned 16-bit	unsigned short
`u32`	Unsigned 32-bit	unsigned int OR unsigned long
`wchar_t`	When using UTF-16BE: Unsigned 16-bit	unsigned short
`void`	void type (used in void * declarations)	void
`IWNN_FILE`	File stream type (Used as a file stream type)	void

Note: For information on whether the defined values listed below can be changed, refer to section D.2 Defined Values Set at Compile Time (Defined Values).

3.1.1 Maximum Conversion Reading String Length (NJ_MAX_LEN)

This defines the maximum reading string length to undergo conversion/prediction, and the maximum string length of words added to the learning dictionary.

The string length given specifies the number of elements in the `wchar_t` array. (This is specified in 2-byte units when using UTF-16BE.) The number of characters is configured, excluding the terminal character.

If you reduce the maximum reading string length, adjustments must be made so that none of the candidates in the dictionary exceed the maximum reading string length.

Minimum: 20, Maximum: `NJ_MAX_RESULT_LEN`, Default: 40

3.1.2 Maximum Conversion Candidate String Length (NJ_MAX_RESULT_LEN)

This defines the maximum string length for conversion result strings and the maximum word candidate string length that can be registered in the learning dictionary.

It specifies the number of elements in the `wchar_t` array, excluding the terminal character.

Minimum: 40, Maximum (when using UTF-16BE): 55, Default: 40

3.1.3 Maximum Additional Information Character Array Length (NJ_MAX_ADDITIONAL_LEN)

This defines the maximum string length for learnable additional information strings. It specifies the number of elements in the `wchar_t` array, excluding the terminal character.

Minimum: 20, Maximum (when using UTF-16BE): 55, Default: 40

3.1.4 Maximum Number of Pieces of Additional Information that can be Mounted (NJ_MAX_ADDITIONAL_INFO)

This defines the maximum number of pieces of additional information that can be set for a single dictionary.

Minimum: 1, Maximum: 5, Default: 2

3.1.5 Maximum Number of Obtainable Candidates (NJ_MAX_CANDIDATE)

This defines the maximum number of obtainable candidates when getting all candidates or getting prediction candidates.

Minimum: 100, Maximum: 500, Default: 300

3.1.6 Maximum User Dictionary Word Registration String Length (NJ_MAX_USER_LEN)

This defines the maximum string length for reading strings to be registered in the user dictionary.

It specifies the number of elements in the wchar_t array, excluding the terminal character.

Minimum: 20, Maximum: NJ_MAX_LEN, Default: 40

3.1.7 Maximum User Dictionary Word Registration Candidate String Length (NJ_MAX_USER_CANDIDATE_LEN)

This defines the maximum string length for candidate strings to be registered in the user dictionary.

It specifies the number of elements in the wchar_t array, excluding the terminal character.

Minimum: 20, Maximum: NJ_MAX_RESULT_LEN, Default: 40

3.1.8 Maximum User Additional Information String Length (NJ_MAX_USER_ADDITIONAL_LEN)

This defines the maximum string length for additional information strings that can be set in a user dictionary that includes additional information.

It specifies the number of elements in the wchar_t array, excluding the terminal character.

Minimum: 20, Maximum: NJ_MAX_ADDITIONAL_LEN, Default: 40

3.1.9 Maximum Number of Registerable Words in a User Dictionary (NJ_MAX_USER_COUNT)

This defines the maximum number of words that can be registered in a user dictionary.

Minimum: 10, Maximum: 255, Default: 100

3.1.10 Maximum Number of Mountable Dictionaries (NJ_MAX_DIC)

This defines the maximum number of dictionaries that can be specified in a dictionary set.

Although a value of up to 100 may be set for the maximum number of mountable dictionaries, note that performance in terms of speed can be expected to go down as the number of dictionaries goes up.

Minimum: 4, Maximum: 100, Default: 20

3.1.11 Maximum Morphological Analysis String Length (MM_MAX_MORPHOLIZE_LEN)

This defines the maximum string length for the input string to undergo morphological analysis.

It specifies the number of elements in the wchar_t array, excluding the terminal character.

Minimum: 40, Maximum (when using UTF-16BE): 50, Default: 100

3.1.12 Maximum Number of Same Reading Dictionary Lookups during Multiple Phrase Conversion (NJ_MAX_GET_RESULTS)

This defines the maximum number of dictionary lookups to make during processing of the function for multiple phrase conversion.

Although reducing this value can increase processing speed during multiple phrase conversion, conversion precision is reduced.

Minimum: 5, Maximum: 32, Default: 32

3.1.13 Maximum Ancillary Word Parsing String Length when Getting All Candidates (NJ_MAX_ANCILLARY_LEN)

This specifies how many words at the end of an input reading string to parse as an ancillary word, during processing to get all candidates in the function for getting all candidates and the function for prediction.

Although decreasing this value can increase the processing speed of getting all candidates, it may not be possible to get some candidates in cases where the ancillary word part of the input string is too long.

Minimum: 0, Maximum: NJ_MAX_LEN/2, Default: NJ_MAX_LEN/2

3.1.14 Maximum Number of Registerable Fuzzy Characters (NJ_MAX_CHARSET)

This defines the maximum number of registrations that can be specified for the fuzzy character set (IWNN_CHARSET).

Minimum: 1, Maximum: 255, Default: 50

3.1.15 Maximum Cache Size (NJ_SEARCH_CACHE_SIZE)

This defines the maximum cache size that can be specified for a dictionary set.

Fuzzy prediction searches may become impossible, if this value is set too low when input strings are long.

Minimum: 10, Maximum: 1000, Default: 50

3.1.16 String Terminator Size (NJ_TERM_SIZE)

Set a value of 1 for the size of the terminator for strings.

3.1.17 Maximum Number of Obtainable Candidates (NJ_MAX_CANDIDATE), Maximum Length of Conversion Reading Strings (NJ_MAX_LEN)

Increasing the values set for the maximum morphological analysis string length (MM_MAX_MORPHOLIZE_LEN) and the maximum conversion candidate string length (NJ_MAX_RESULT_LEN) will increase the amount of (fixed-size) memory used by iWnn to store strings.

3.2 Including Header Files

Be sure to include the following header file in all applications that use iWnn.

`mw\iwnn\iwnnCTR.h`

3.3 Access to Dictionary Files (OnMemory Version Only)

iWnn operates by deploying dictionary data to memory, but the following API functions access the dictionary data files.

1. **`NjxGetFlashDicCacheSize`**
2. **`NjxSetFlashDicInfo`**
3. **`NjxCheckDic`**

Note: The **`NjxGetFlashDicCacheSize`** and **`NjxSetFlashDicInfo`** functions are used when deploying integrated dictionaries to memory. Before calling these API functions, it is necessary to set the dictionary data file handle to the `IWNN_FILE` structure.

There is no further access to the dictionary files after the API functions above are called, so close the file handle.

4 List of Used Structures and Functions

4.1 List of Structures

The following structures are used in the iWnn API.

4.1.1 Dictionary Sets (IWNN_DIC_SET, IWNN_DIC_INFO, IWNN_FLASH_DIC_INFO)

These specify the dictionaries to be used by iWnn.

4.1.2 Processing Result (IWNN_RESULT)

This stores the processing results of dictionary lookups, conversions, predictions, and morphological analyses.

4.1.3 Dictionary Search Cursor (IWNN_CURSOR)

This specifies search conditions. Search results are stored by calling the word search function.

4.1.4 Word Information (IWNN_WORD_INFO)

This specifies the reading string, candidate string, and part of speech group to be registered in the user dictionary,

4.1.5 Fuzzy Character Set (IWNN_CHARSET)

This specifies the fuzzy character pattern to be used to perform a fuzzy search.

4.1.6 Parsing Information Class (IWNN_CLASS)

This is a work area used by iWnn.

4.1.7 Option Settings (IWNN_OPTION)

This specifies operational parameters of iWnn.

4.1.8 Prediction Options (IWNN_ANALYZE_OPTION)

This specifies operational parameters to use during prediction and conversion.

4.1.9 State Settings (IWNN_STATE)

This sets the peripheral state required to perform situational applicable prediction.

4.1.10 Merge Candidates (IWNN_MERGE_RESULT)

This stores candidate list information where multiple process result strings have been merged.

4.2 List of Functions

The following functions can be used with the iWnn API.

4.2.1 Initialization (NjxInit)

This initializes iWnn variables.

4.2.2 Get Reading String (NjxGetStroke)

This gets a reading string based on the processing result structure.

4.2.3 Get Candidate String (NjxGetCandidate)

This gets a candidate string based on the processing result structure.

4.2.4 Get Dictionary Handle (NjxGetDicHandle)

This gets a dictionary handle based on the processing result structure.

4.2.5 Create Dictionary Region (NjxCreateDic)

This writes header information into the user dictionary and learning region, and initializes the dictionary.

4.2.6 Check Dictionary (NjxCheckDic)

This checks dictionary integrity and performs automatic recovery.

4.2.7 Get Character Type (NjxGetCharType)

This determines the character type (hiragana, half-width/full-width katakana, half-width/full-width numeric characters) for independent and ancillary words.

4.2.8 Change Dictionary Type (NjxChangeDicType)

This changes the dictionary type (header information) of learning dictionaries and uncompressed customized dictionaries.

4.2.9 Get Prediction Candidate (NjxAnalyze)

This gets one candidate at a time from a list of candidates comprehensively evaluated in terms of kana-kanji conversion of the reading string, all candidate processing, and history search.

4.2.10 Kana-kanji Conversion (NjxConversion)

This performs kana-kanji conversion (multiple phrase conversion or single phrase conversion) of a reading string. Conversion where the locations of phrase delimiters are specified is also possible.

4.2.11 Get All Candidates (NjxAllCandidates)

This gets one candidate at a time from a list of candidates for the specified phrase.

4.2.12 Learn (NjxSelect)

This learns the specified processing result structure.

4.2.13 Undo Learning (NjxUndo)

This deletes the information just learned.

4.2.14 Search Word (NjxSearchWord)

This searches the dictionary based on the specified reading string.

4.2.15 Get Word (NjxGetWord)

This gets one word at a time, based on single word search results.

4.2.16 Register Word (NjxAddWord)

This registers the specified word to a user dictionary or learning dictionary.

4.2.17 Delete Word (NjxDeleteWord)

This deletes a word from a user dictionary or learning dictionary, based on the result of getting a word.

4.2.18 Delimited Input (MmxSplitWord)

This performs morphological analysis on the specified string and splits it into multiple phrase units.

4.2.19 Get Part of Speech Group (MmxGetPartsOfSpeech)

This determines the part of a speech group: such as a noun group, a pseudo-group, and so forth.

4.2.20 Get Reading String for Morphological Analysis (MmxGetReading)

This gets one reading string at a time from a list of readings for a delimited phrase.

4.2.21 Learn by Morphological Analysis (MmxSelect)

This learns words and places them in the learning dictionary, so that morphological analysis results can be re-used.

4.2.22 Set Options (NjxSetOption)

This sets iWnn operational parameters in the parsing information class (IWNN_CLASS).

4.2.23 Set State (NjxSetState)

This sets state parameters, for situational applicable prediction.

4.2.24 Get State Setting (NjxGetState)

This returns the current state setting parameter stored within iWnn.

4.2.25 Get Word Information (NjxGetWordInfo)

This gets information for word registration from the processing result structure.

4.2.26 Get No Conversion Candidates (NjxGetStrokeWord)

This creates a phrase having the same notation as the reading.

4.2.27 Merge Candidate Lists (NjxMergeWordList)

This merges multiple candidate lists to form a single list.

4.2.28 Manage Learning Dictionary (NjxManageLearnDic)

This performs various types of management on the learning dictionary. It is used when merging learning dictionaries.

4.2.29 Get Additional Information String (NjxGetAdditionalInfo)

This gets additional information strings from the processing result structure.

4.2.30 Check Additional Information Region (NjxCheckAdditionalInfo)

This checks if there is an additional information region for the dictionary handle.

4.2.31 Get FLASH Dictionary Cache Size (NjxGetFlashDicCacheSize)

This gets the cache size required by the FLASH dictionary.

4.2.32 Set FLASH Dictionary Information (NjxSetFlashDicInfo)

This sets the required information for the FLASH dictionary.

5 Expiration Period and Applicable Scope of the Process Result Structure

There is an expiration period in place for the processing result structure (`IWNN_RESULT`). Applications cannot pass processing results that have exceeded their expiration period to iWnn functions. Processing results are valid from the point they are obtained by an iWnn function until a function that affects the learning dictionary or user dictionary is called. If a function that affects the learning dictionary or user dictionary is called, all processing results obtained before that point are invalidated. However, this restriction does not apply to multiple phrase conversion processing results obtained by the `NjxConversion` function (processing results for multiple phrases for which parsing was possible).

Figure 5-1 Example of Timing for Getting Process Results

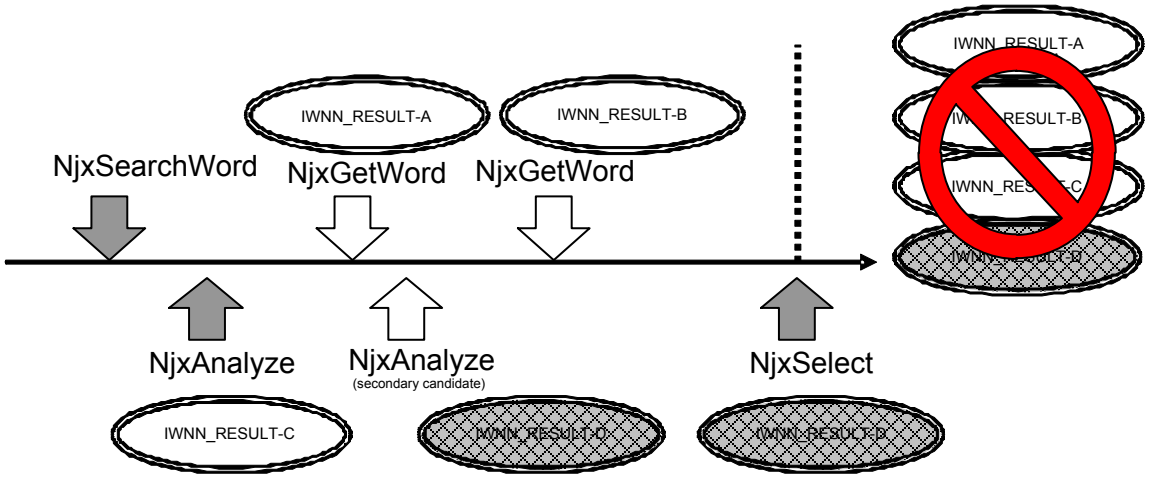


Figure 5-1 is an example of the timing for getting the processing result (`IWNN_RESULT`), when searching the dictionary using the word search function (**`NjxSearchWord`**) and get word function (**`NjxGetWord`**) in combination with the prediction function (**`NjxAnalyze`**). When the learning function (**`NjxSelect`**) is used, the processing results expire and all processing results (`IWNN_RESULT-A` through `IWNN_RESULT-D`) can no longer be used.

When processing results become invalid, they must be obtained again from the engine.

The following functions affect learning dictionaries and user dictionaries.

- Initialize function (**`NjxInit`**).
- Learn function (**`NjxSelect`**).
- Add word function (**`NjxAddWord`**).
- Delete word function (**`NjxDeleteWord`**).
- Undo learning function (**`NjxUndo`**).
- Create dictionary region function (**`NjxCreateDic`**).
- Change dictionary type function (**`NjxChangeDicType`**).
- Check dictionary function (**`NjxCheckDic`**).

The expiration period for the processing result structure (IWNN_RESULT) is not checked by iWnn functions, and must, therefore, be managed by applications.

5.1 Applicable Scope of the Processing Result Structure

The functions that can be used after processing results (IWNN_RESULT) have been obtained depend on the function that obtained them.

Table 5-1 Applicable Scope of the Processing Result Structure

Function Getting Processing Results	Usable Functions	Unusable Functions
NjxAnalyze	NjxGetStroke NjxGetCandidate NjxGetAdditionalInfo NjxSelect NjxGetDicHandle NjxGetCharType NjxDeleteWord	NjxAllCandidates MmxGetPartsOfSpeech MmxGetReading MmxSelect
NjxConversion	NjxGetStroke NjxGetCandidate NjxGetAdditionalInfo NjxSelect NjxGetDicHandle NjxAllCandidates NjxGetCharType	NjxDeleteWord MmxGetPartsOfSpeech MmxGetReading MmxSelect
NjxAllCandidates	NjxGetStroke NjxGetCandidate NjxGetAdditionalInfo NjxSelect NjxGetDicHandle NjxAllCandidates NjxGetCharType	NjxDeleteWord MmxGetPartsOfSpeech MmxGetReading MmxSelect
NjxGetWord	NjxGetStroke NjxGetCandidate NjxGetAdditionalInfo NjxSelect NjxGetDicHandle NjxDeleteWord NjxGetCharType	NjxAllCandidates MmxGetPartsOfSpeech MmxGetReading MmxSelect
MmxSplitWord	MmxGetPartsOfSpeech MmxGetReading MmxSelect NjxGetCandidate NjxGetAdditionalInfo NjxGetDicHandle NjxGetCharType	NjxDeleteWord NjxAllCandidates NjxGetStroke NjxSelect

6 Handling Strings

6.1 Definition of a String

Strings are declared as a `wchar_t` array. iWnn supports Unicode (UTF-16BE).

When using UTF-16BE, strings are handled as an unsigned short array.

Strings are terminated by 0x0000. Strings do not include a byte order mark (BOM).

For example, when specifying the reading "わたし" in UTF-16BE:

Table 6-1 `wchar_t` (unsigned short) Array

Array Index	Value	
	Big-Endian Environment	Little-Endian Environment
0	0x308f	0x8f30
1	0x305f	0x5f30
2	0x3057	0x5730
3	0x0000	0x0000

6.2 Counting the Length of Strings

With iWnn, string length is counted in `wchar_t` units.

Unless otherwise specified, the string length is the same as the number of elements in the `wchar_t` array.

When using UTF-16BE, `wchar_t` is defined as an unsigned short array, where every two bytes count as a single character.

With UTF-16, characters included in UCS-2 are represented by two bytes, but some UCS-4 characters not included in UCS-2 (U+10000 through U+10FFFF) are represented by four bytes (surrogate pairs).

With iWnn, the string length of characters represented by a surrogate pair is 2, and all other characters are counted as 1.

6.3 Notes on Input Strings

The address of an input string specified to an iWnn function is used in processing results (`IWNN_RESULT`). The memory in which input strings are stored must, therefore, continue to be maintained until processing results (`IWNN_RESULT`) are no longer needed.

6.4 Definitions of Hiragana, Katakana, and Numeric Characters

These definitions are used to determine the character type for functions that get character types.

The following full-width symbols are treated as auxiliary hiragana characters. Such “auxiliary characters” are treated as hiragana characters, even if auxiliary symbols exist within full-width hiragana text.

The same is true for full-width katakana text. However, text consisting entirely of auxiliary characters cannot be identified as being either hiragana or katakana text.

(See the appendix for information on hiragana, katakana, and half-space kana characters.)

，	。	、	．	？	！	゛	゜	ー	～	・	＆	＝

The following half-space symbols are treated as auxiliary characters for half-space kana. However, text consisting entirely of auxiliary characters cannot be identified as half-space kana text.

，	。	、	．	？	！	ﾞ	ﾟ	－	～	・	＆	＝	-

The following full-space symbols are treated as auxiliary characters for full-space numeric characters. However, text consisting entirely of auxiliary characters cannot be identified as full-space numeric character text. Furthermore, it cannot be identified as full-space numeric character text, even if auxiliary characters come at the end of the full-space numeric characters.

，	．

The following half-space symbols are treated as auxiliary characters for half-space numeric characters. However, text consisting entirely of auxiliary characters cannot be identified as half-space numeric character text. Furthermore, it cannot be identified as half-space numeric character text, even if auxiliary characters come at the end of the half-space numeric characters.

,	.

7 Operations Overview

This chapter describes the basic use of iWnn functions.

For settings and a detailed description, refer to Chapter 9 Detailed Descriptions of Functions.

The return values of iWnn functions have been standardized, so that a value greater than 0 indicates normal termination and a negative value indicates an error. In the case of an error, the type of error can be determined by the return value. For more information, refer to Chapter 10 Errors.

7.1 Startup

Create dictionaries, check dictionaries, and initialize variables, according to the procedure given below, before executing morphological analyses, predictions, conversions, or dictionary lookups.

1. Create dictionary region function (**`NjxCreateDic`**).
2. Create a user dictionary and learning dictionary, when power is turned on the first time after shipment from the factory.

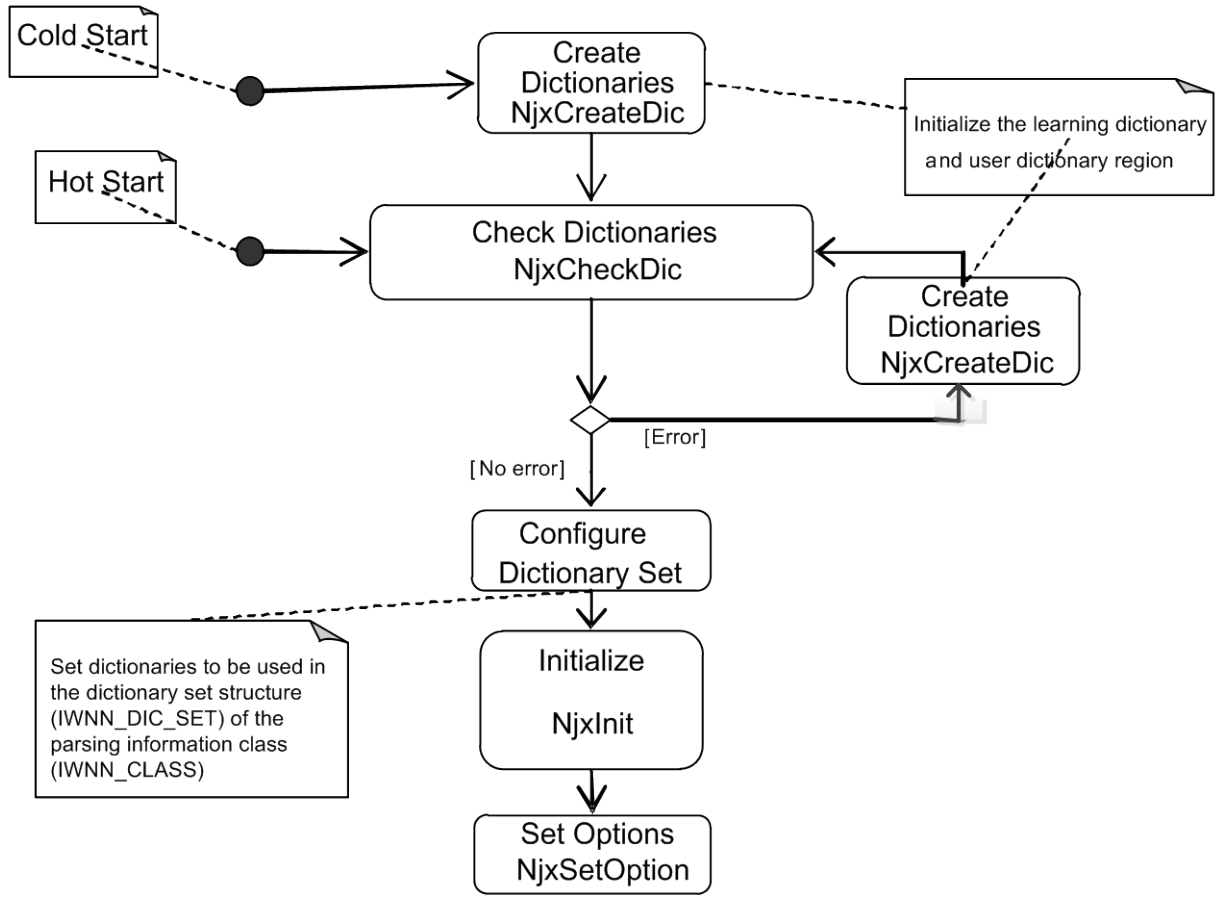
Note: For a supplemental description of the timing at which dictionaries are created, refer to section D.3 Creating Dictionaries.

This function initializes dictionaries based on the configured dictionary handle and size.

3. Check dictionary function (**`NjxCheckDic`**).
   A dictionary check is performed on downloaded dictionaries and/or dictionaries loaded from a non-volatile device. If the user dictionary and learning dictionary are destroyed, automatic recovery is performed.
4. Configure a dictionary set.
   Configure the dictionaries to be used in the dictionary set structure (`IWNN_DIC_SET`) in the parsing information class (`IWNN_CLASS`). As for integrated dictionaries and no reading prediction dictionaries that have additional information, be sure to execute the check additional information region function (`NjxCheckAdditionalInfo`) before setting additional information.
5. Initialization function (**`NjxInit`**).
   This function initializes the iWnn work area. Always call this function when changing the configuration of a dictionary set.
6. Set options function (**`NjxSetOption`**).
   This function sets iWnn operational parameters.

Note: This operation is unnecessary if using default settings.

Figure 7-1 Startup



Note: For a supplemental description of cold start and hot start, refer to section D.4 Cold Start/Hot Start.

Perform initialization using the create dictionary region function when using a user dictionary or learning dictionary region for the first time. If using a user dictionary or learning dictionary that has already been created, the integrity of the dictionary is checked using the dictionary check function. If there is an error, recovery is performed. If using a downloaded dictionary, the integrity of the dictionary is checked using the check dictionary function after it is downloaded.

Note: Distinguishing between a FLASH Dictionary and a non-FLASH Dictionary

The process used to set information in the dictionary set structure (`IWNN_DIC_SET`) differs depending on whether the dictionary in question is a FLASH or a non-FLASH type dictionary. If the dictionary type is not already known, call the get FLASH dictionary cache size function (`NjxGetFlashDicCacheSize`) and determine it to be a non-FLASH dictionary if `NJ_ERR_PARAM_TYPE_INVALID` is returned. (If determined to be a FLASH dictionary, next call the Set FLASH dictionary information function (`NjxSetFlashDicInfo`) and set information required for a FLASH dictionary in the FLASH dictionary information structure.)

Note: For information on the significance of FLASH and non-FLASH dictionaries, refer to section D.5 FLASH Dictionaries/Non- FLASH Dictionaries.

7.2 From Getting Prediction Candidates to Learning

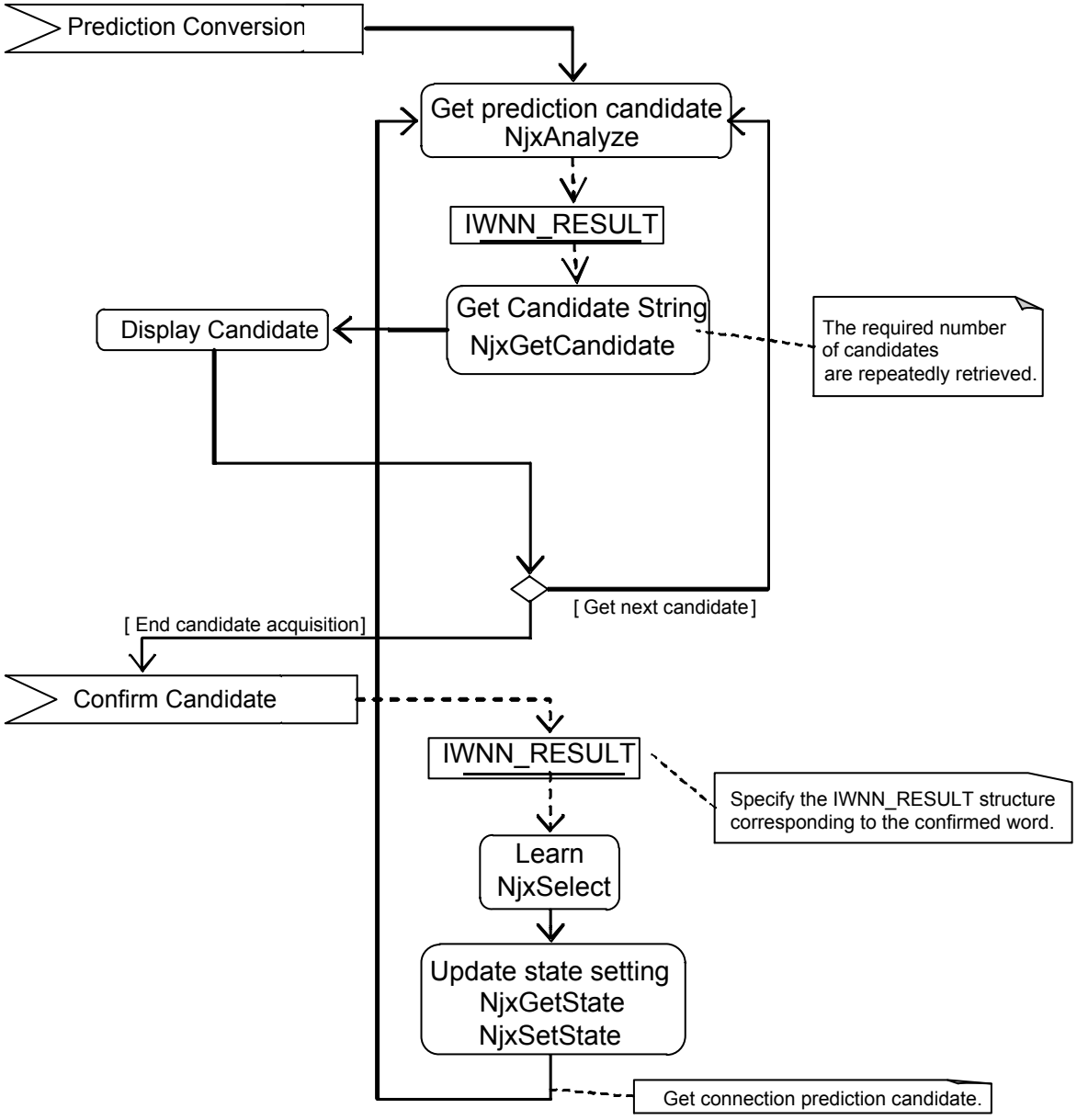
Perform operations from getting prediction candidates to learning according to the procedure given below.

1. Get prediction candidate information function (**`NjxAnalyze`**).
   Get prediction candidates from the reading.
   Obtained prediction candidates are stored in the processing result structure (`IWNN_RESULT`).
   Because this function returns results one at a time, you can get as many candidates as required by repeatedly calling this function while checking the return value.
   To get a candidate string from obtained results (`IWNN_RESULT`), use the get candidate string function (`NjxGetCandidate`).

2. Learn function (**`NjxSelect`**).
   Specify the processing result (`IWNN_RESULT`) obtained by the get prediction candidate function, to learn the stored information.

3. Set state function (**`NjxGetState`**/**`NjxSetState`**).
   Access the state setting value updated internally by iWnn, and update to the appropriate setting value.
   If the learn function has been called, iWnn adds 1 to the state setting value of attributes that have a confirmed candidate.
   The application must reset the state setting by referring to the difference in state setting values.

Figure 7-2 Learning From Obtained Prediction Candidates



To use the fuzzy prediction feature, you must first set the fuzzy character set (`IWNN_CHARSET`) and the cache management region for the dictionary that will undergo fuzzy prediction search.

7.3 From Multiple Phrase Conversion to Learning

Operations from multiple phrase conversion until learning are performed according to the following procedure. (The same is true for single phrase conversion.)

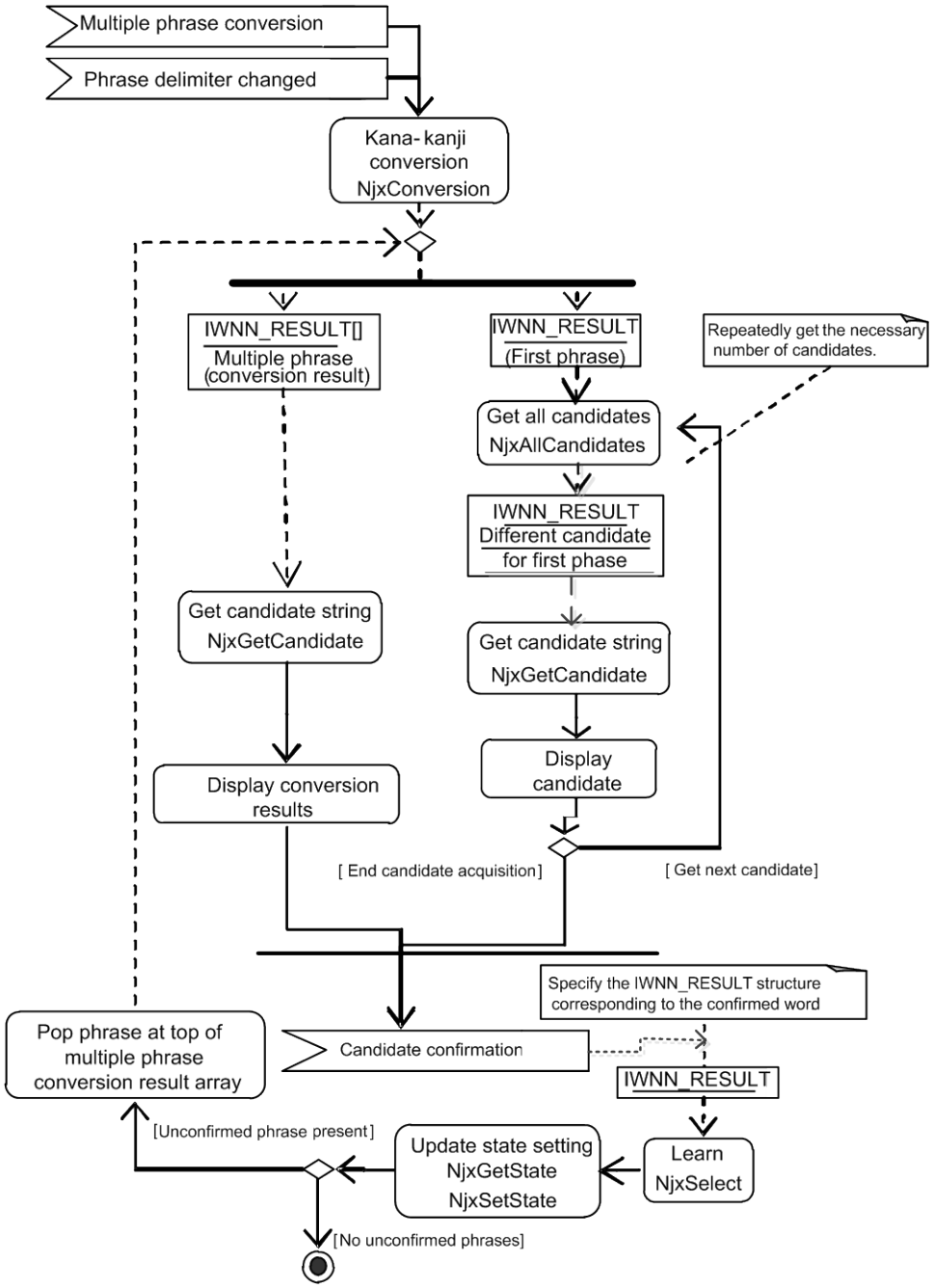
1. Kana-kanji conversion function (**`NjxConversion`**).
   Get the result of multiple phrase conversion of the specified reading string.
   Conversion results are stored in an array of processing result structures (`IWNN_RESULT`).
   When the user changes the location of any delimiters in the conversion, specify the delimiter location and call this function again. In the case of single phrase conversion, specify the delimiter position for the first phrase as the end of the text and call this function.
   The candidate string of the processing result can be obtained using the get candidate string function.

2. Get all candidates function (**`NjxAllCandidates`**)
   Specify the processing result structure (`IWNN_RESULT`) for the first phrase obtained by the kana-kanji conversion function and get words having the same sounding but different notation.
   The candidate is stored in the processing result structure (`IWNN_RESULT`). Because this function returns candidates one at a time, you can get as many candidates as required by repeatedly calling this function while checking the return value.
   The candidate string of the processing result can be obtained using the get candidate string function.

3. Learn function (**`NjxSelect`**)
   Specify the processing result (`IWNN_RESULT`) obtained by the get all candidates function, to perform learning.
   If multiple phrase conversion has been performed, processing results (`IWNN_RESULT`) for multiple phrases are learned in phrase order.

4. Set state function (`NjxGetState`/**`NjxSetState`**)
   Access the state setting value updated internally by iWnn, and update to the appropriate setting value.
   When the learn function has been called, iWnn adds 1 to the state setting value of attributes that have a confirmed candidate.
   The application must reset the state setting by referring to the difference in state setting values.

Figure 7-3 From Multiple Phrase Conversion to Learning



7.4 No Conversion Confirmation

Cases in which characters input from the keyboard are directly confirmed and then learned, without selecting a conversion candidate, are handled according to the following procedure.

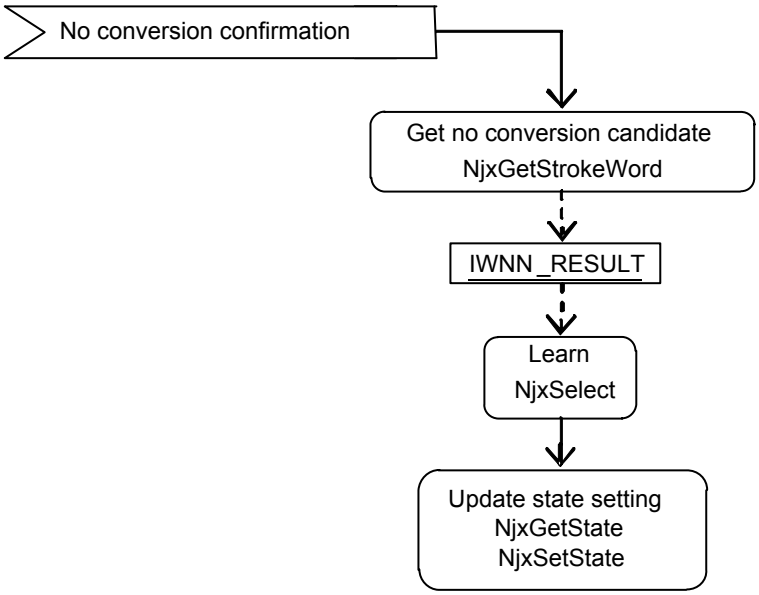
1. Get no conversion candidate function (**`NjxGetStrokeWord`**).

   A reading string is specified and a single phrase candidate having the same notation as the reading string is obtained.

2. Learn function (**NjxSelect**).
   Learn the results (IWNN_RESULT) obtained from the get no conversion candidate process.

3. Set state function (**NjxGetState**/**NjxSetState**).
   Access the state setting value updated internally by iWnn, and update to the appropriate setting value.
   If the learn function has been called, iWnn adds 1 to the state setting value of attributes that have a confirmed candidate.
   The application must reset the state setting by referring to the difference in state setting values.
   The appropriate part of speech information can be obtained for directly confirmed strings by executing a get no conversion candidate process. This allows you to obtain more appropriate connection candidates when getting connection prediction candidates for the immediately following phrase.

Figure 7-4 Confirming No Conversion



7.5 Search Registered Words in Dictionary and Get List

Operations for searching for words registered in a user or learning dictionary and getting a list, are performed according to the procedure given below.

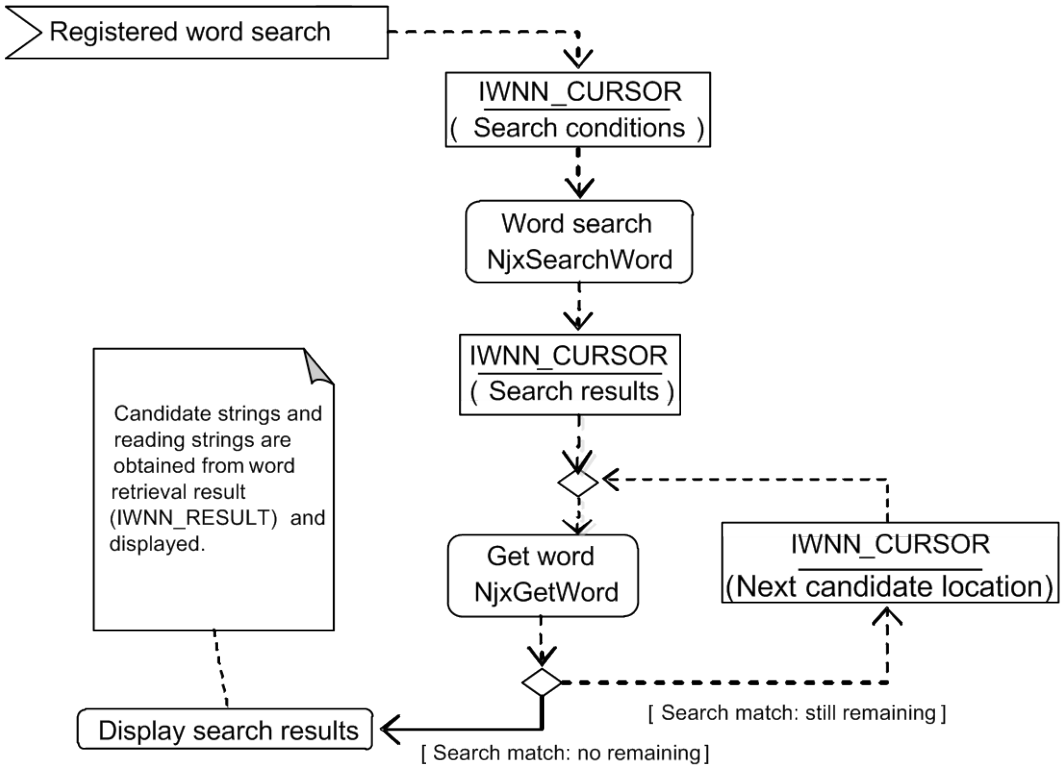
1. Search word function (**NjxSearchWord**).
   Set search conditions in both the dictionary search cursor structure (IWNN_CURSOR) and the dictionary set structure (IWNN_DIC_SET), and search for words.
   Search results are stored in the specified dictionary search cursor structure (IWNN_CURSOR).

2. Get word function (**NjxGetWord**).

Specify the dictionary search cursor structure (`IWNN_CURSOR`) obtained using the search word function and get candidates. Processing results (candidates) are stored in the processing result structure (`IWNN_RESULT`).

The reading string for the obtained result can be obtained using the get reading string function (**`NjxGetStroke`**). The candidate string can be obtained using the get candidate string function (**`NjxGetCandidate`**).

Figure 7-5 Searching for Registered Words in the Dictionary and Getting a List



Sometimes the same word is registered in the learning dictionary more than once. When using the get word function, these words will be retrieved separately. So be sure your application handles this by gathering duplicate words together, as necessary.

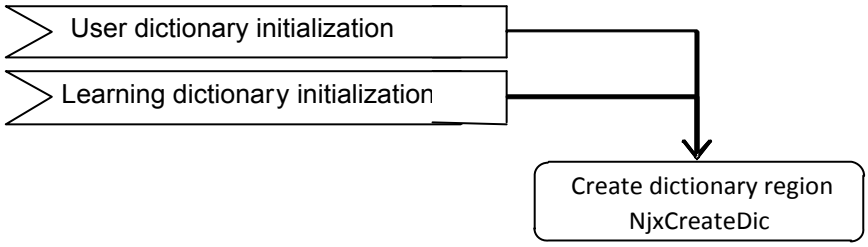
Note: When the content of the learning dictionary or user dictionary has changed, the content of the search result dictionary search cursor structure (`IWNN_CURSOR`) and the processing results (`IWNN_RESULT`) of the get word function will become invalid. Be careful not to write to the learning dictionary or user dictionary during search operations.

7.6 User Dictionary/Learning Dictionary Initialization

Initialization of the user dictionary and learning dictionary region is performed according the procedure given below.

1. Create dictionary region function (**`NjxCreateDic`**).
   Initializes the user dictionary or learning dictionary region.

Figure 7-6 User Dictionary/Learning Dictionary Initialization

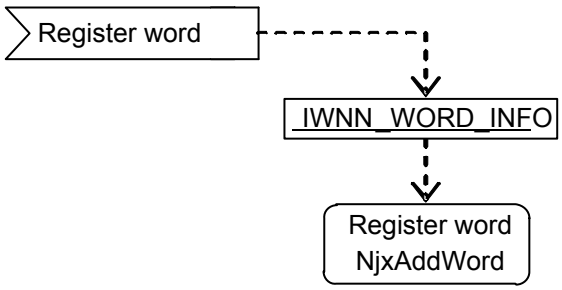


7.7 Registering Words to the User Dictionary/Learning Dictionary

Operations for registering words to the user dictionary/learning dictionary are performed according to the procedure given below.

1. Register word function (**`NjxAddWord`**).
   With a user dictionary/learning dictionary handle set in the dictionary set structure (`IWNN_DIC_SET`), words are registered by specifying a reading string, candidate string, part of speech group, and connection flag in the word registration information structure (`IWNN_WORD_INFO`). Be sure to specify an additional information string as well, if the user dictionary/learning dictionary includes additional information.
   Specify a user dictionary or learning dictionary for the type of dictionary to be registered to. If a user dictionary and learning dictionary have both been specified in the dictionary set, words registered to the user dictionary will be automatically registered to the learning dictionary, as well.

Figure 7-7 Registering Words to the User Dictionary/Learning Dictionary



7.8 Deleting Words from the User Dictionary/Learning Dictionary

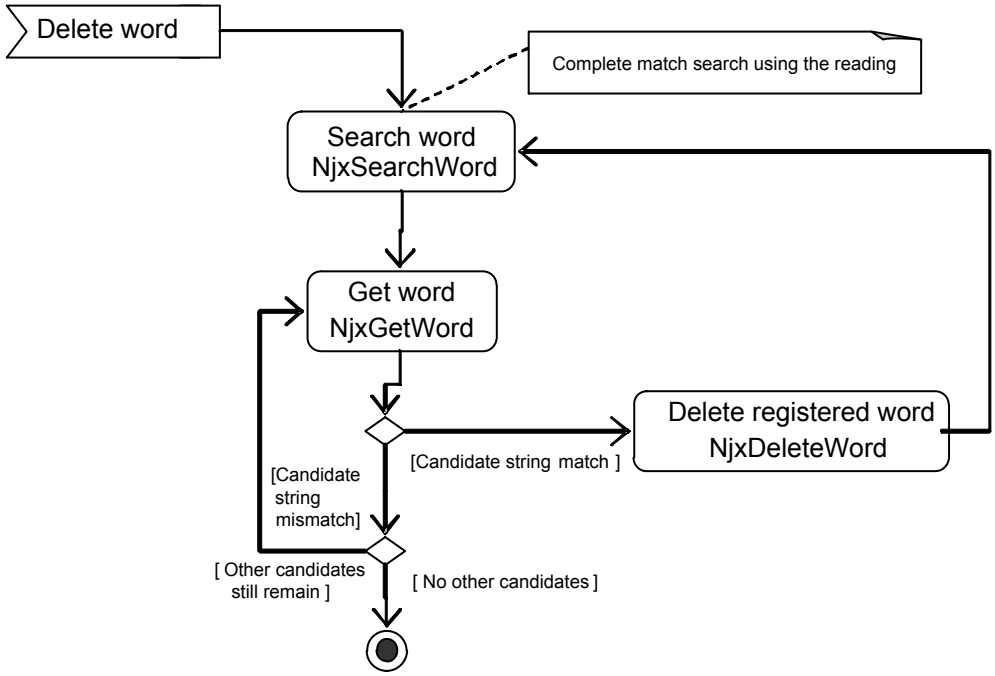
Operations for deleting words from the user dictionary and learning dictionary are performed as described below.

Duplicates of the same word may be registered in the learning dictionary. It is, therefore, necessary to repeatedly search and delete until all instances of the word to be deleted are gone.

If deleting all registered words, this can be achieved faster by using the create dictionary region function to initialize the dictionary.

1. Search word function (**`NjxSearchWord`**).
   A complete match search is made of the dictionary, using the reading string of the word to be deleted.
2. Get word function (**`NjxGetWord`**).
   Get the candidate by specifying the dictionary search cursor structure (`IWNN_CURSOR`) obtained by the search word function. The candidate string obtained using the get candidate string function (**`NjxGetCandidate`**) is checked to see if the word should be deleted.
3. Delete word function (**`NjxDeleteWord`**).
   Get the word by specifying the processing result structure (`IWNN_RESULT`), obtained by the get word function, and delete the word.

Figure 7-8 Deleting Words from the User Dictionary/Learning Dictionary

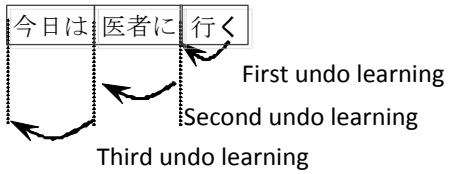


7.9 Undo Learning

Operations for restoring the learning dictionary to the state before learning, when candidate confirmation has been undone, are performed according to the procedure given below.

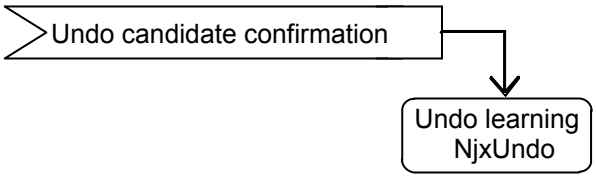
To restore the learning dictionary to the state immediately before a learning operation, you must specify the number of times the learn function (**`NjxSelect`**) has been called to undo the learning function.

For example, when the three phrases “今日は”, “医者に”, “行く” have been learned:



1. Undo learning function (**`NjxUndo`**).
   Set the number of undo’s and restore the learning dictionary to the state before learning.

Figure 7-9 Undo Learning

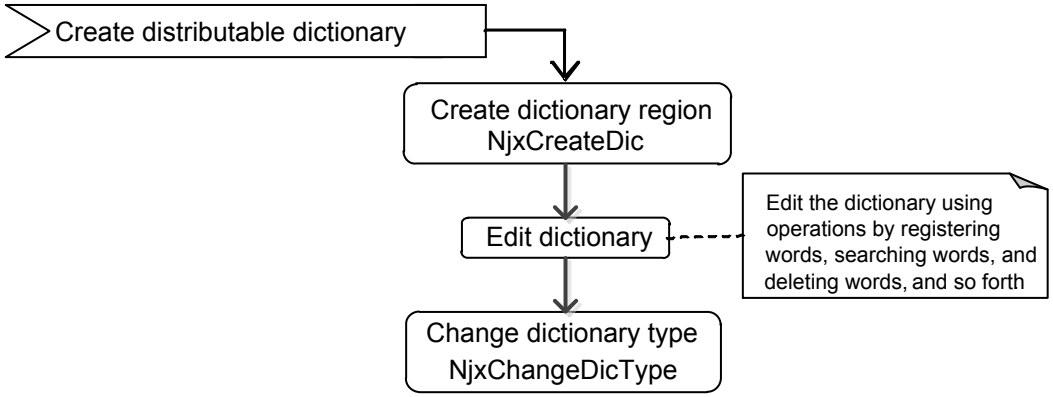


7.10 Creating a Distributable Dictionary on a Terminal

To create a distributable dictionary on a terminal, first create a learning dictionary and register words to it, and then perform operations to convert the learning dictionary to a distributable dictionary format.

1. Create dictionary region function (**`NjxCreateDic`**).
   This allocates memory for the distributable dictionary and initializes it as a learning dictionary.
2. Edit dictionary.
   Edit the contents of the created learning dictionary, using the register word function (NjxAddWord) and delete word function (**`NjxDeleteWord`**).
3. Change dictionary type function (**`NjxChangeDicType`**).
   Change the dictionary type to an uncompressed customized dictionary (distributable dictionary). A distributable dictionary created using this function can be restored to a learning dictionary format.

Figure 7-10 Creating a Distributable Dictionary on a Terminal



7.11 Morphological Analysis (Delimited Input)

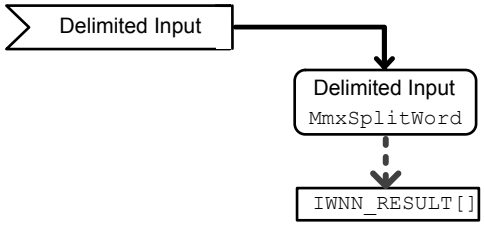
To split a sentence into phrases (delimited input), follow the procedure given below.

For example, when “今日は医者に行く” is converted to delimited input, the result is:

“今日は” “医者に” “行く”.

1. Delimited input function (**`MmxSplitWord`**).
   With the dictionary to be used for morphological analysis registered in `IWNN_DIC_SET`, specify the sentence to be parsed and call the delimited input function (**`MmxSplitWord`**).
   The results of delimiting are stored in an array of processing result structures (`IWNN_RESULT`).

Figure 7-11 Morphological Analysis (Delimited Input)



Depending on the phrase, there may be more than one possible reading. (For example, 今日 may become きょう or こんにち.)

Therefore, use the get reading string by using the morphological analysis function (`MmxGetReading`) to get readings from the processing result (`IWNN_RESULT`), obtained by the delimited input function. Note that the get reading string function (**`NjxGetStroke`**) cannot be used.

Use the get candidate string function (**`NjxGetCandidate`**) to get the notational string for the phrase.

Use the get additional information function (**`NjxGetAdditonalInfo`**) to get the additional information string for the phrase.

7.12 Automatic Learning When Replying to E-mail (Learning Morphological Analysis Results)

Note: For more information on using this feature, refer to section D.6 Automatic Learning When Replying to E-mail.

When replying to e-mail, you can predict and convert words while prioritizing words included in the received e-mail, by learning through morphological analysis of the text in the received e-mail (automatic learning when replying to e-mail). Because the probability of using words included in the received e-mail is relatively high when replying to an e-mail, usefulness when creating the reply is improved.

Automatic learning when replying is performed according to the procedure given below.

1. Delimited input function (**`MmxSplitWord`**).
   The text of the received e-mail being replied to undergoes morphological analysis.
   A dictionary set (`IWNN_DIC_SET`) is prepared ahead of time, to include a dictionary for morphological analysis and a learning dictionary for automatic learning when replying. The text of the received e-mail is then parsed using this dictionary set.
   Because there is a limit on the string length that can be parsed at one time, the entire text is split multiple times and parsed.

2. Select morphological analysis candidates.
   Candidates to be used for learning are selected as necessary from the processing result (`IWNN_RESULT`), obtained by the delimited input function (**`MmxSplitWord`**).
   You can do things like: learning only a particular part of speech (for example, learning only uninflected words) by using the get part of speech group function (**`MmxGetPartsOfSpeech`**), or learning only a particular character type (for example, learning only katakana words) by using the get character type function (**`NjxGetCharType`**).

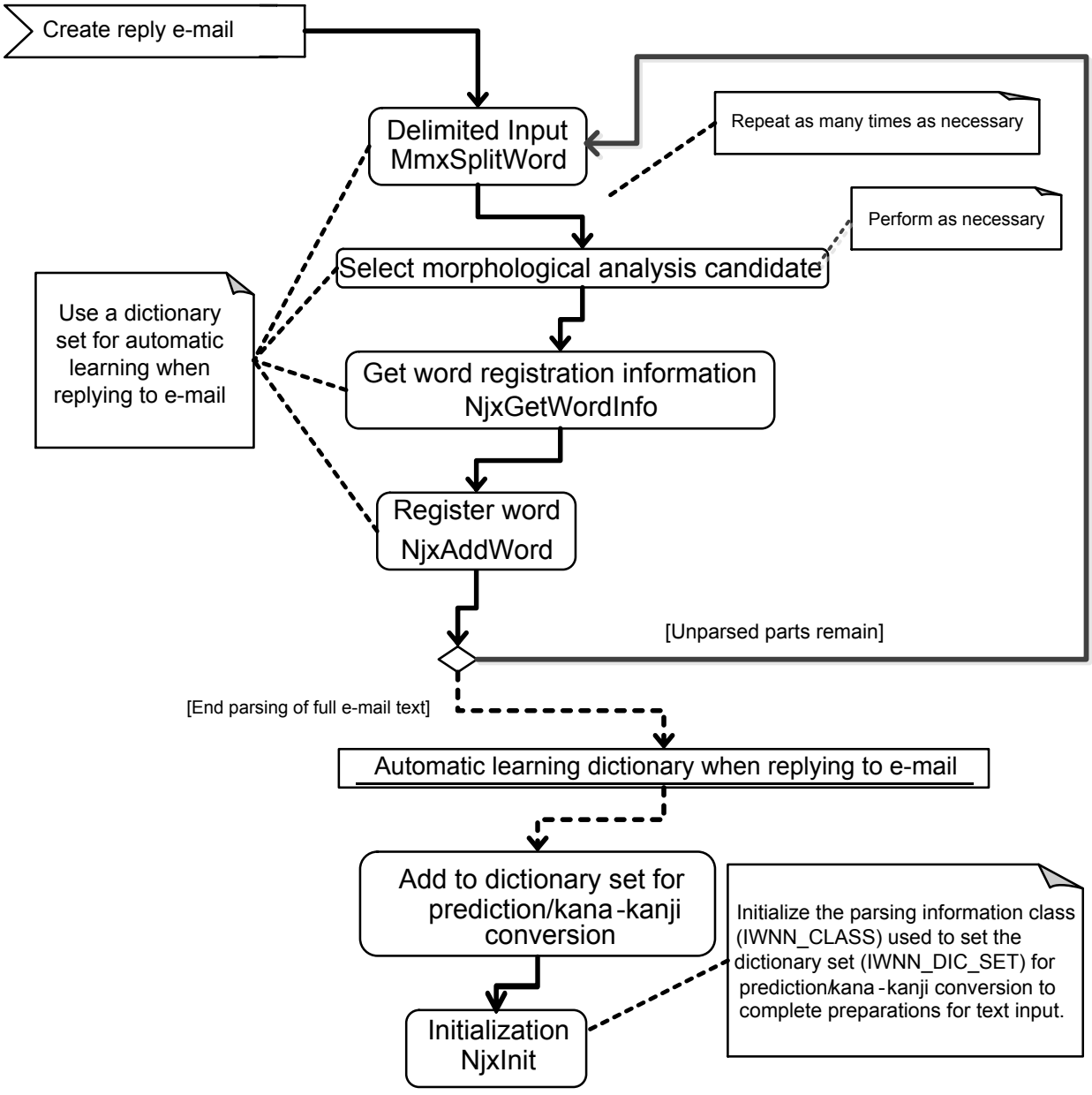
3. Get word registration information function (**`NjxGetWordInfo`**).
   Word information to be learned (`IWNN_WORD_INFO`) is obtained from the morphological analysis processing result (`IWNN_RESULT`). Depending on the word, there may be more than one reading, but the first candidate will be used.

4. Register word function (**`NjxAddWord`**).
   Word registration information (`IWNN_WORD_INFO`), obtained using the dictionary set for automatically learning when replying, is registered in the automatic learning dictionary when replying.

5. Set a dictionary set for prediction/kana-kanji conversion.
   The dictionary for automatic learning when creating a reply is set in the dictionary set used for prediction/kana-kanji conversion.
   Using the learn function (**`NjxSelect`**), confirmed candidates are registered in a dictionary having a learning dictionary format configured beforehand, in the order mounted in the dictionary set. For this reason, be sure to set the dictionary for automatic learning when replying after using normal learning dictionaries.

6. Initialization function (**`NjxInit`**).
   Along with changes to the content of the dictionary set for prediction/kana-kanji conversion, the parsing information class (`IWNN_CLASS`) is initialized.

Figure 7-12 Automatic Learning When Replying to E-mail (Learning Morphological Analysis Results)



To create an automatic learning dictionary when replying, you can also use the get morphological analysis reading string function (**`MmxGetReading`**) and learn using the morphological analysis function (`MmxSelect`), instead of using the get word registration information function and register word function.

8 Detailed Description of Structures

8.1 Parsing Information Class (IWNN_CLASS)

The parsing information class structure stores variables used by iWnn in the conversion and parsing processes.

The dictionary to be used can be changed by configuring a dictionary set to be included in this structure.

For information on configuring a dictionary set, refer to section 8.2 Dictionary Sets (IWNN_DIC_SET, IWNN_DIC_INFO, and IWNN_FLASH_DIC_INFO).

Memory other than that for the dictionary set is initialized using the initialization function (**NjxInit**).

Code 8-1 Parsing Class (IWNN_CLASS) Structure Configuration

```
struct IWNN_CLASS {
 ...
 IWNN_DIC_SET  dicSet;   // Dictionary set
 ...
};
```

8.2 Dictionary Sets (IWNN_DIC_SET, IWNN_DIC_INFO, and IWNN_FLASH_DIC_INFO)

The dictionary set structure specifies the set of dictionaries to be subjected to each process when dictionary lookups, conversions, predictions, or morphological analyses using iWnn are requested. Settings for each dictionary are defined in the IWNN_DIC_INFO structure. The set of dictionaries to be used is defined in IWNN_DIC_SET.

Code 8-2 Dictionary Set (IWNN_DIC_SET, IWNN_DIC_INFO, and IWNN_FLASH_DIC_INFO) Structure Configuration

```
struct IWNN_FLASH_DIC_INFO {
 u32 dicType;                              // Internal use
 u32 dicSize;                              // Internal use
 u32 mode;                           // Internal use
 IWNN_FILE* fileStream;              // Internal use
 void* cacheArea[NJ_FLASH_DIC_CACHE_MAX];  // Internal use
 void* extensionData;                // Internal use
};

struct IWNN_DIC_INFO {
 u8 type;                            // Dictionary type
 u8 limit;                           // Search limit
```

```
 IWNN_DIC_HANDLE handle;             // Dictionary handle
 void* extensionArea;                      // Pseudo-dictionary work area
 void* addInfo[NJ_MAX_ADDITIONAL_INFO];    // Additional information region
 struct {
        u16 base;        // Base frequency by dictionary handle
        u16 high;        // Maximum frequency by dictionary handle
 } dicFrequency[NJ_MODE_TYPE_MAX];
 NJ_SEARCH_CACHE* searchCache;       // Cache management region
};

struct IWNN_DIC_SET {
 IWNN_DIC_INFO dic[NJ_MAX_DIC];            // Dictionary information
 IWNN_DIC_HANDLE ruleHandle[NJ_MODE_TYPE_MAX];
                                           // Rule dictionary handle
 u16 mode;                                 // Internal use
 wchar_t keyword[NJ_MAX_KEYWORD];          // Internal use
};
```

The `IWNN_DIC_INFO` structure is used to set a pointer (handle) to the dictionary itself and the frequency range (`dicFrequency`). A separate frequency range is set for conversion (`NJ_MODE_TYPE_CONVERSION`), prediction (`NJ_MODE_TYPE_PREDICTION`), and morphological analysis (`NJ_MODE_TYPE_MORPHOLIZE`). A pointer (`searchCache`) to the cache region must be specified for dictionaries subjected to fuzzy searches.

However, the rule dictionary is an exception. Directly set a pointer to the dictionary using `IWNN_DIC_SET.ruleHandle[]`. Set a rule dictionary for each conversion, prediction, and morphological analysis operation.

For the FLASH dictionary (target: integrated dictionary), instead of setting a pointer (handle) to the dictionary structure, set all required information in the `IWNN_FLASH_DIC_INFO` structure, and then set a start pointer to the `IWNN_FLASH_DIC_INFO` structure.

Because functions for prediction, morphological analysis, conversion, and dictionary lookup search dictionaries in the order registered in `IWNN_DIC_SET`, if homonym candidates having the same frequency occur in more than one dictionary, the order of registration in the structure will become the order of candidates.

Table 8-1 `IWNN_DIC_SET` Structure Members

Member	Description
`IWNN_DIC_INFO dic`	Dictionary information other than for a rule dictionary. Set `dic[].handle = NULL` for unused areas.
`IWNN_DIC_HANDLE ruleHandle`	Dictionary handle for the rule dictionary. **This must be specified.**

Table 8-2 `IWNN_DIC_INFO` Structure Members

Member	Description
`u8 type`	Dictionary type. Used to distinguish between normal dictionaries and pseudo-dictionaries and identify the type of pseudo-dictionaries. `IWNN_DIC_HANDLE_TYPE_NRM`: Normal dictionary. `IWNN_DIC_HANDLE_TYPE_PROGRAM`: Pseudo-dictionary. `IWNN_DIC_HANDLE_TYPE_PROGRAM_ANCILLARY`: Pseudo-dictionary (ancillary words). `IWNN_DIC_HANDLE_TYPE_ON_FLASH`: FLASH dictionary.
`u8 limit`	Search limit by dictionary handle. Sets the length of the input string to start prediction search. Specified as a number of characters under typical usage. Setting range: From 0 to `NJ_MAX_LEN`+1 (0 to infinity). Set to `NJ_MAX_LEN`+1 when not being used.
`IWNN_DIC_HANDLE handle`	Dictionary handle. Set to NULL for unused regions.
`void* extensionArea`	Used as the work area for the pseudo-dictionary. Allocate memory of the size required by each pseudo-dictionary.
`void* addInfo[]`	Additional information region. Sets additional information in the dictionary. This member can be used to set additional information for integrated dictionaries and no reading prediction dictionaries. Specify NULL for all elements in the case of other dictionaries. Additional information can be set only for the number of array elements. Set NULL in elements that do not store additional information. Only when learnable additional information has been set in the 0th element is that additional information the target of learning (capable of being learned). (However, if the additional information is longer than `NJ_MAX_ADDITIONAL_LEN`, it is not the target of learning.) Additional information set in an element other than the 0th and additional information that cannot be learned is not the target of learning (cannot be learned) by the learning dictionary.
`u16 dicFrequency.base`	Base frequency by dictionary handle. This value is added to the frequency value by word in the dictionary, and used as the default frequency for words. Setting range: 0 to 1000.

Member	Description
u16 dicFrequency.high	Maximum frequency by dictionary handle. This value is used as the default frequency if the default frequency during dictionary lookup is greater than or equal to this value. Setting range: 0 to 1000. Set a value equal to or greater than the base frequency using dictionary handle.
NJ_SEARCH_CACHE* searchCache	Cache management region by dictionary handle. Allocate and configure memory for cache management (NJ_SEARCH_CACHE) for dictionaries that will undergo fuzzy prediction searches. Set to NULL for unused regions. The following dictionaries can undergo fuzzy prediction search. • Integrated dictionary • User dictionary • Customized dictionary • Learning dictionary

IWNN_FLASH_DIC_INFO Structure Members

Members of this structure are all set using the API.

The cache size required by the FLASH dictionary can be obtained using the function for getting the FLASH dictionary cache size (**NjxGetFlashDicCacheSize**). Allocate the memory size required. After allocation, use the function for setting FLASH dictionary information (**NjxSetFlashDicInfo**) to set the information needed for the FLASH dictionary to this structure.

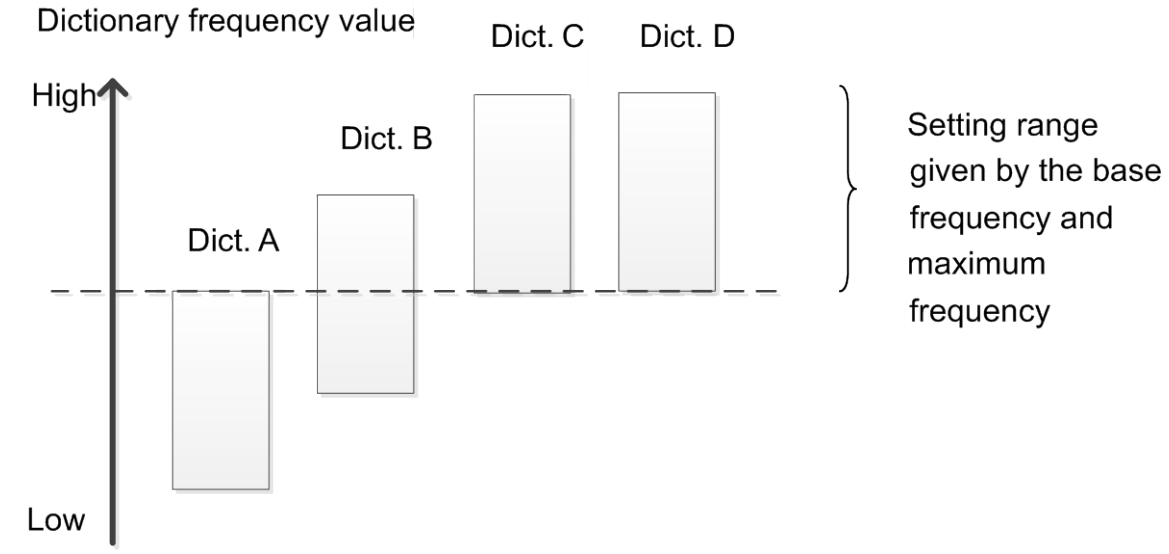
8.3 Dictionary Frequency Value Settings

These settings allow you to specify the priority relationship among dictionaries according to a dictionary frequency value.

Dictionary frequency values are set as a pair of values representing base frequency and high frequency. iWnn handles the priority of stored words within a specified frequency range.

In addition, dictionary frequency values can be set separately for normal conversion (kana-kanji conversion and get all candidates), prediction conversion, and morphological analysis.

Figure 8-1 Dictionary Frequency Setting Example



In the example shown in Figure 8-1, words stored in Dictionary C are always prioritized above words stored in Dictionary A. In addition, by overlapping setting ranges as shown for Dictionary A and Dictionary B, you can prioritize only those words stored in Dictionary B that have a higher priority than words in Dictionary A. If the setting ranges for two dictionaries are the same, as shown for Dictionary C and Dictionary D, priority is determined based solely on the frequency value information stored for each word.

However, depending on the state setting during connection prediction and situational adaptive prediction, the priority of a word may be changed because it lies outside the setting range of its dictionary.

8.3.1 Limitations on Base Frequency/Maximum Frequency by Dictionary Handle

Specify the base frequency and maximum frequency by dictionary handle in the range from 0 to 1000.

If the base frequency is greater than the maximum frequency, that dictionary will not be used.

Using a word search function with this setting results in an error.

8.3.2 Recommended Values When Using Multiple Customized Dictionaries

Table 8-3 Recommended Values When Using Multiple Customized Dictionaries

Dictionary Type	Fuzzy Search	For Conversion		For Prediction		For Morphological Analysis	
		Base	Maximum	Base	Maximum	Base	Maximum
Learning Dictionary	Yes	501	1000	501	1000	10	0
Integrated Dictionary	Yes	400	550	100	560	400	500
Customized Dictionaries (multiple)	Yes	0 to 400	0 to 400	0 to 400	0 to 400	10	0
User Dictionary	No	410	410	10	0	10	0
Single Kanji Dictionary	No	0	10	10	0	0	10
Ancillary Word Dictionary	No	400	500	10	0	400	500
No Reading Prediction Dictionary	No	10	0	100	244	10	0

Because there are so many words in the integrated dictionary, many words will be assigned to each possible frequency. A broad frequency range from 0 to 400 is assigned to the customized dictionary so that more words appear in the candidate list than other customized dictionaries.

For dictionaries not used in any particular mode, set a base frequency of 10 and a maximum frequency of 0 (base frequency greater than the maximum frequency) to exclude them from use.

8.3.3 Recommended Values When Using a No Reading Prediction Dictionary (Start of Text)

Table 8-4 Recommended Values When Using a No Reading Prediction Dictionary (Start of Text)

Dictionary Type	Fuzzy Search	For Conversion		For Prediction		For Morphological Analysis	
		Base	Maximum	Base	Maximum	Base	Maximum
Learning Dictionary	Yes	501	1000	501	1000	10	0
Integrated Dictionary	Yes	400	550	100	560	400	500
Customized Dictionaries (multiple)	Yes	0 to 400	0 to 400	0 to 400	0 to 400	10	0
User Dictionary	No	410	410	10	0	10	0
Single Kanji Dictionary	No	0	10	10	0	0	10
Ancillary Word Dictionary	No	400	500	10	0	400	500
No Reading Prediction Dictionary	No	10	0	100	244	10	0
No Reading Prediction Dictionary <start of text >	No	10	0	100	900	10	0

8.3.4 Recommended Values When Using Learning Results Based on Morphological Analysis

Table 8-5 When Prioritizing Morphological Analysis Results

Dictionary Type	Fuzzy Search	For Conversion		For Prediction		For Morphological Analysis	
		Base	Maximum	Base	Maximum	Base	Maximum
Morphological Learning Dictionary	Yes	900	1000	900	1000	10	0
Learning Dictionary	Yes	501	1000	501	1000	10	0
Integrated Dictionary	Yes	400	550	100	560	400	500
User Dictionary	No	410	410	10	0	10	0
Single Kanji Dictionary	No	0	10	10	0	0	10
Ancillary Word Dictionary	No	400	500	10	0	400	500
No Reading Prediction Dictionary	No	10	0	100	244	10	0

Table 8-6 When Prioritizing the Learning Dictionary

Dictionary Type	Fuzzy Search	For Conversion		For Prediction		For Morphological Analysis	
		Base	Maximum	Base	Maximum	Base	Maximum
Learning Dictionary	Yes	501	1000	501	1000	10	0
Morphological Learning Dictionary	Yes	700	800	700	800	10	0
Integrated Dictionary	Yes	400	550	100	560	400	500
User Dictionary	No	410	410	10	0	10	0
Single Kanji Dictionary	No	0	10	10	0	0	10
Ancillary Word Dictionary	No	400	500	10	0	400	500
No Reading Prediction Dictionary	No	10	0	100	244	10	0

If we assume morphological analysis of about 100 words parsed for learning information of approximately 1000 words in a 32KB learning dictionary, a discrepancy will arise in the number of words per frequency. Therefore, we specify a wider range of frequencies, as illustrated above, to average out priorities and the number of words obtained.

8.3.5 Definition of the Standard Dictionary Frequency Value

The following macros have been defined based on recommended values for the standard dictionary frequency value.

Table 8-7 Standard Dictionary Frequency Value Definitions

Dictionary	Use		Macro
Integrated Dictionary	Conversion	Maximum	NJ_DEFAULT_FREQUENCY_DIC_UNION_CONVERSION_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_UNION_CONVERSION_BASE
	Prediction	Maximum	NJ_DEFAULT_FREQUENCY_DIC_UNION_PREDICTION_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_UNION_PREDICTION_BASE
	Morphological Analysis	Maximum	NJ_DEFAULT_FREQUENCY_DIC_UNION_MORPHOLIZE_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_UNION_MORPHOLIZE_BASE
Single Kanji Dictionary	Conversion	Maximum	NJ_DEFAULT_FREQUENCY_DIC_SINGLE_CONVERSION_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_SINGLE_CONVERSION_BASE
	Prediction	Maximum	NJ_DEFAULT_FREQUENCY_DIC_SINGLE_PREDICTION_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_SINGLE_PREDICTION_BASE
	Morphological Analysis	Maximum	NJ_DEFAULT_FREQUENCY_DIC_SINGLE_MORPHOLIZE_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_SINGLE_MORPHOLIZE_BASE
Ancillary Word Dictionary	Conversion	Maximum	NJ_DEFAULT_FREQUENCY_DIC_ANCILLARY_CONVERSION_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_ANCILLARY_CONVERSION_BASE
	Prediction	Maximum	NJ_DEFAULT_FREQUENCY_DIC_ANCILLARY_PREDICTION_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_ANCILLARY_PREDICTION_BASE
	Morphological Analysis	Maximum	NJ_DEFAULT_FREQUENCY_DIC_ANCILLARY_MORPHOLIZE_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_ANCILLARY_MORPHOLIZE_BASE
No Reading Prediction Dictionary	Conversion	Maximum	NJ_DEFAULT_FREQUENCY_DIC_READING_CONVERSION_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_READING_CONVERSION_BASE
	Prediction	Maximum	NJ_DEFAULT_FREQUENCY_DIC_READING_PREDICTION_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_READING_PREDICTION_BASE
	Morphological Analysis	Maximum	NJ_DEFAULT_FREQUENCY_DIC_READING_MORPHOLIZE_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_READING_MORPHOLIZE_BASE
No Reading Prediction Dictionary (start of text)	Conversion	Maximum	NJ_DEFAULT_FREQUENCY_DIC_HEAD_READING_CONVERSION_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_HEAD_READING_CONVERSION_BASE
	Prediction	Maximum	NJ_DEFAULT_FREQUENCY_DIC_HEAD_READING_PREDICTION_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_HEAD_READING_PREDICTION_BASE
	Morphological	Maximum	NJ_DEFAULT_FREQUENCY_DIC_HEAD_READING_MORPHOLIZE_HIGH

Dictionary	Use		Macro
	Analysis	Base	NJ_DEFAULT_FREQUENCY_DIC_HEAD_READING_MORPHOLIZE_BASE
User Dictionary	Conversion	Maximum	NJ_DEFAULT_FREQUENCY_DIC_USER_CONVERSION_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_USER_CONVERSION_BASE
	Prediction	Maximum	NJ_DEFAULT_FREQUENCY_DIC_USER_PREDICTION_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_USER_PREDICTION_BASE
	Morphological Analysis	Maximum	NJ_DEFAULT_FREQUENCY_DIC_USER_MORPHOLIZE_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_USER_MORPHOLIZE_BASE
Learning Dictionary	Conversion	Maximum	NJ_DEFAULT_FREQUENCY_DIC_LEARN_CONVERSION_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_LEARN_CONVERSION_BASE
	Prediction	Maximum	NJ_DEFAULT_FREQUENCY_DIC_LEARN_PREDICTION_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_LEARN_PREDICTION_BASE
	Morphological Analysis	Maximum	NJ_DEFAULT_FREQUENCY_DIC_LEARN_MORPHOLIZE_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_LEARN_MORPHOLIZE_BASE
Customized Dictionary	Conversion	Maximum	NJ_DEFAULT_FREQUENCY_DIC_CUSTOM_CONVERSION_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_CUSTOM_CONVERSION_BASE
	Prediction	Maximum	NJ_DEFAULT_FREQUENCY_DIC_CUSTOM_PREDICTION_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_CUSTOM_PREDICTION_BASE
	Morphological Analysis	Maximum	NJ_DEFAULT_FREQUENCY_DIC_CUSTOM_MORPHOLIZE_HIGH
		Base	NJ_DEFAULT_FREQUENCY_DIC_CUSTOM_MORPHOLIZE_BASE

8.4 Processing Results (IWNN_RESULT)

The processing result structure stores processing results when dictionary lookups, conversions, or predictions using iWnn have been requested. Do not directly access or modify information stored in this structure.

Reading strings resulting from processing are obtained using the **NjxGetStroke** function, candidate strings resulting from processing are obtained using the **NjxGetCandidate** function, and the dictionary to retrieve them from is obtained using the **NjxGetDicHandle** function.

Code 8-3 Processing Result (IWNN_RESULT) Structure Configuration

```
struct IWNN_RESULT {
 u16 operationId;         // Operation information
 struct IWNN_WORD word;   // Member used for internal processing
};
```

Table 8-8 Structure Members

Member	Description
u16 Operation ID	Operation information. Every 4 bits, stores information for operation, feature, word information, and dictionary type.
struct IWNN_WORD Word	Variable used for internal processing. This variable cannot be accessed from outside the API.

Table 8-9 Operation Information

Category	Type	Description
Operation	NJ_OPERATION_SEARCH	Dictionary Lookup Processing Result.
	NJ_OPERATION_CONVERT	Conversion processing result.
	NJ_OPERATION_ANALYZE	Evaluation (prediction) processing result.
	NJ_OPERATION_MORPHOLIZE	Morphological analysis processing result.
	NJ_OPERATION_ENVIRONMENT	Common processing result.
Feature	NJ_FUNC_SEARCH NJ_FUNC_RELATION	Dictionary lookup features.
	NJ_FUNC_SEARCH_REVERSE	Dictionary lookup feature (reverse).
	NJ_FUNC_CONVERT_MULTIPLE	Multiple phrase (multiple phrase) conversion feature.
	NJ_FUNC_CONVERT_SINGLE	Multiple phrase (single phrase) conversion feature.
	NJ_FUNC_ALL_CANDIDATES	Single phrase conversion (all candidates) feature.
	NJ_FUNC_NEXT	No reading prediction feature.
	NJ_FUNC_SPLIT_WORD	Delimited input feature.
Word Information	The high order 1 bit (NJ_TYPE_PSEUDO_BIT) is the pseudo-candidate bit. If this bit is set, it indicates a pseudo-candidate.	
Dictionary Type	NJ_DIC_PSEUDO	Generated by iWnn system (no target dictionary).
	NJ_DIC_CUSTOMIZE	Customized dictionary.
	NJ_DIC_LEARN	Learning dictionary.
	NJ_DIC_USER	User dictionary.
	NJ_DIC_STATIC	Standard dictionary other than the above.
	NJ_DIC_PROGRAM	Pseudo-dictionary.

8.5 Dictionary Search Cursor (IWNN_CURSOR)

When searching for words, the dictionary set, search string, internal dictionary search position (IWNN_SEARCH_LOCATION_SET), and search conditions (IWNN_SEARCH_CONDITION) are set in the dictionary search cursor structure, and a word search function is used to execute a search.

Word location information resulting from search is stored in the search cursor structure for the same dictionary. However, internal information and the IWNN_SEARCH_LOCATION_SET structure must not be directly accessed or changed by the application.

Code 8-4 Dictionary Search Cursor (IWNN_CURSOR) Structure Configuration

```
struct IWNN_CURSOR {
 struct IWNN_SEARCH_CONDITION condition;
 struct IWNN_SEARCH_LOCATION_SET locationSet[NJ_MAX_DIC];
};
```

Code 8-5 Search Condition (IWNN_SEARCH_CONDITION) Structure Configuration

```
struct IWNN_SEARCH_CONDITION {
 u8 operation;                 // Search method
 u8 mode;                      // Search candidate order
 struct IWNN_DIC_SET* dicSet;  // Dictionary set
 struct { …} partsOfSpeech;          // Member for internal processing
 wchar_t* reading;             // Search reading string
 u16 readingLen;               // Member for internal processing
 wchar_t* kanji;               // Pre-confirmation string during
                                       prediction candidate search
 IWNN_CHARSET* charSet;        // Fuzzy character set
 ...
};
```

Table 8-10 Structure Members

Member	Description
u8 condition.operation	Search method: Forward lookup complete match search (0) Forward lookup prefix match search (1) Derived search (2) Reverse lookup complete match search (3) Reverse lookup prefix match search (4)
u8 condition.mode	Search candidate order: Frequency order (0) Reading order (1) Registration order (2) Reading order search can be used only when forward lookup prefix match search is specified. Registration order searches can only be used with a learning dictionary or user dictionary. As such, only an empty string can be specified as the reading string for words to be searched.
IWNN_DIC_SET condition.dicSet	Dictionary set to be searched.
wchar_t* condition.reading	Reading string of the word to be searched for. Add a terminator at the end of the string.
wchar_t* condition.kanji	Pre-confirmation string during derived search. Add a terminator at the end of the string.
IWNN_CHARSET* condition.charSet	Fuzzy character set to use in fuzzy search. Set to NULL if not using fuzzy searches. Fuzzy searches can be used only with forward lookup prefix match search and reverse lookup prefix match search.

8.5.1 Search Methods and Search Candidate Order by Dictionary

The following limitations are placed on the search method and search candidate order specification, depending on the dictionary structure.

Table 8-11 Search Method and Search Candidate Order by Dictionary

Dictionary Type	Dictionary
Dictionaries for forward lookup complete match search	Compressed customized dictionary Ancillary word dictionary Single kanji dictionary
Dictionaries for forward lookup prefix match searches	Compressed customized dictionary
Dictionaries for reverse lookup complete match searches	Compressed customized dictionary Single kanji dictionary (with morphological analysis support) User dictionary
Integrated dictionary	Integrated dictionary
Dictionary for derived searches	Uncompressed customized dictionary Learning dictionary
Other	No reading prediction dictionary Rule dictionary

Dictionaries that can be used for fuzzy searches are dictionaries for forward lookup prefix match search, dictionaries for reverse lookup prefix match searches, dictionaries for derived searches, and the integrated dictionary.

8.5.2 Dictionary Type and Search Method

Table 8-12 Dictionary Type and Search Method

Search Method / Dictionary Type	Forward Lookup Complete Match Search	Forward Lookup Prefix Match Search	Reverse Lookup Complete Match Search	Reverse Lookup Prefix Match Search	Derived Search
Dictionary for forward lookup complete match search	Yes	No	No	No	No
Dictionary for forward lookup prefix match search	Yes	Yes	No	No	No
Dictionary for reverse lookup complete match search	Yes	Yes	Yes	No	No
Integrated dictionary	Yes	Yes	Yes	Yes	Yes
Dictionary for derived searches	Yes	Yes	Yes	No	Yes

8.5.3 Search Method and Search Candidate Order

Table 8-13 Search Method and Search Candidate Order

Search Candidate Order / Search Method	Frequency Order	Reading Order	Registration Order
FFFForward lookup complete match search	Yes	No	No
Forward lookup prefix match search	Yes	Yes	Yes
Reverse lookup complete match searches	Yes	No	No
Reverse lookup prefix match search	Yes	No	No
Derived search	Yes	No	No

- No reading prediction dictionaries and rule dictionaries cannot be searched.
- Obtained candidates with the same reading are re-arranged in candidate order only for user dictionaries. Candidates with the same reading are arranged in frequency order in the case of other dictionaries.
- Searches can be made in registration order only for learning dictionaries and user dictionaries.

8.6 Word Registration Information (`IWNN_WORD_INFO`)

The word registration information structure specifies parameters when adding words to the user dictionary or learning dictionary.

Code 8-6 Word Registration Information (`IWNN_WORD_INFO`) Structure Configuration

```
struct IWNN_WORD_INFO {
 u8 partsOfSpeechGroup;                          // Part of speech
                                                                group
 wchar_t reading[NJ_MAX_LEN+NJ_TERM_SIZE];              // Reading
 wchar_t candidate[NJ_MAX_RESULT_LEN+NJ_TERM_SIZE];     // Candidate
 wchar_t additional[NJ_MAX_ADDITIONAL_LEN+NJ_TERM_SIZE];     // Additional
                                                               information
 struct {                                       // <Independent
                                                             word information >
       u16 readingLen;                          // Reading string length
       u16 candidateLen;                              // Candidate string
                                                           length
       u32 partsOfSpeech;                             // Part of speech
       u32 attr;                                // Attribute data
```

```
       u16 frequency;                    // Frequency
 } stem;
struct {                                       // <Anciallary word
                                                    information>
       u16 readingLen;                   // Reading string length
       u16 candidateLen;                 // Candidate string
                                                    length
       u32 partsOfSpeech;                       // Part of speech
       s16 frequency;                    // Frequency
 } ancillary;
 s16 connect;                                  // Connection flag
};
```

Table 8-14 Word Registration Information Structure Members

Member	Description
u8 partsOfSpeechGroup	Part of speech group number for word to be added. NJ_PARTS_OF_SPEECH_NOUN: Common noun/proper noun (conjugates with "suru"). NJ_PARTS_OF_SPEECH_NOUN_NO_CONJUGATION: Common noun/proper noun (does not conjugate with "suru"). NJ_PARTS_OF_SPEECH_PERSON_NAME: Person's name. NJ_PARTS_OF_SPEECH_PLACE_NAME: Place name/station name. NJ_PARTS_OF_SPEECH_SYMBOL: Symbol. NJ_PARTS_OF_SPEECH_DETAIL: Get details. If get details (NJ_PARTS_OF_SPEECH_DETAIL) is specified, the word is registered using part of speech information for independent word information (stem) and ancillary word information (ancillary). If you are not using the registerword information function (**NjxGetWordInfo**), set a value other than NJ_PARTS_OF_SPEECH_DETAIL.
wchar_t reading	Reading string for the word to be added. Add a terminator at the end of the string.
wchar_t candidate	Candidate (conversion result) string for the word to be registered. Add a terminator at the end of the string.
wchar_t additional	Additional information string for the word to be registered. Add a terminator at the end of the string. If the dictionary being registered into does not include additional information, the content of this member is ignored.
struct stem	Independent word information. This includes the reading string length of the independent word part, the candidate string length, the part of speech, attribute data, and frequency information. This information is set using the get word information function (NjxGetWordInfo) from the processing result structure (IWNN_RESULT).

Member	Description
struct ancillary	Ancillary word information. This includes the reading string length of the ancillary word part, the candidate string length, the part of speech, attribute data, and frequency information. This information is set using the get word information function (**NjxGetWordInfo**) from the processing result structure (IWNN_RESULT).
s16 connect	Connection flag. Specifies whether to perform associative learning with the word added immediately before. 0: Do not perform associative learning. 1: Perform associative learning. This setting is enabled only for the learning dictionary.

8.7 Fuzzy Character Set (IWNN_CHARSET)

The fuzzy character set structure specifies fuzzy character patterns to use during fuzzy searches.

Code 8-7 Fuzzy Character Set (IWNN_CHARSET) Structure Configuration

```
struct IWNN_CHARSET {
 u16 charSetCnt;                    // Number of registrations
 wchar_t* from[NJ_MAX_CHARSET];           // Pre-replacement character
 wchar_t* to[NJ_MAX_CHARSET];       // Post-replacement character
};
```

Table 8-15 Fuzzy Character Set Structure Members

Member	Description
u16 charSetCnt	Number of registrations. Specify the number of configured fuzzy string patterns.
wchar_t* from	Pre-replacement character. Specify one character for the pre-replacement character. Add a terminator at the end of the string.
wchar_t* to	Post-replacement character. Specify 1 to 3 characters for the post-replacement character. Add a terminator at the end of the string.

For example, make the following definitions to show the candidates “ば” and “ば゜”, when the character “は” has been entered for a fuzzy search.

Code 8-8 Fuzzy Search Character Set (IWNN_CHARSET) Structure Definition Example

```
const u8 HA_utf16BE[4]={0x30,0x6F,0x00,0x00};    // は
const u8 BA_utf16BE[4]={0x30,0x70,0x00,0x00};    // ば
const u8 PA_utf16BE[4]={0x30,0x71,0x00,0x00};    // ぱ
static IWNN_CHARSET charSet =
{
 2,                                          // charSetCnt
 {(wchar_t*)HA_utf16BE, (wchar_t*)HA_utf16BE, 0},      // from
 {(wchar_t*)BA_utf16BE, (wchar_t*)PA_utf16BE, 0},      // to
};
```

8.8 Option Settings (IWNN_OPTION)

The option settings structure specifies iWnn operational parameters.

Code 8-9 Option Settings (IWNN_OPTION) Structure Configuration

```
struct IWNN_OPTION {
 u16 autoConversionCnt;  // Number of candidates for automatically
                                 starting multiple phrase conversion.
 u16 extensionMode;            // API operational mode setting
 void* phase2Filter;           // Candidate filter function pointer
 void* phase2Option;           // Candidate filter function option
 void* phase1Filter;           // Dictionary lookup filter function pointer
 void* phase1Option;           // Dictionary lookup filter function option
};
```

Table 8-16 Option Settings Structure Members

Member	Description
u16 autoConversionCnt	Number of candidates for automatically starting multiple phrase conversion. If the number of prediction candidates that can be obtained by the get prediction candidates function is less than this value, multiple phrase conversion results are added. The default value is NJ_MAX_CANDIDATE.
u16 extensionMode	API operational mode setting. Sets OR for the API processing method. NJ_ADD_WORD_OPTIMIZE_OFF: Turns off optimization of the learning dictionary when using the register word function. (See section 13.5B.3 FAQ Regarding Processing Methods.) IWNN_OPTION_FORECAST_TOP_FREQUENCY: Enables the retrieval of prefix match candidates in frequency order when using the get prediction candidates function. (See section 13.5B.3 FAQ Regarding Processing Methods.)
void* phase2Filter	Candidate filter function pointer Specifies a pointer to the function performing filtering of phrase candidates generated during the conversion process. When NULL is specified, dictionary lookup filtering is not executed. The default value is NULL.
void* phase2Option	Candidate filter function option. Used to set an option for the candidate filter function. The data to be set is defined for each filter function.
void* phase1Filter	Dictionary lookup filter function pointer. Specifies a pointer to the function performing filtering of words matching search conditions during the dictionary search process. When NULL is specified, dictionary lookup filtering is not executed. The default value is NULL.
void* phase1Option	Dictionary lookup filter function option. Used to set an option for the dictionary lookup filter function. The data to be set is defined for each filter function.

For details on filtering, refer to Chapter 12 Candidate/Dictionary Lookup Filtering.

8.9 Prediction Options (IWNN_ANALYZE_OPTION)

The prediction options structure specifies operational parameters used during prediction and conversion.

Code 8-10 Analysis Options (IWNN_ANALYZE_OPTION) Structure Configuration

```
struct IWNN_ANALYZE_OPTION {
 u16 mode;               // Parsing limit
 u16 forecastLearnLimit; // Maximum number of prediction candidates to
                            get from the learning dictionary
 u16 forecastLimit;         // Maximum number of obtainable prediction
                            candidates
 u8 charMin;             // Minimum number of reading characters
 u8 charMax;             // Maximum number of reading characters
}
```

Table 8-17 Prediction Options Structure Members

Member	Description
u16 mode	Parsing limit (bit specification). NJ_NO_LEARN: No prediction results in learning dictionary. NJ_NO_PREDICTION: No prediction results in prediction dictionary. NJ_NO_MULTI_CONVERSION: No multiple phrase conversion results. NJ_NO_SINGLE_CONVERSION: No single phrase conversion results. NJ_NO_ALL_CANDIDATE: No results from get all candidates. NJ_NO_READING_ON: No reading prediction filter search enabled. NJ_RELATION_ON: Derived prediction filter search enabled. NJ_CLEAR_LEARN_CACHE: Suppress clearing of learning dictionary cache. Recommended value: (NJ_NO_MULTI_CONVERSION | NJ_NO_SINGLE_CONVERSION | NJ_RELATION_ON | NJ_NO_READING_ON)
u16 forecastLearnLimit	Maximum number of prediction candidates to get from the learning dictionary. Limits the number of results retrieved from prefix match prediction analysis and no reading parsing. This parameter is ignored when NULL is specified for the string to be parsed (reading). This parameter is ignored if NJ_NO_LEARN is specified for the parsing limit (mode). Minimum value: 0, Maximum value 100. Recommended value: 20 to number allowed by the candidate window size.

Member	Description
u16 forecastLimit	Maximum number of obtainable prediction candidates. Limits the number of results of prefix match prediction analysis and no reading analysis retrieved from a learning dictionary, standard prediction dictionary, compressed customized dictionary, or uncompressed customized dictionary. This parameter is ignored when NULL is specified for the string to be parsed (reading). Minimum value: 0, Maximum value: 100 (Recommended value: 100). Specify a value equal to or greater than forecastLearnLimit and less than or equal to the maximum number of obtainable candidates (NJ_MAX_CANDIDATE).
u8 charMin	Minimum number of reading characters. Specifies the minimum number of reading characters for words output by prediction. Specify the number of characters according to the typical meaning of a character. Minimum value: 0, Maximum value: NJ_MAX_LEN (default 0).
u8 charMax	Maximum number of reading characters. Specifies the maximum number of reading characters for words output by prediction. Specify the number of characters according to the typical meaning of a character. Minimum value: 0, Maximum value: NJ_MAX_LEN (default NJ_MAX_LEN).

Parsing Limits

These parameters can be used to specify the candidates that can be obtained using this API according to the specification made.

If multiple settings are made, specify the logical sum (OR) of each value.

If prediction candidates for the same reading string are to be obtained, always specify the same parsing limit.

Table 8-18 Prediction Options Parsing Limits

Type	Description
NJ_NO_LEARN	No prediction results in a learning dictionary. Does not get prediction results from learning dictionary.
NJ_NO_PREDICTION	No prediction results in a prediction dictionary. Does not get prediction results from a standard prediction dictionary, compressed customized dictionary, or uncompressed customized dictionary.
NJ_NO_MULTI_CONVERSION	No multiple phrase conversion results. Does not get multiple phrase conversion results. In addition, results are not obtained when retrieval of results is being suppressed by an option setting, even if this is not specified.
NJ_NO_SINGLE_CONVERSION	No single phrase conversion results. Does not get single phrase conversion results.

Type	Description
NJ_NO_ALL_CANDIDATE	No results from get all candidates. Does not get results from get all candidates.
NJ_NO_READING_ON	No reading prediction filter search enabled. Specify this parameter when including prediction results from a no reading prediction dictionary, with prediction results for a specified reading.
NJ_RELATION_ON	Derived prediction filter search enabled. Specify this parameter when including derived prediction results from an integrated dictionary with prediction results for a specified reading.
NJ_NO_CLEAR_LEARN_CACHE	Suppress clearing of learning dictionary cache. Specify this parameter when suppressing the clearing of the cache, for the learning dictionary being used for a fuzzy search.

Specify parsing limits according to the length of strings to be analyzed, because processing to create candidates using multiple/single phrase conversion will take a long time, if the string to be parsed is too long.

8.10 State Setting (IWNN_STATE)

The state setting structure sets the peripheral state required for making situational adaptive predictions.

Code 8-11 State Setting (IWNN_STATE) Structure Configuration

```
struct IWNN_STATE {
 s16 system[NJ_MAX_STATE];                  // Standard state setting
 s16 extension[NJ_MAX_EXTENSION_STATE];     // Extended state setting
 IWNN_STATE_CALC_PARAMETER* calcParam;      // State calculation parameter
}
```

Table 8-19 State Setting Structure Members

Member	Description
`s16` `system`	Standard state setting. This state setting is used internally by iWnn in the conversion process. A bias value is set for each state in each of the elements of the array. The allowable setting range is -1000 to 1000. Each bias value is updated during the iWnn learning process. If a default bias value has already been set by the application, iWnn will operate in line with that state setting.
`s16` `extension`	Extended state setting. Standard state settings can also be used to set unexpressible states. This is not used internally by iWnn. This setting can be freely used with pseudo-dictionaries.
`IWNN_STATE_CALC_PARAMETER*` `calcParam`	State calculation parameter. Sets a parameter used to do things like automatically update the state setting bias value and determine the priority level of candidates using the bias value. Default values are set when `NULL` is specified.

8.10.1 Standard State Settings

The types of states used internally by iWnn are as follows.

8.10.1.1 16 Categories

Table 8-20 Standard State Settings (16 Categories)

Type	Category	State Control		Index of the Standard State Setting Array Description
		App.	iWnn	
Input Field	Person's Name	Yes	—	NJ_CATEGORY_FIELD_PERSON Sets priority for personal names. This is used with name input fields, for instance.
	Noun Concatenation			NJ_CATEGORY_FIELD_NOUN Prioritizes nouns. This is used in situations such as file name input.
	Start of Text		Yes	NJ_CATEGORY_FIELD_HEAD Prioritizes words that start text.
Expression Types	Colloquial Expressions	Yes	Yes	NJ_CATEGORY_EXPRESSION_COLLOQUIAL
	Written Expressions			NJ_CATEGORY_EXPRESSION_WRITTEN
Time Expressions	Time of day: Morning	Yes	Yes	NJ_CATEGORY_TIME_MORNING
	Time of day: Afternoon			NJ_CATEGORY_TIME_NOON
	Time of day: Night			NJ_CATEGORY_TIME_NIGHT
	Season: Spring	Yes	Yes	NJ_CATEGORY_TIME_SPRING
	Season: Summer			NJ_CATEGORY_TIME_SUMMER
	Season: Fall			NJ_CATEGORY_TIME_AUTUMN
	Season: Winter			NJ_CATEGORY_TIME_WINTER
	Relative time: Past	—	○	NJ_CATEGORY_TIME_PAST
	Relative time: Future			NJ_CATEGORY_TIME_FUTURE
Feeling Expressions	Positive Sense	Yes	Yes	NJ_CATEGORY_FEEL_PLUS
	Negative Sense			NJ_CATEGORY_FEEL_MINUS

8.10.1.2 32 Categories

The number of categories can be extended by specifying the NJ_OPT_STATE_TYPE2 compile option.

Table 8-21 Standard State Settings (32 Categories)

Type	Category	State Control		Index of the Standard State Setting Array Description
		App.	iWnn	
Input Field	Person's Name	Yes	—	NJ_CATEGORY_FIELD_PERSON Sets priority for personal names. This is used with name input fields, for instance.
	Noun Continue			NJ_CATEGORY_FIELD_NOUN Prioritizes nouns. This is used in situations such as file name input.
	Start of Text		Yes	NJ_CATEGORY_FIELD_HEAD Prioritizes words that start text.
Expression Types	Colloquial Expressions	Yes	Yes	NJ_CATEGORY_EXPRESSION_COLLOQUIAL
	Written Expressions			NJ_CATEGORY_EXPRESSION_WRITTEN
Time Expressions	Time of day: Morning	Yes	Yes	NJ_CATEGORY_TIME_MORNING
	Time of day: Afternoon			NJ_CATEGORY_TIME_NOON
	Time of day: Night			NJ_CATEGORY_TIME_NIGHT
	Relative time: Past	—	Yes	NJ_CATEGORY_TIME_PAST
	Relative time: Future			NJ_CATEGORY_TIME_FUTURE
Feeling Expressions	Positive Sense	Yes	Yes	NJ_CATEGORY_FEEL_PLUS
	Negative Sense			NJ_CATEGORY_FEEL_MINUS

Type	Category	State Control		Index of the Standard State Setting Array
Month Expressions	Month: January	Yes	Yes	NJ_CATEGORY_MONTH_JANUARY
	Month: February			NJ_CATEGORY_MONTH_FEBRUARY
	Month: March			NJ_CATEGORY_MONTH_MARCH
	Month: April			NJ_CATEGORY_MONTH_APRIL
	Month: May			NJ_CATEGORY_MONTH_MAY
	Month: June			NJ_CATEGORY_MONTH_JUNE
	Month: July			NJ_CATEGORY_MONTH_JULY
	Month: August			NJ_CATEGORY_MONTH_AUGUST
	Month: September			NJ_CATEGORY_MONTH_SEPTEMBER
	Month: October			NJ_CATEGORY_MONTH_OCTOBER
	Month: November			NJ_CATEGORY_MONTH_NOVEMBER
	Month: December			NJ_CATEGORY_MONTH_DECEMBER

Excluding the input field type state, if learning is performed by default, iWnn adds one to the state bias values for all categories to which the learned word belongs. Also, the bias value for the start of text category is always initialized to zero during word learning.

You can retrieve candidates by prioritizing those applicable to the state, by having the application check for changes in the bias value during learning and updating the state setting accordingly. For example, if the past attribute is incremented by one, given a past category with bias of +30 and a future category of bias -800, words in the past category will be prioritized during the next prediction conversion, and words in the future category will be moved to the bottom of the candidate list.

By setting the compile option to `NJ_ADD_STATE_TYPE2`, the state setting bias value can be automatically updated within iWnn to an appropriate value when using the standard dictionary frequency value.

The bias value is added to the candidate frequency value when determining the priority of candidates. For example, 100 is added to the normal frequency value for candidates of a category with a bias value of +100.

The frequency value of the candidate varies in the range 0 to 1000. If the value goes over the upper limit or under the lower limit, it is clamped at either 1000 or 0. In other words, a candidate in a category with a bias value of +1000 will always have a maximum frequency value (1000) and have priority over learning dictionary candidates. Also, a candidate with a bias value of -1000 always has the minimum frequency value (0) and has the lowest priority.

8.11 State Calculation Parameters (IWNN_STATE_CALC_PARAMETER)

The state calculation parameters structure sets parameters used with the following processes.

- Updating the state setting bias value for each category during the learning process.
- Determining the priority order of candidates during situational adaptive prediction.

Code 8-12 State Calculation Parameters (IWNN_STATE_CALC_PARAMETER) Structure Configuration

```
struct IWNN_STATE_CALC_PARAMETER {
s16 sysMaxBias[NJ_MAX_STATE];        // Maximum bias value
s16 sysMinBias[NJ_MAX_STATE];        // Minimum bias value
s16 sysAddBias[NJ_MAX_STATE];        // Value always added to bias
s16 sysBaseBias[NJ_MAX_STATE];       // Bias restore value
s16 sysChangeBias[NJ_MAX_STATE][NJ_MAX_STATE];// Value added to bias during
                                                 learning
s16 dicInfoMax[NJ_MAX_DIC];                // Maximum state setting
                                               dictionary frequency value
s16 dicInfoMin[NJ_MAX_DIC];                // Minimum state setting
                                               dictionary frequency value
}
```

Table 8-22 State Calculation Parameters Structure Members

Member	Description
s16 sysMaxBias	Maximum bias value. Specifies the maximum bias value to be updated internally by iWnn during learning. The allowable range is -1000 to 1000.
s16 sysMinBias	Minimum bias value. Specifies the minimum bias value to be updated internally by iWnn during learning. The allowable range is -1000 to 1000.
s16 sysAddBias	Value always added to bias. Specifies a bias value to always be added to a bias of zero or less during learning. The allowable range is -1000 to 1000.
s16 sysBaseBias	Bias restore value. Specifies a bias value to be restored when the bias is equal to or less than this value. The allowable range is -1000 to 1000.
s16 sysChangeBias	Value added to bias during learning. Specifies the bias value to be added to bias values for each state during learning. The allowable range is -1000 to 1000.
s16 dicInfoMax	Maximum state setting dictionary frequency value. Sets the maximum dictionary frequency value when using situational adaptive prediction. The allowable range is -1000 to 1000.

Member	Description
s16 dicInfoMin	Minimum state setting dictionary frequency value. Sets the minimum dictionary frequency value when using situational adaptive prediction. The allowable range is -1000 to 1000.

8.12 Merge Candidates (IWNN_MERGE_RESULT)

The merge candidates structure is used to store candidate list information formed from merging multiple processing results (IWNN_RESULT). This is used by the merge candidate lists function (**NjxMergeWordList**).

Code 8-13 Merge Candidates (IWNN_MERGE_RESULT) Structure Configuration

```
struct IWNN_MERGE_RESULT {
 IWNN_RESULT* result; // Phrase information
 IWNN_CLASS* iwnn; // Parsing information class
}
```

Table 8-23 Merge Candidates Structure Members

Member	Description
IWNN_RESULT* result	Phrase information (processing result structure). This is the processing result, resulting from obtaining conversions, word searches, or other such processes.
IWNN_CLASS* iwnn	Parsing information class. The parsing information class is used when generating the phrase information (result). This is used to get information such as the reading or notation of a phrase from phrase information.

9 Detailed Descriptions of Functions

9.1 Get Reading String Function (NjxGetStroke)

This function is used to get candidate reading strings from the process result structure.

If the return value indicates an error, the contents of the buffer for storing the reading string (buf) are invalid.

Code 9-1 Get Reading String Function (NjxGetStroke) Declaration

```
s16 NjxGetStroke(
 wchar_t* buf,                  // Reading string storage buffer
 const IWNN_CLASS* iwnn, // Analysis information class
 const IWNN_RESULT* result,    // Process result structure
 u16 bufSize             // Reading string storage buffer byte size
)
```

Table 9-1 Get Reading String Function (NjxGetStroke) Arguments

Input/ Output	Argument	Description
OUT	wchar_t* buf	Reading string storage buffer. Allocates a buffer and specifies a pointer. Allocates memory for a wchar_t array having size of NJ_MAX_LEN + NJ_TERM_SIZE.
IN	const IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
IN	const IWNN_RESULT* result	Process result structure. An error results if NULL is specified.
IN	u16 bufSize	Reading string storage buffer byte size. Specify the size including string terminator. Specify the size in terms of number of bytes.

Table 9-2 Get Reading String Function (NjxGetStroke) Return Values

Return Value	Description
s16	String length of the obtained string (not including the terminator). Negative value: Error.

Table 9-3 Get Reading String Function (NjxGetStroke) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_PARAM_RESULT_NULL	A NULL pointer was specified in the argument (result).
NJ_ERR_PARAM_READING_NULL	A NULL pointer was specified in the argument (reading).
NJ_ERR_BUFFER_NOT_ENOUGH	• A NULL pointer was specified in the argument (buf). • 0 was specified in the argument (bufSize). • The reading string length in the argument (result) is greater than the argument (bufSize).
NJ_ERR_INVALID_RESULT	• Unsupported code was set for operations on the argument (result). • The dictionary to get candidates from did not return a reading string for reverse lookup results. • The argument (result) specifies morphological analysis process results.
NJ_ERR_DIC_TYPE_INVALID	Unsupported code was set for the dictionary type obtained from the word dictionary address given by the argument (result).
NJ_ERR_DIC_BROKEN	Returned when one of the following situations occur when an uncompressed dictionary is targeted by the argument (result): • Queue data stored in the given queue ID is destroyed. • Reading data having a length greater than NJ_MAX_LEN+1 is stored for word data in the dictionary. • The link information when one word uses multiple keys is destroyed.
NJ_ERR_READING_TOO_LONG	The reading string length of the argument (result) is greater than NJ_MAX_LEN.

9.2 Get Candidate String Function (NjxGetCandidate)

This function is used to get a candidate's candidate string (process result) from the process result structure.

If the return value indicates an error, undefined values may be stored in the buffer for storing candidate strings.

Code 9-2 Get Candidate String Function (NjxGetCandidate) Declaration

```
s16 NjxGetCandidate(
 wchar_t* buf,           // Candidate string storage buffer
 const IWNN_CLASS* iwnn,       // Analysis information class
 const IWNN_RESULT* result,    // Process result structure
 u16 bufSize             // Buffer byte size
)
```

Table 9-4 Get Candidate String Function (NjxGetCandidate) Arguments

Input/ Output	Argument	Description
OUT	wchar_t* buf	Candidate string storage buffer. Allocates a buffer and specifies a pointer. If the candidate is a result of morphological analysis, allocate a buffer of the size MM_MAX_MORPHOLIZE_LEN+NJ_TERM_SIZE. If the result of other processing, allocate a wchar_t array having the size NJ_MAX_RESULT_LEN+NJ_TERM_SIZE.
IN	const IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
IN	const IWNN_RESULT* result	Process result structure An error results if NULL is specified.
IN	u16 bufSize	Candidate string storage buffer byte size. Specify the size including string terminator. Specify the size in terms of number of bytes.

Table 9-5 Get Candidate String Function (NjxGetCandidate) Return Values

Return Value	Description
s16	String length of the obtained string (not including the terminator). Negative value: Error.

Table 9-6 Get Candidate String Function (NjxGetCandidate) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_PARAM_RESULT_NULL	A NULL pointer was specified in the argument (result).
NJ_ERR_BUFFER_NOT_ENOUGH	• A NULL pointer was specified in the argument (buf). • 0 was specified in the argument (bufSize). • The reading string length in the argument (result) is greater than the argument (bufSize).
NJ_ERR_INVALID_RESULT	• Unsupported code was set for operations on the argument (result). • 0 was specified for the candidate string length in the argument (result).
NJ_ERR_CANDIDATE_TOO_LONG	A value greater than NJ_MAX_RESULT_LEN was set for the candidate string length in the argument (result).

Error Code	Description
NJ_ERR_DIC_TYPE_INVALID	Unsupported code was set for the dictionary type obtained from the single-word dictionary address given by the argument (result).
NJ_ERR_DIC_BROKEN	Returned when one of the following situations occur when an uncompressed dictionary is targeted by the argument (result): • The queue ID (result->word.stem.location.current) is destroyed. • The reading string in a user dictionary is longer than or equal to NJ_MAX_USER_LEN, or the candidate string is longer than or equal to NJ _MAX_USER_CANDIDATE_LEN. • The reading string in a dictionary other than a user dictionary is longer than NJ_MAX_USER_LEN, or the candidate string is longer than to NJ_MAX_USER_CANDIDATE_LEN. • When one word uses multiple keys, link information is destroyed.

9.3 Get Dictionary Handle Function (NjxGetDicHandle)

This function is used to get a dictionary handle from the processing result structure.

Code 9-3 Get Dictionary Handle Function (NjxGetDicHandle) Declaration

```
IWNN_DIC_HANDLE  NjxGetDicHandle(
 const IWNN_CLASS* iwnn,       // Analysis information class
 const IWNN_RESULT* result     // Process result structure
)
```

Table 9-7 Get Dictionary Handle Function (NjxGetDicHandle) Arguments

Input/ Output	Argument	Description
IN	const IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
IN	const IWNN_RESULT* result	Process result structure. “Unable to retrieve” is returned if NULL is specified.

Table 9-8 Get Dictionary Handle Function (NjxGetDicHandle) Return Values

Return Value	Description
IWNN_DIC_HANDLE	Dictionary handle. NULL indicates the handle cannot be obtained.

Table 9-9 Get Dictionary Handle Function (NjxGetDicHandle) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).

The dictionary handle cannot be obtained when the following process result structure specifications have been made.

- If the operation information is NJ_FUNC_CONVERT_MULTIPLE or NJ_FUNC_CONVERT_SINGLE for process results from the get prediction candidate function (**NjxAnalyze**).
- If the operation information is NJ_DIC_PSEUDO for process results.
- If process results are NULL.

If process results (IWNN_RESULT) were obtained using the conversion function (**NjxConversion**) or get all candidates function (**NjxAllCandidates**), the dictionary handle can be obtained for independent words included in process results.

9.4 Create Dictionary Function (NjxCreateDic)

This function writes dictionary header information into the specified user dictionary or learning dictionary, and initializes the dictionary. To allocate memory for these dictionaries, you must specify the dictionary handle and its size.

Code 9-4 Create Dictionary Function (NjxCreateDic) Declaration

```
s16 NjxCreateDic(
 IWNN_DIC_HANDLE handle, // Dictionary handle
 const IWNN_CLASS* iwnn,       // Analysis information class
 s8 type,                // Dictionary type
 u32 size                // Dictionary byte size
)
```

Table 9-10 Create Dictionary Function (NjxCreateDic) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_DIC_HANDLE Handle	The dictionary handle of a user dictionary or learning dictionary. An error results if NULL is specified.
IN	const IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
IN	u8 type	Dictionary type. NJ_CREATE_DIC_TYPE_USER: User dictionary. NJ_CREATE_DIC_TYPE_LEARN_AWNN: Learning dictionary -AWnn type. NJ_CREATE_DIC_TYPE_LEARN: Learning dictionary -iWnn type. NJ_CREATE_DIC_TYPE_USER_ADDITIONAL: User dictionary (additional information). NJ_CREATE_DIC_TYPE_LEARN_ADDITIONAL: Learning dictionary -iWnn type- (additional information).
IN	u32 size	Byte size of the memory region specified by handle. This is set at either NJ_USER_DIC_SIZE (dictionary without additional information) or NJ_USER2_DIC_SIZE (dictionary with additional information) for user dictionaries. A learning structure is created according to the size of the specified memory region for learning dictionaries. An error results if the memory cannot be initialized as a dictionary.

Table 9-11 Create Dictionary Function (NjxCreateDic) Return Values

Return Value	Description
s16	Negative value: Error. Other values indicate normal termination.

Table 9-12 Create Dictionary Function (NjxCreateDic) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_DIC_HANDLE_NULL	NULL was specified in the argument (handle).
NJ_ERR_CREATE_TYPE_INVALID	A value other than 0, 1, or 2 was specified in the argument (type).
NJ_ERR_AREA_SIZE_INVALID	• A value less than the minimum size required to create a dictionary was specified in the argument (size). • A value where the queue size exceeds 256KB was specified in the argument (size).

With iWnn, to account for restoration of the learning memory region and word registration region, writing to these memory regions is performed in units of u16 (2 bytes). Warnings are, therefore, sometimes output when compiling this system.

In addition, if the start address of the dictionary handle specified to the function for creating a dictionary memory region is odd (depending on the system), there is a chance of a bus error occurring. So be sure the dictionary handle specified to this function is assigned to an even address.

9.4.1 Size and Number of Registered Entries in User Dictionaries

The dictionary size (NJ_USER_DIC_SIZE) of a user dictionary is calculated based on:

- The maximum reading string length of words registered in the user dictionary (NJ_MAX_USER_LEN).
- The maximum candidate string length of words registered in the user dictionary (NJ_MAX_USER_CANDIDATE_LEN).
- The maximum number of words registered in the user dictionary (NJ_MAX_USER_COUNT).

Table 9-13 Size and Number of Registered Entries in User Dictionaries

Category	Description
Header Region	80 bytes.
Memory Per Entry	Maximum reading string length of words registered in the user dictionary *sizeof(wchar_t) + Maximum candidate string length of words registered in the user dictionary *sizeof(wchar_t) + 9 bytes.

For example, when using Unicode (UTF-16BE), if the maximum reading string length of words registered in the user dictionary (NJ_MAX_USER_LEN) and the maximum candidate string length of words registered in the user dictionary (NJ_MAX_USER_CANDIDATE_LEN) are both 30, and the maximum number of words registered in the user dictionary (NJ_MAX_USER_COUNT) is 100, a memory region of the following size is required:

80 + (((30 * 2) + (30 * 2) + 9) * 100) = 12980 bytes

9.4.2 Size and Number of Registered Entries in User Dictionaries (With Additional Information)

The dictionary size (NJ_USER2_DIC_SIZE) of a user dictionary (with additional information) is calculated based on:

- The maximum reading string length of words registered in the user dictionary (NJ_MAX_USER_LEN).
- The maximum candidate string length of words registered in the user dictionary (NJ_MAX_USER_CANDIDATE_LEN).
- The maximum additional information string length for words registered in the user dictionary (NJ_MAX_ADDITIONAL_LEN).
- The maximum number of words registered in the user dictionary (NJ_MAX_USER_COUNT).

Table 9-14 Size and Number of Registered Entries in User Dictionaries (With Additional Information)

Category	Description
Header region	80 bytes.
Memory per entry	Maximum reading string length of words registered in the user dictionary `*sizeof(wchar_t)` + Maximum candidate string length of words registered in the user dictionary `*sizeof(wchar_t)` + Maximum additional information string length for words registered in the user dictionary `*sizeof(wchar_t)` + 9 bytes.

For example, when using Unicode (UTF-16BE), if the maximum reading string length of words registered in the user dictionary (`NJ_MAX_USER_LEN`), the maximum candidate string length of words registered in the user dictionary (`NJ_MAX_USER_CANDIDATE_LEN`), and the maximum additional information string length for words registered in the user dictionary are all 30, and the maximum number of words registered in the user dictionary (`NJ_MAX_USER_COUNT`) is 100, a memory region of the following size is required:

80 + (((30 * 2) + (30 * 2) +(30 * 2) + 9) * 100) = 18980 Byte

9.4.3 Size and Number of Registered Entries in Learning Dictionaries

The dictionary size of a learning dictionary depends on the maximum number of registered phrases.

Table 9-15 Size and Number of Registered Entries in Learning Dictionaries (`iWnn` Type)

Category	Description
Header Region	80 bytes
Memory Per Entry	42 bytes

If 1000 entries (maximum 1000 phrases) are registered, a memory region of the size given below is required.

80 + (42 * 1000) = 42080 bytes

Table 9-16 Size and Number of Registered Entries in Learning Dictionaries (`iWnn` Type With Additional Information)

Category	Description
Header Region	80 bytes
Memory Per Entry	44 + Maximum additional information string length `*sizeof(wchar_t)` bytes.

If 1000 entries (maximum 1000 phrases) are registered, a memory region of the size given below is required.

80 + (84 * 1000) = 84080 bytes

Note: If the maximum additional information string length is 40.

Table 9-17 Size and Number of Registered Entries in Learning Dictionaries (`AWnn` Type)

Category	Description
Header Region	80 bytes
Memory Per Entry	36 bytes

If 1000 entries (maximum 1000 phrases) are registered, a memory region of the size given below is required.

80 + (36 * 1000) = 36080 bytes

In the case of both types, one phrase can be learned in one registration area for phrases whose reading length plus candidate length fits in 27 bytes. If the reading length plus candidate length of a single phrase exceeds 27 bytes, single-phrase learning is performed using multiple registration areas in order to consume memory in terms of 27-byte units.

Due to the specifications for predictive text input and kana/kanji conversion, the number of learned words (phrases) that can be used is the maximum number of phrases that can be registered in the learning dictionary minus the maximum conversion candidate string length (`NJ_MAX_RESULT_LEN`).

For example, if a learning dictionary can register a maximum of 1,000 phrases and the longest conversion candidate string length is 55, the number of learned words (phrases) that can be used would be as follows:

1000 - 55 = 945 phrases

9.5 Initialize Function (`NjxInit`)

This function initializes iWnn variables.

Always call this function when starting to use iWnn and after changing dictionary sets.

Code 9-5 Initialize Function (`NjxInit`) Declaration

```
s16 NjxInit(
 IWNN_CLASS* iwnn,              // Analysis information class
 const IWNN_OPTION* option      // Option settings
)
```

Table 9-18 Initialize Function (NjxInit) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
IN	const IWNN_OPTION* option	Option settings. Settings specified for the analysis information class are set by this function. Default values are set in the analysis information class if NULL is specified.

Table 9-19 Initialize Function (NjxInit) Return Values

Return Value	Description
s16	Negative value: Error. Other values indicate normal termination.

Table 9-20 Initialize Function (NjxInit) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).

9.6 Check Dictionary Function (NjxCheckDic)

This function checks the integrity and compatibility of dictionaries, and automatically restores them.

Restoration can be executed by specifying the automatic restoration flag, but only when the dictionary handle of a user dictionary or learning dictionary has been specified. An error results if automatic restoration is impossible. If an error occurs, re-create the user or learning dictionary using the create dictionary function (**NjxCreateDic**).

Only a minimal dictionary check is made. Sometimes corruption of the dictionary is detected while using the dictionary, even though no problem was found with the integrity or compatibility of the dictionary. Try automatic restoration if an error related to dictionary corruption occurs while using this function.

Code 9-6 Check Dictionary Function (NjxCheckDic) Declaration

```
s16 NjxCheckDic(
 IWNN_DIC_HANDLE handle, // Dictionary handle
 const IWNN_CLASS* iwnn, // Analysis information class
 u8 dicType,             // Dictionary handle type
 u8 restore,             // Automatic restoration flag
 u32 size                // Dictionary byte size
)
```

Table 9-21 Check Dictionary Function (NjxCheckDic) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_DIC_HANDLE Handle	Dictionary handle to be checked. An error results if NULL is specified.
IN	const IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
IN	u8 dicType	The dictionary handle type of the dictionary to be checked.
IN	u8 restore	Automatic restoration flag. Restoration is performed automatically if the automatic restoration flag is set for a user dictionary or learning dictionary. 1: Perform automatic restoration. 0: Do not perform restoration (check dictionary only).
IN	u32 size	Dictionary byte size. Byte size of the dictionary specified by handle.

Table 9-22 Check Dictionary Function (NjxCheckDic) Return Values

Return Value	Description
s16	Negative value: Error. Other values indicate normal termination.

Table 9-23 Check Dictionary Function (NjxCheckDic) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_DIC_HANDLE_NULL	NULL was specified in the argument (dic).
NJ_ERR_INVALID_FLAG	An invalid value was specified in the argument (restore).
NJ_ERR_AREA_SIZE_INVALID	• A value less than or equal to the dictionary header size was specified in the argument (size). • The dictionary size calculated from information in the dictionary header given by handle does not match that in the argument (size).

Error Code	Description
NJ_ERR_FORMAT_INVALID	• The ID for the dictionary start/end is invalid. • The dictionary version inside the dictionary area is invalid. • The maximum reading length for words stored in the dictionary, as given in the dictionary header, exceeds NJ_MAX_LEN. • The maximum candidate length for words stored in the dictionary, as given in the dictionary header, exceeds NJ_MAX_RESULT_LEN. • An unsupported dictionary type is set in the dictionary header.
NJ_ERR_DIC_BROKEN	• 0 was specified in the argument (restore). • The number of registered entries given in the dictionary header exceeds the maximum number of registerable words. • An invalid value is stored in the index area inside the dictionary. • Backup data in case of power failure is corrupted. • Status allows recovery from power failure.
NJ_ERR_CANNOT_RESTORE	• 1 was specified in the argument (restore). • The number of registered entries given in the dictionary header exceeds the maximum number of registerable words. • An invalid value is stored in the index area inside the dictionary. • Backup data in case of power failure is corrupted. • An attempt was made to recovery from power failure, but status does not allow recovery.

9.7 Get Character Type Function (NjxGetCharType)

This function gets the string type of the specified result as an independent word or ancillary word.

Because the dividing point between independent words and ancillary words cannot be identified for multiple and single-phrase conversion results obtained by the prediction function, and candidates obtained from the learning dictionary, the character type is stored only for the independent word type (stemType).

If an error occurs, an invalid value may be stored in the character type (stemType or ancillaryWordType).

Table 9-24 List of Character Types

Definition	Description
NJ_TYPE_HIRAGANA	Hiragana.
NJ_TYPE_KATAKANA	Full-width katakana.
NJ_TYPE_HALF_KATAKANA	Half-width katakana.
NJ_TYPE_HALF_NUMERIC	Half-width numeric (Arabic notation).
NJ_TYPE_FULL_NUMERIC	Full-width character (Arabic notation).
NJ_TYPE_UNDEFINE	Other than the above. (Including cases where character types are mixed, or half-width and full-width characters are mixed.)

If stemType is other than NULL and there are no independent words, the character type becomes NJ_TYPE_NONE. This is the same for ancillary words.

Depending on the where the process result (result) was obtained from, it may not be possible to distinguish between independent words and ancillary words.

Table 9-25 When Independent/Ancillary Words Cannot be Distinguished From the Source of the Process Result (result)

Function for Getting Results	Operation	Note
NjxAnalyze	The character type is set in the argument (stemType) for all process results. NJ_TYPE_NONE is always set for the argument (ancillaryWordType).	Because multiple phrase analysis results and single-phrase analysis results consist of multiple phrases, character type is determined by this API for candidates that consist only of independent words. Because independent words and ancillary words cannot be distinguished for candidates obtained from the learning dictionary, this function determines the character type for candidates made up only of independent words.
NjxConversion	Basic get character type operation.	Because independent words and ancillary words cannot be distinguished for candidates obtained from the learning dictionary, this function determines the character type for candidates made up only of independent words.
NjxAllCandidates	Basic get character type operation.	Because independent words and ancillary words cannot be distinguished for candidates obtained from the learning dictionary, this function determines the character type for candidates made up only of independent words.

Function for Getting Results	Operation	Note
NjxGetWord	Basic get character type operation.	Because independent words and ancillary words cannot be distinguished for candidates obtained from the learning dictionary, this function determines the character type for candidates made up only of independent words.
MmxSplitWord	Basic get character type operation.	-

Code 9-7 Get Character Type Function (NjxGetCharType) Declaration

```
s16 NjxGetCharType(
 u8* stemType,                  // Independent word character type
 u8* ancillaryWordType,  // Ancillary word character type
 const IWNN_CLASS* iwnn, // Analysis information class
 const IWNN_RESULT* result     // Process result
)
```

Table 9-26 Get Character Type Function (NjxGetCharType) Arguments

Input/ Output	Argument	Description
OUT	u8* stemType	Independent word character type. The character type is not identified if NULL is specified.
OUT	u8* ancillaryWordType	Ancillary word character type. The character type is not identified if NULL is specified.
IN	const IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
IN	const IWNN_RESULT* result	Process result. An error results if NULL is specified.

Table 9-27 Get Character Type Function (NjxGetCharType) Return Values

Return Value	Description
s16	Negative value: Error. 0 indicates normal termination. Returns 0 if both stemType and ancillaryWordType are NULL.

Table 9-28 Get Character Type Function (NjxGetCharType) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_PARAM_RESULT_NULL	A NULL pointer was specified in the argument (result).
NJ_ERR_INVALID_RESULT	Unsupported code was set for operations on the argument (result).
NJ_ERR_CANDIDATE_TOO_LONG	The candidate string length set in the *result* argument was greater than NJ_MAX_RESULT_LEN+1.
NJ_ERR_DIC_TYPE_INVALID	Unsupported code was set for the dictionary type obtained from the single-word dictionary address given by the argument (result).
NJ_ERR_DIC_BROKEN	Returned when one of the following situations occur when an uncompressed dictionary is targeted by the argument (result): • The queue ID (result->word.stem.location.current) is destroyed. • When a reading string in the user dictionary longer than NJ_MAX_USER_LEN or a candidate string longer than NJ_MAX_USER_CANDIDATE_LEN is found. • The link information when one word uses multiple keys is destroyed.

9.8 Change Dictionary Type Function (NjxChangeDicType)

This function changes the specified learning dictionary into an uncompressed custom dictionary.

This function can also change an uncompressed custom dictionary, made from a learning dictionary, back into a learning dictionary.

Because only dictionary header information inside the specified dictionary handle changes when using this function, be sure to check dictionary data and verify integrity using the function for checking dictionaries (**NjxCheckDic**).

When a learning dictionary (with additional information) is converted, the type is changed to an uncompressed custom dictionary. When an uncompressed custom dictionary is converted, the type is changed to a learning dictionary (with additional information).

Code 9-8 Change Dictionary Type Function (NjxChangeDicType) Declaration

```
s16 NjxChangeDicType(
 IWNN_DIC_HANDLE handle, // Dictionary handle
 const IWNN_CLASS* iwnn, // Analysis information class
 u8 direct               // Dictionary type to change to
)
```

Table 9-29 Change Dictionary Type Function (NjxChangeDicType) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_DIC_HANDLE handle	Dictionary handle. An error results if NULL is specified.
IN	const IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
IN	u8 direct	Dictionary type to change to: 0: Change from a learning dictionary to an uncompressed custom dictionary. 1: Change from an uncompressed custom dictionary to a learning dictionary. Note that an uncompressed custom dictionary created using the custom dictionary creation tool cannot be changed to a learning dictionary.

Table 9-30 Change Dictionary Type Function (NjxChangeDicType) Return Values

Return Value	Description
s16	Negative value: Error. 0 indicates normal termination.

Table 9-31 Change Dictionary Type Function (NjxChangeDicType) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_DIC_HANDLE_NULL	A NULL pointer was specified in the argument (handle).
NJ_ERR_DIC_TYPE_INVALID	A dictionary other than a learning dictionary or uncompressed custom dictionary was specified in the argument (handle).
NJ_ERR_INVALID_FLAG	An invalid value was specified in the argument (direct). The dictionary type specified in the argument (handle) cannot be changed to the dictionary type specified in the argument (direct).
NJ_ERR_DIC_VERSION_INVALID	A dictionary from Advanced Wnn V1.11 or earlier was specified in the argument (handle).

9.9 Get Prediction Candidate Function (NjxAnalyze)

This function is used to get a list of best candidates. This includes prediction candidate results (fuzzy prediction candidates), kana-kanji conversion results (multiple and single-phrase conversion), and get all candidate results.

If the return value is indicates no candidate (0) or an error, an undefined value may be stored in the processing result (result).

Processing results can be obtained one at a time in the following order by calling this function once.

When a reading is specified:

1. Prediction results from a learning dictionary or dictionary that allows prefix match searches.
2. Multiple phrase conversion results.
3. Single phrase conversion results.
4. Get all candidates results.

When a reading is not specified:

(When getting a prediction candidate to follow one of the above results “When a reading is specified”):

- Relationship prediction results of a learning dictionary, no reading prediction dictionary, integrated dictionary, or relationship prediction dictionary.

When a reading is not specified:

(When getting a prediction candidate to follow a start-of-text part of speech when the state setting category has been set to “start of text”):

- Relationship prediction results of a learning dictionary, no reading prediction dictionary, integrated dictionary, or relationship prediction dictionary.

The dictionaries used for each prediction/conversion process are given below.

Table 9-32 Dictionaries Used for Each Prediction/Conversion Process

Type of Dictionary	Prediction	Multiple/single-Phrase Conversion	Get All Candidates	Relationship Prediction
Integrated Dictionary	◎	○	○	○
Single Kanji Dictionary	–	–	○	–
User Dictionary	◎	○	○	–
Learning Dictionary	◎	○	○	○
No Reading Prediction Dictionary	○	–	–	○
Ancillary Word Dictionary		○	○	–
Compressed Custom Dictionary (Dictionary for forward lookup complete match search)	–	–	○	–
Compressed Custom Dictionary (Dictionary for forward lookup prefix match search)	◎	–	○	–

Type of Dictionary	Prediction	Multiple/single-Phrase Conversion	Get All Candidates	Relationship Prediction
Compressed Custom Dictionary (Dictionary for reverse lookup complete match search)	◎	—	○	—
Uncompressed Custom Dictionary	◎	—	○	○
Rule Dictionary	○	○	○	○

(◎: Use (fuzzy search included), ○: Use (no fuzzy search), —: Not used)

Be sure to include an ancillary word dictionary in the dictionary set when using multiple phrase conversion and single-phrase conversion results.

The longer the input character string, the more time required for multiple and single-phrase analysis. A means for dealing with this, such as applying a limit based on input string length, may be required when getting multiple phrase ad single-phrase analysis results under high-speed button input conditions.

These operational settings are made using prediction options (`IWNN_ANALYZE_OPTION`). For details, refer to section 8.9 Prediction Options (IWNN_ANALYZE_OPTION).

Code 9-9 Get Prediction Candidate Function (`NjxAnalyze`) Declaration

```
s16 NjxAnalyze(
 IWNN_CLASS* iwnn,              // Analysis information class
 IWNN_RESULT* result,                 // Process result structure
 const IWNN_CHARSET* charSet,  // Fuzzy character set structure
 const wchar_t* reading,              // String to be analyzed
 const IWNN_ANALYZE_OPTION* option   // Analysis option
)
```

Table 9-33 Get Prediction Candidate Function (NjxAnalyze) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
OUT	IWNN_RESULT* Result	Analysis result buffer. Memory having a size of sizeof(IWNN_RESULT) bytes or more must be provided. An error results if NULL is specified.
IN	const IWNN_CHARSET* charSet	Fuzzy character set structure for making fuzzy searches. Specify NULL when not being used. Do not change the content of this buffer until all operations have ended because it is used internally by the system.
IN	const wchar_t* reading	String to be analyzed. Be sure to add a terminator at the end of the string. If NULL is specified, the character string to be analyzed during the previous analysis will be used again. If an empty string ("") is specified, optimum analysis results to be connected to previously learned results (NjxSelect) can be obtained. Do not change the contents of the buffer until all operations are complete because this character string memory is used internally by iWnn.
IN	const IWNN_ANALYZE_OPTION* option	Analysis option. Sets processing to be performed, such as the type of candidate to be obtained or the maximum number of candidates to get. If NULL is specified, operations are performed using recommended and default prediction options. **Note:** See 8.9 Prediction Options (IWNN_ANALYZE_OPTION).

Table 9-34 Get Prediction Candidate Function (NjxAnalyze) Return Values

Return Value	Description
s16	Negative value: Error. 0: No candidates. 1: Candidates present.

Table 9-35 Get Prediction Candidate Function (NjxAnalyze) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_PARAM_DIC_NULL	A NULL pointer was specified in the argument (iwnn->dicSet).
NJ_ERR_NO_RULE_DIC	A rule dictionary is not set in the argument (iwnn->dicSet).
NJ_ERR_PARAM_RESULT_NULL	A NULL pointer was specified in the argument (result).

Error Code	Description
NJ_ERR_NOT_SELECT_YET	NULL was specified in the argument (reading), but there are no prior prediction conditions.
NJ_ERR_PARAM_READING_SIZE	A reading string with length longer than NJ_MAX_LEN was specified in the argument (reading).
NJ_ERR_PARAM_ILLEGAL_LIMIT	An illegal numeric value was specified in the argument (option).
NJ_ERR_DIC_BROKEN	• Cannot get the location to add learning dictionary information, user dictionary information from the learning dictionary, or user dictionary specified in the argument (iwnn->dicSet). • The number of registered words given in the learning dictionary or user dictionary specified in the argument (iwnn->dicSet) is larger than the maximum number of registerable words. • The queue data inside the learning dictionary or user dictionary specified in the argument (iwnn->dicSet) is corrupted.
NJ_ERR_NO_PARTS_OF_SPEECH	The required part-of-speech information cannot be obtained from the rule dictionary.
NJ_ERR_DIC_TYPE_INVALID	A dictionary other than that defined was set in the argument (iwnn->dicSet).
NJ_ERR_CACHE_BROKEN	The cache management area specified in the argument (iwnn->dicSet) has been destroyed.
NJ_ERR_PROTECTION_ERR	A function that operates on learning dictionaries has performed an operation on protected memory.

9.10 Kana-Kanji Conversion Function (NjxConversion)

This function performs single-phrase conversion or multiple phrase conversion of the specified reading string for the specified dictionary set. If the locations of word divisions are specified, multiple phrase conversion based on those divisions is performed.

Conversion results are stored in the conversion result storage buffer for each phrase. If the number of phrases to be analyzed is specified, analysis is performed only up to the specified number of phrases to be analyzed.

This API uses the following dictionaries inside the specified dictionary set (iwnn->dicSet).

- Integrated dictionary.
- User dictionary.
- Learning dictionary.
- Ancillary word dictionary.
- Compressed custom dictionary.
- Uncompressed custom dictionary.
- Rule dictionary.

The single-kanji dictionary is also used if the location of phrase divisions has been specified.

Code 9-10 Kana-Kanji Function (NjxConversion) Declaration

```
s16 NjxConversion(
 IWNN_CLASS* iwnn,              // Analysis information class
 IWNN_RESULT* results,    // Conversion result storage buffer
 const wchar_t* reading,        // Reading string
 u8 analyzeLevel,         // Number of phrases to analyze
 u8 devidePos                   // Phrase division position
)
```

Table 9-36 Kana-Kanji Function (NjxConversion) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
OUT	IWNN_RESULT* results	Conversion result storage buffer. Enough memory must be allocated to store the number of phrases to be analyzed.
IN	const wchar_t* reading	Reading string to undergo multiple phrase conversion. Be sure to add a terminator to the end of the string. Do not change this value until conversion operations have ended, because this string region is used (overwritten) internally by iWnn during kana-kanji conversion.
IN	u8 analyzeLevel	Number of phrases to analyze. Specify 1 or NJ_MAX_PHRASE. Single-phrase conversion is performed if 1 is specified. Multiple phrase conversion is performed if NJ_MAX_PHRASE is specified. An error results if a value other than 1 or NJ_MAX_PHRASE is specified.
IN	u8 devidePos	Phrase division position. Use when explicitly specifying where the phrase has been divided. Specify the division position in terms of string length (using the immediately following array element index number). Set to 0 if phrase division positions are not being specified.

Table 9-37 Kana-Kanji Function (NjxConversion) Return Values

Return Value	Description
s16	Positive value: Number of phrases (1 or more) stored in the conversion result storage buffer. Negative value: Error.

Table 9-38 Kana-Kanji Function (NjxConversion) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in an argument (iwnn).
NJ_ERR_PARAM_ILLEGAL_LEVEL	• A value other than 1 or NJ_MAX_PHRASE was set in the argument (analyzeLevel). • A value of 1 was set for the argument (analyzeLevel) when a value other than 0 was set for the argument (devidePos).
NJ_ERR_DIC_BROKEN	• Cannot get the location to add learning dictionary information or user dictionary information from the learning dictionary or user dictionary specified in the argument (iwnn->dicSet). • The number of registered words given in the learning dictionary or user dictionary specified in the argument (iwnn->dicSet) is larger than the maximum number of registerable words. • The queue data inside the learning dictionary or user dictionary specified in the argument (iwnn->dicSet) is corrupted.
NJ_ERR_PARAM_RESULT_NULL	A NULL pointer was specified in the argument (result).
NJ_ERR_PARAM_READING_NULL	A NULL pointer was specified in the argument (reading)
NJ_ERR_PARAM_READING_SIZE	An empty character string or reading string longer than NJ_MAX_LEN was specified in the argument (reading).
NJ_ERR_PARAM_DIVISION	A value greater than the length of the specified reading string was specified in the argument (devidePos).
NJ_ERR_NO_RULE_DIC	A rule dictionary is not set in the argument (iwnn->dicSet).
NJ_ERR_NO_PARTS_OF_SPEECH	Required part-of-speech information cannot be obtained from the rule dictionary.
NJ_ERR_PROTECTION_ERR	A function that operates on learning dictionaries has performed an operation on protected memory.
NJ_ERR_DIC_TYPE_INVALID	An unsupported dictionary was set in the argument (iwnn->dicSet).

9.11 Get All Candidates Function (NjxAllCandidates)

This function returns one result for the specified position out of all candidates for the specified candidate.

An error results if this function is used in a situation where the initialize function and kana-kanji conversion function have not already been called.

If the return value indicates that there is no candidate (return value 0), or if there is an error, undefined values may be stored in the buffer for getting candidates (result).

This API uses the following dictionaries inside the specified dictionary set (`iwnn->dicSet`).

- Integrated dictionary.
- Single kanji dictionary.
- User dictionary.
- Learning dictionary.
- Ancillary word dictionary.
- Compressed custom dictionary.
- Uncompressed custom dictionary.
- Rule dictionary.

Code 9-11 Get All Candidates Function (`NjxAllCandidates`) Declaration

```
s16 NjxAllCandidates(
 IWNN_CLASS* iwnn,              // Analysis information class
 IWNN_RESULT* result,           // Obtained candidate storage buffer
 const IWNN_RESULT* target,     // Target phrase conversion results
 u16 candidateNum        // Obtained candidate number
)
```

Table 9-39 Get All Candidates Function (`NjxAllCandidates`) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
OUT	IWNN_RESULT* result	Obtained candidate storage buffer. Enough memory to store one candidate must be allocated.
IN	const IWNN_RESULT* target	Target phrase conversion results. Be sure to specify the process result structure for the phrase position for which all candidates are to be obtained from inside the conversion result storage buffer for the kana-kanji conversion function. Be sure to specify NULL when getting candidates other than for the first time (when getting the second or later candidate). However, an error is returned if NULL is specified when getting candidates the first time (when getting the first candidate).
IN	u16 candidateNum	Obtained candidate number (starting from 0). Be sure to specify the index of the candidate to be obtained from all candidates. If a negative value is specified, or if the candidate to be obtained does not exist (the upper limit was reached), 0 is returned.

Table 9-40 Get All Candidates Function (NjxAllCandidates) Return Values

Return Value	Description
s16	Negative value: Error. Positive value: Number of candidates when getting all candidates. 0: No candidate to be obtained.

Table 9-41 Get All Candidates Function (NjxAllCandidates) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_NOT_CONVERTED	This API was called when conversion has not been performed.
NJ_ERR_NO_CANDIDATA_LIST	NULL was set in the argument (target) immediately after conversion.
NJ_ERR_PARAM_RESULT_NULL	NULL was set in the argument (result).
NJ_ERR_NO_RULE_DIC	A rule dictionary is not set in the argument (iwnn->dicSet).
NJ_ERR_NO_PARTS_OF_SPEECH	Required part-of-speech information cannot be obtained from the rule dictionary.
NJ_ERR_DIC_TYPE_INVALID	An unsupported dictionary was set in the argument (iwnn->dicSet).
NJ_ERR_DIC_BROKEN	• Cannot get the location to add learning dictionary information, user dictionary information from the learning dictionary, or user dictionary specified in the argument (iwnn->dicSet). • The number of registered words given in the learning dictionary or user dictionary specified in the argument (iwnn->dicSet) is larger than the maximum number of registerable words. • The queue data inside the learning dictionary or user dictionary specified in the argument (iwnn->dicSet) is corrupted.
NJ_ERR_INVALID_RESULT	Process results obtained by a function other than NjxConversion or NjxAllCandidates were specified in the argument (target).

9.12 Learning Function (NjxSelect)

This function performs the following operations, on the process result structures that may be specified.

- Process result structure (for learning).
  Registers entries in a learning dictionary or pseudo dictionary, based on learning information in the process result structure.
- Process result structure (for pre-confirmation information).
  Sets learning information in the process result structure, as pre-confirmation information.

Do not specify results obtained from the delimited input function for morphological analysis to this function. If learning of morphological analysis results is needed, use the morphological analysis learning function.

Code 9-12 Learning Function (`NjxSelect`) Declaration

```
s16 NjxSelect(
 IWNN_CLASS* iwnn,                    // Analysis information class
 const IWNN_RESULT* learningResult,   // Process result structure(for
                                                learning)
 const IWNN_RESULT* preConfirmedResult,    // Process result structure(for
                                              pre-confirmation information)
 u8 connect                           // Associative learning flag
)
```

Table 9-42 Learning Function (`NjxSelect`) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
IN	const IWNN_RESULT* learningResult	Process result structure (for learning). The result of learning is not registered in the learning dictionary if NULL is specified.
IN	const IWNN_RESULT* preConfirmedResult	Process result structure (for pre-confirmation information). Pre-confirmation information is deleted if NULL is specified.
IN	u8 connect	Associative learning flag. Specifies whether to perform associative learning versus pre-confirmed candidates. 0: Do not perform associative learning. 1: Perform associative learning.

Table 9-43 Learning Function (`NjxSelect`) Return Values

Return Value	Description
s16	Negative value: Error. Other: Normal exit.

Table 9-44 Learning Function (NjxSelect) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_NO_RULE_DIC	A rule dictionary is not set in the argument (iwnn->dicSet).
NJ_ERR_INVALID_FLAG	A value other than 0 or 1 was set in the argument (connect).
NJ_ERR_INVALID_RESULT	• Unsupported code was set for an operation on the arguments (learningResult or preConfirmedResult). • The dictionary from which candidates are obtained does not return reading strings for reverse lookup results. • The arguments (learningResult, preConfirmedResult) represent the results of morphological analysis.
NJ_ERR_DIC_TYPE_INVALID	Unsupported code was set for the dictionary type obtained from the word dictionary address in the arguments (learningResult, preConfirmedResult).
NJ_ERR_DIC_BROKEN	Returned when one of the following situations occur when an uncompressed dictionary is the target in the arguments (learningResult, preConfirmedResult): • The queue data to be stored given by the queue ID was destroyed. • A reading string longer than NJ_MAX_USER_LEN was found in the user dictionary. • The link information when one word uses multiple keys was destroyed. Returned when one of the following situations occur when an uncompressed dictionary is the target in the arguments (learningResult, preConfirmedResult): • The queue ID was destroyed. • A reading string longer than NJ_MAX_USER_LEN or a candidate string longer than NM_MAX_CANDIDATE_LEN was found in the user dictionary. • A reading string longer than NJ_MAX_USER_LEN or a candidate string longer than NM_MAX_CANDIDATE_LEN was found in a dictionary other than the user dictionary. • The link information when one word uses multiple keys was destroyed. • Cannot get the location to add learning dictionary information, user dictionary information from the learning dictionary, or user dictionary specified in the argument (iwnn->dicSet). • The number of registered words given in the learning dictionary or user dictionary specified in the argument (iwnn->dicSet) is larger than the maximum number of registerable words. The queue data inside the learning dictionary or user dictionary specified in the argument (iwnn->dicSet) is corrupted.
NJ_ERR_READING_TOO_LONG	The reading string length exceeds NJ_MAX_LEN when the arguments (learningResult, preConfirmedResult) were generated by an operation other than morphological analysis.

Error Code	Description
NJ_ERR_PROTECTION_ERR	A function that operates on learning dictionaries has performed an operation on protected memory.
NJ_ERR_CANDIDATE_TOO_LONG	A value greater than NJ_MAX_RESULT_LEN was set for the candidate string length in the arguments (learningResult, preConfirmedResult).

9.13 Undo Learning Function (NjxUndo)

This function undoes learning information for the specified undo count.

It also clears pre-confirmation information.

Deleted learning information cannot be undone because old information in the learning dictionary is constantly being deleted as learning repeats.

Code 9-13 Undo Learning Function (NjxUndo) Declaration

```
s16 NjxUndo(
 IWNN_CLASS* iwnn,       // Analysis information class
 u16 undoCount                 // Undo count
)
```

Table 9-45 Undo Learning Function (NjxUndo) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
IN	u16 undoCount	Count to undo.

Table 9-46 Undo Learning Function (NjxUndo) Return Values

Return Value	Description
s16	Negative value: Error. Other: The undo count.

Table 9-47 Undo Learning Function (NjxUndo) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_DIC_NOT_FOUND	The dictionary specified by the argument (type) does not exist in the dictionary set specified by the argument (iwnn->dicSet).
NJ_ERR_PROTECTION_ERR	A function that operates on learning dictionaries has performed an operation on protected memory.
NJ_ERR_DIC_BROKEN	• Cannot get the location to add learning dictionary information, user dictionary information from the learning dictionary, or user dictionary specified in the argument (iwnn->dicSet). • The number of registered words given in the learning dictionary or user dictionary specified in the argument (iwnn->dicSet) is larger than the maximum number of registerable words. • The queue data inside the learning dictionary or user dictionary specified in the argument (iwnn->dicSet) is corrupted.

9.14 Search Word Function (NjxSearchWord)

This function configures a search cursor structure used to look up the specified reading string and get words.

To get the target word, use the dictionary search cursor structure to get the word using the get word function (**NjxGetWord**).

If getting all words through dictionary lookup from a user dictionary and learning dictionary, specify an empty string (“”) for the reading string of the word to be searched for using the dictionary search cursor structure and specify prefix match search for the search method.

If the return value indicates that no candidate matches the search conditions (0) or an error, undefined values may be stored in the dictionary search cursor (cursor).

Code 9-14 Search Word Function (NjxSearchWord) Declaration

```
s16 NjxSearchWord(
 IWNN_CURSOR* cursor,          // Dictionary search cursor structure
 const IWNN_CLASS* iwnn  // Analysis information class
)
```

Table 9-48 Search Word Function (NjxSearchWord) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CURSOR* cursor	Dictionary search cursor structure. The target word is searched for by setting a dictionary set structure, search conditions, a reading string, and other necessary information. For details on the structure, see section 8.5 Dictionary Search Cursor (IWNN_CURSOR). An error results if NULL is specified.
IN	const IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.

Table 9-49 Search Word Function (NjxSearchWord) Return Values

Return Value	Description
s16	Negative value: Error. 1: A candidate matching search conditions exists. 0: No candidates match search conditions.

Table 9-50 Search Word Function (NjxSearchWord) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_PARAM_CURSOR_NULL	NULL was specified in the argument (cursor).
NJ_ERR_PARAM_DIC_NULL	NULL was specified for the dictionary set in the argument (cursor).
NJ_ERR_PARAM_READING_NULL	NULL was specified for the reading in the argument (cursor).
NJ_ERR_READING_TOO_LONG	The reading string length in the argument (cursor) exceeds NJ_MAX_LEN.
NJ_ERR_PARAM_KANJI_NULL	A derived search was set for the search method in the argument (cursor) and NULL was specified for the candidate string.
NJ_ERR_CANDIDATE_TOO_LONG	The candidate string length in the argument (cursor) exceeds NJ_MAX_RESULT_LEN.
NJ_ERR_PARAM_OPERATION	An unsupported value was specified for the search method in the argument (cursor).
NJ_ERR_PARAM_MODE	An unsupported value was specified for the candidate retrieval order in the argument (cursor).
NJ_ERR_DIC_TYPE_INVALID	An unsupported dictionary was specified in the dictionary set for the argument (cursor).

Error Code	Description
NJ_ERR_DIC_BROKEN	• Cannot get the location to add learning dictionary information, user dictionary information from the learning dictionary, or user dictionary specified in the argument (iwnn->dicSet). • The number of registered words in the learning dictionary or user dictionary specified in the argument (iwnn->dicSet) is larger than the maximum number of registerable words. • The queue data inside the learning dictionary or user dictionary specified in the argument (iwnn->dicSet) is corrupted.

If you register a word or perform a learning operation after getting it with the dictionary search cursor (IWNN_CURSOR), the user dictionary or learning dictionary will be updated. So you will need to get the dictionary search cursor again.

When using the dictionaries included in this package, you cannot get all words by specifying an empty string (“”).

9.15 Get Word Function (NjxGetWord)

This function gets words one at a time, according to the specified search conditions, based on information in the dictionary search cursor structure obtained using the search word function.

The obtained word is stored in the process result buffer. If there is no candidate that matches the search conditions (return value 0) or an error occurs, undefined values may be stored in process results.

Code 9-15 Get Word Function (NjxGetWord) Declaration

```
s16 NjxGetWord(
 IWNN_CURSOR* cursor,          // Dictionary search cursor structure
 IWNN_RESULT* result,          // Process result buffer
 const IWNN_CLASS* iwnn  // Analysis information class
)
```

Table 9-51 Get Word Function (NjxGetWord) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CURSOR* cursor	Dictionary search cursor structure. Specifies the dictionary search cursor obtained by the get word function. An error results if NULL is specified.
OUT	IWNN_RESULT* result	Process result buffer. Enough memory to store one candidate must be allocated. An error results if NULL is specified.
IN	const IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.

Table 9-52 Get Word Function (NjxGetWord) Return Values

Return Value	Description
s16	Negative value: Error. 0: Cannot find matching candidate. Other: Normal exit.

Table 9-53 Get Word Function (NjxGetWord) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_PARAM_CURSOR_NULL	NULL was specified in the argument (cursor).
NJ_ERR_PARAM_RESULT_NULL	NULL was specified in the argument (result).
NJ_ERR_DIC_TYPE_INVALID	An unsupported dictionary was set for the dictionary set in the argument (cursor).
NJ_ERR_CANNOT_GET_QUEUE	• The word search position in the argument (cursor) was destroyed. • Queue data inside the uncompressed dictionary specified in the dictionary set in the argument (cursor) is corrupted.

If you register a word or perform a learning operation after getting it with the dictionary search cursor (IWNN_CURSOR), the user dictionary or learning dictionary will be updated. So you will need to get the dictionary search cursor again.

9.16 Add Word Function (NjxAddWord)

This function adds the specified information in the add word information structure to a learnable dictionary (user dictionary, learning dictionary, or pseudo dictionary).

If both a user dictionary and learning dictionary have been set in the dictionary set structure, the same words are registered in both dictionaries. However, if multiple learnable dictionaries of the same type have been set (multiple learning dictionaries or multiple user dictionaries), words are registered in the learnable dictionary set first. An error does not result, even if no learnable dictionaries are set in the dictionary set, but words are not registered.

Multiple instances of a word having the same reading string, candidate string, or part of speech group cannot be registered in a user dictionary.

If a user dictionary is specified as the destination to add a word, the associative learning flag is ignored.

Code 9-16 Add Word Function (NjxAddWord) Declaration

```
s16 NjxAddWord(
 IWNN_CLASS* iwnn,              // Analysis information class
 const IWNN_WORD_INFO* word,    // Word registration information structure
 u8 type,                 // Dictionary type
 u8 connect               // Associative learning flag
)
```

Table 9-54 Add Word Function (NjxAddWord) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CLASS* Iwnn	Analysis information class. An error results if NULL is specified.
IN	const IWNN_WORD_INFO* word	Word registration information structure. Stores the word information to be registered. An error results if NULL or an invalid value is specified in IWNN_WORD_INFO.
IN	u8 type	Dictionary type to be registered. 0: Register a user dictionary. 1: Register a learning dictionary. 2: Register a pseudo dictionary.
IN	u8 connect	Associative learning flag. This flag is enabled only when a learning dictionary is specified for the dictionary type in which words are to be registered. 0: Do not perform associative learning with the previously registered word. 1: Perform associative learning with the previously registered word.

Table 9-55 Add Word Function (NjxAddWord) Return Values

Return Value	Description
s16	Negative value: Error. (Including cases where the candidate has the same reading string and candidate string.) Other: Normal exit.

Table 9-56 Add Word Function (NjxAddWord) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_PARAM_DIC_NULL	NULL was set in the argument (iwnn->dicSet).
NJ_ERR_NO_RULE_DIC	A rule dictionary is not set in the argument (iwnn->dicSet).
NJ_ERR_DIC_TYPE_INVALID	An invalid value is set in the argument (type).
NJ_ERR_WORD_INFO_NULL	NULL was set in the argument (word).
NJ_ERR_USER_READING_INVALID	• A reading string equal to or greater than NJ_MAX_LEN was specified for a word to be registered in a learning dictionary. • A reading string equal to or greater than NJ_MAX_USER_LEN was specified for a word to be registered in a user dictionary.
NJ_ERR_USER_CANDIDATE_INVALID	• A candidate string equal to or greater than NJ_MAX_RESULT_LEN was specified for a word to be registered in a learning dictionary. • A candidate string equal to or greater than NJ_MAX_USER_CANDIDATE_LEN was specified for a word to be registered in a user dictionary.
NJ_ERR_PARTS_OF_SPEECH_GROUP_INVALID	An unsupported value was set for the part of speech group in the argument (word).
NJ_ERR_DIC_NOT_FOUND	The dictionary specified by the argument (type) does not exist in the dictionary set specified by the argument (iwnn->dicSet).
NJ_ERR_USER_DIC_FULL	There was a specification to add a word to the user dictionary, but the user dictionary already contains the maximum number of words allowed.
NJ_ERR_DIC_BROKEN	• Cannot get the location to add learning dictionary information, user dictionary information from the learning dictionary, or user dictionary specified in the argument (iwnn->dicSet). • The number of registered words given in the learning dictionary or user dictionary specified in the argument (iwnn->dicSet) is larger than the maximum number of registerable words. • The queue data inside the learning dictionary or user dictionary specified in the argument (iwnn->dicSet) is corrupted.

Error Code	Description
NJ_ERR_SAME_WORD	There was a specification to add a word to the user dictionary, but the same word has already been registered.
NJ_ERR_PROTECTION_ERR	A function that operates on learning dictionaries has performed an operation on protected memory.
NJ_ERR_INVALID_FLAG	A value greater than 1 was specified in the argument (connect).

9.17 Delete Word Function (NjxDeleteWord)

This function deletes words using the following search results.

- Search results obtained from getting a word from a learnable dictionary (user dictionary or learning dictionary).
- Search results obtained from getting a word from a pseudo dictionary.

An error results if process results other than those listed above are specified. An error also results if a process result for anything other than dictionary lookup is specified.

Code 9-17 Delete Word Function (NjxDeleteWord) Declaration

```
s16 NjxDeleteWord(
 IWNN_CLASS* iwnn,       // Analysis information class
 const IWNN_RESULT* result     // Process result
)
```

Table 9-57 Delete Word Function (NjxDeleteWord) Arguments

Input/ Output	Argument	Description
IN	IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
IN	const IWNN_RESULT* result	Process result. An error results if NULL is specified.

Table 9-58 Delete Word Function (NjxDeleteWord) Return Values

Return Value	Description
s16	Negative value: Error. Other values indicate normal termination.

Table 9-59 Delete Word Function (NjxDeleteWord) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_PARAM_RESULT_NULL	NULL was specified for the argument (result).
NJ_ERR_DIC_HANDLE_NULL	The dictionary handle in the argument (result) is NULL.
NJ_ERR_DIC_TYPE_INVALID	The dictionary handle in the argument (result) is not that of a learning dictionary or user dictionary.
NJ_ERR_INVALID_RESULT	A process result (result) other than one generated by a dictionary lookup was specified.
NJ_ERR_WORD_NOT_FOUND	The word specified by the argument (result) has already been deleted.
NJ_ERR_PROTECTION_ERR	A function that operates on learning dictionaries has performed an operation on protected memory.
NJ_ERR_DIC_BROKEN	• Cannot get the location to add user dictionary information from the user dictionary specified in the argument (iwnn->dicSet). • The number of registered words given in the user dictionary specified in the argument (iwnn->dicSet) is larger than the maximum number of registerable words. • The queue data inside the user dictionary specified in the argument (iwnn->dicSet) is corrupted.

9.18 Split Word Function (MmxSplitWord)

This function splits the input string into phrase units, and returns process results for the number of phrases that result.

If the return value indicates an error, an undefined value may be stored in process results or for the analysis end string length.

This API uses the following dictionaries inside the specified dictionary set (iwnn->dicSet).

- Integrated dictionary.
- User dictionary.
- Ancillary word dictionary.
- Rule dictionary.
- Compressed custom dictionary.
- Uncompressed custom dictionary.
- Single-kanji dictionary.

However, do not specify a custom dictionary in which parts of speech are not exactly assigned in the dictionary set.

Code 9-18 Split Word Function (MmxSplitWord) Declaration

```
s16 MmxSplitWord(
 IWNN_CLASS* iwnn,       // Analysis information class
 IWNN_RESULT* result,          // Process result storage buffer
 u8* processLen,         // Analysis end string length
 const wchar_t* input          // Input string
)
```

Table 9-60 Split Word Function (MmxSplitWord) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
OUT	IWNN_RESULT* result	Process result storage buffer. Be sure to prepare a buffer for storing a result of the size given by the maximum morphological analysis string length (MM_MAX_MORPHOLIZE_LEN). An error results if NULL is specified.
OUT	u8* processLen	The analysis end position (string length) after splitting words. An error results if NULL is specified.
IN	const wchar_t* input	The input string to be split. An error results if NULL is specified. 0 is returned if an empty string (“”) is specified. Be sure to add a terminator at the end of the string. Because this string is used (overwritten) internally by iWnn during delimited input, do not change its contents until operations are complete.

Table 9-61 Split Word Function (MmxSplitWord) Return Values

Return Value	Description
s16	Negative value: Error. 0: An empty string is specified for the input string. Other: Normal exit.

Table 9-62 Split Word Function (MmxSplitWord) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_PARAM_READING_NULL	A NULL pointer was specified in the argument (input).
NJ_ERR_PARAM_PROCESS_LEN_NULL	A NULL pointer was specified in the argument (processLen).
NJ_ERR_PARAM_RESULT_NULL	A NULL pointer was specified in the argument (result).
NJ_ERR_DIC_BROKEN	• Cannot get the location to add user dictionary information from the user dictionary specified in the argument (iwnn->dicSet). • The number of registered words given in the user dictionary specified in the argument (iwnn->dicSet) is larger than the maximum number of registerable words. • The queue data inside the user dictionary specified in the argument (iwnn->dicSet) is corrupted.
NJ_ERR_NO_RULE_DIC	A rule dictionary is not set in the argument (iwnn->dicSet).
NJ_ERR_NO_PARTS_OF_SPEECH	The required part of speech information could not be obtained from the rule dictionary.

Get Split Word Position

The length of the input string in the process result structure (IWNN_RESULT), obtained by the split word function (**MmxSplitWord**), can in turn be obtained using the MM_GET_CANDIDATE_LEN macro. In addition, you can get the boundary between independent words and ancillary words in the process result structure (IWNN_RESULT), using the MM_GET_STEM_LEN macro.

Table 9-63 Macros for String Length or Word Boundary

Macro	Description
MM_GET_STEM_LEN(IWNN_RESULT *)	Gets the length of the independent word part of the candidate string.
MM_GET_CANDIDATE_LEN(IWNN_RESULT *)	Gets the string length of the candidate string.

Depending on the specified text, phrases consisting of ancillary words only (such as “ですね” and “さ”) may be returned. The MM_GET_STEM_LEN returns 0 in this case.

9.19 Get Part of Speech Group Function (MmxGetPartsOfSpeech)

This function returns a part of a speech group of the independent words in the process results obtained using the split word function.

Do not specify results obtained using any function other than the delimited input function.

Table 9-64 Part of Speech Groups

Part of Speech Group	Description	Example
Part of speech group. (MM_PARTS_OF_SPEECH_GROUP_NOUN)	Part of speech related to nouns such as regular nouns, fixed names, people's names, place names, and so forth.	子供が
Pseudo group. (MM_PARTS_OF_SPEECH_GROUP_PSEUDO)	Pseudo words or words not found in the dictionary.	ABCDEFG です
Other. (MM_PARTS_OF_SPEECH_GROUP_OTHER)	Parts of speech other than those above.	動く

Code 9-19 Get Part of Speech Group Function (MmxGetPartsOfSpeech) Declaration

```
s16 MmxGetPartsOfSpeech(
 IWNN_CLASS* iwnn,                    // Analysis information class
 const IWNN_RESULT* result            // Process result
)
```

Table 9-65 Get Part of Speech Group Function (MmxGetPartsOfSpeech) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
IN	const IWNN_RESULT* result	Split word process result. An error results if NULL is specified.

Table 9-66 Get Part of Speech Group Function (MmxGetPartsOfSpeech) Return Values

Return Value	Description
s16	Obtained part of speech group. Negative value: Error.

Table 9-67 Get Part of Speech Group Function (MmxGetPartsOfSpeech) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_PARAM_RESULT_NULL	A NULL pointer was specified in the argument (result).
NJ_ERR_INVALID_RESULT	The result of a process other than morphological analysis was specified in the argument (result).
NJ_ERR_NO_RULE_DIC	A rule dictionary is not set in the argument (iwnn->dicSet).

9.20 Get Reading String for Morphological Analysis Function (MmxGetReading)

This function returns reading strings for words having the same notation as the specified process results. One candidate is returned at a time, in order of frequency. Only readings for specified result phrases (independent words, or independent words and ancillary words) are returned for the reading string. The reading string length of independent words is stored in stemLen.

Be sure not to specify process results obtained by functions other than the split word function.

Invalid values may be stored in the reading string, or reading string for independent words in the following cases.

- When an error has occurred.
- When the reading string length exceeds NJ_MAX_LEN.
- When the reading string exceeds the reading string buffer size.
- When the candidate string exceeds NJ_MAX_RESULT_LEN.

This API uses the following dictionaries inside the specified dictionary set (iwnn->dicSet).

- Integrated dictionary.
- User dictionary.
- Ancillary word dictionary.
- Rule dictionary.
- Compressed custom dictionary.
- Uncompressed custom dictionary.
- Single-kanji dictionary.

This function cannot be used on an uncompressed custom dictionary created through conversion, using the change dictionary type function.

Do not specify a custom dictionary for which parts of speech are not exactly assigned in the dictionary set.

Code 9-20 Get Reading String for Morphological Analysis Function (MmxGetReading) Declaration

```
s16 MmxGetReading(
 IWNN_CLASS* iwnn,       // Analysis information class
 wchar_t* reading,       // Reading string
 u8* stemLen,                  // Reading string length of independent word
 const IWNN_RESULT* result,    // Process result
 u16 readingSize         // Reading string buffer size
)
```

Table 9-68 Get Reading String for Morphological Analysis Function (MmxGetReading) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
OUT	wchar_t* reading	Reading string storage buffer. Prepare a wchar_t array of size NJ_MAX_LEN+NJ_TERM_SIZE.
OUT	u8* stemLen	Reading string length of independent word. The reading string length is not stored if a NULL pointer is specified.
IN	const IWNN_RESULT* result	Split word process result. Specifies process results obtained from the split word function (**MmxSplitWord**). If NULL is specified, split word process results when getting the previous reading are used, and the next candidate is returned. However, an error is returned if NULL was specified when initially getting the reading.
IN	u16 readingSize	Reading string byte size. Enough memory to include a terminator must be allocated.

Table 9-69 Get Reading String for Morphological Analysis Function (MmxGetReading) Return Values

Return Value	Description
s16	The length of the reading string stored in the reading string storage buffer. 0: Either there was no reading string corresponding to the specified process result, or it could not be obtained/processed. Negative value: Error.

Table 9-70 Get Reading String for Morphological Analysis Function (`MmxGetReading`) Errors

Error Code	Description
`NJ_ERR_PARAM_ENVIRONMENT_NULL`	A NULL pointer was specified in the argument (`iwnn`).
`NJ_ERR_BUFFER_NOT_ENOUGH`	• A NULL pointer was specified in the argument (`reading`). • 0 was specified in the argument (`readingSize`). • The buffer size is less than `(NJ_MAX_LEN+NJ_TERM_SIZE)*sizeof(wchar_t)`.
`NJ_ERR_INVALID_RESULT`	Process results other than those of morphological analysis were specified in the argument (`result`).
`NJ_ERR_DIC_BROKEN`	• Cannot get the location to add user dictionary information from the user dictionary specified in the argument (`iwnn->dicSet`). • The number of registered words given in the user dictionary specified in the argument (`iwnn->dicSet`) is larger than the maximum number of registerable words. • The queue data inside the user dictionary specified in the argument (`iwnn->dicSet`) is corrupted.
`NJ_ERR_NO_RULE_DIC`	A rule dictionary is not set in the argument (`iwnn->dicSet`).
`NJ_ERR_NO_PARTS_OF_SPEECH`	Cannot get required part of speech information from the rule dictionary.
`NJ_ERR_DIC_TYPE_INVALID`	An unsupported dictionary was specified in the argument (`iwnn->dicSet`).
`NJ_ERR_NO_CANDIDATE_LIST`	NULL was specified in the argument (`result`) for the first call.

9.21 Learn by Morphological Analysis Function (`MmxSelect`)

This function adds words to a learning dictionary, based on the specified reading string and the process results obtained by the split word operation.

You can limit candidate character strings, to be learned, to independent words included in process results. Be sure to specify only strings corresponding to an independent word, for the reading string.

Do not specify results obtained using any function other than the split word function.

Code 9-21 Learn by Morphological Analysis Function (`MmxSelect`) Declaration

```
s16 MmxSelect(
 IWNN_CLASS* iwnn,       // Analysis information class
 const IWNN_RESULT* result,    // Process results
 const wchar_t* reading, // Reading string
 u8 independentFlag,           // Independent word specification flag
 u8 connect              // Associative learning flag
)
```

Table 9-71 Learn by Morphological Analysis Function (MmxSelect) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
IN	const IWNN_RESULT* result	Split word process results. An error results if NULL is specified.
IN	const wchar_t* reading	Reading string. Specify a string having a length of up to NJ_MAX_LEN+NJ_TERM_SIZE. An error results if NULL or an empty string (“”) is specified.
IN	u8 independentFlag	Independent word specification flag. 0: Learn process results by phrase. 1: Learn only independent words in process results. Other: Results in an error.
IN	u8 connect	Associative learning flag. Specifies whether to perform associative learning with pre-confirmed candidates. 0: Do not perform associative learning. 1: Perform associative learning. Other: Results in an error.

Table 9-72 Learn by Morphological Analysis Function (MmxSelect) Return Values

Return Value	Description
s16	Negative value: Error. Other: Normal exit.

Table 9-73 Learn by Morphological Analysis Function (MmxSelect) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_DIC_NOT_FOUND	A learning dictionary is not set in the argument (iwnn->dicSet).
NJ_ERR_PARAM_RESULT_NULL	A NULL pointer was specified in the argument (result).
NJ_ERR_PARAM_READING_NULL	• A NULL pointer was specified in the argument (reading). • An empty string was specified in the argument (reading).
NJ_ERR_READING_TOO_LONG	A string length longer than NJ_MAX_LEN was specified in the argument (reading).

Error Code	Description
NJ_ERR_INVALID_FLAG	• A value other than 0 or 1 was specified for the argument (independentFlag). • A value other than 0 or 1 was set in the argument (connect).
NJ_ERR_INVALID_RESULT	• Unsupported code was set for operations on the argument (result). • Process results other than those of morphological analysis were specified in the arguments (result). • A value of 1 was set in the argument (independentFlag), but the arguments (result) represents phrase information for ancillary words only. • The reading string length of the argument (result) is 0.
NJ_ERR_DIC_BROKEN	Returned when one of the following situations occur, when an uncompressed dictionary is the target in the arguments (learningResult and preConfirmedResult): • The queue data to be stored given by the queue ID is destroyed. • A reading string longer than NJ_MAX_USER_LEN is found in the user dictionary. • The link information, when one word uses multiple keys, is destroyed. Returned when one of the following situations occur, when an uncompressed dictionary is the target in the arguments (learningResult and preConfirmedResult): • The queue ID (result->word.stem.location.current) is destroyed. • A reading string longer than NJ_MAX_USER_LEN, or a candidate string longer than NM_MAX_CANDIDATE_LEN is found in the user dictionary. • A reading string longer than NJ_MAX_USER_LEN, or a candidate string longer than NM_MAX_CANDIDATE_LEN is found in a dictionary, other than a user dictionary. • The link information, when one word uses multiple keys, is destroyed. Cannot get the location to add user dictionary information from the user dictionary specified in the argument (iwnn->dicSet). The number of registered words given in the user dictionary specified in the argument (iwnn->dicSet) is larger than the maximum number of registerable words. The queue data inside the user dictionary specified in the argument (iwnn->dicSet) is corrupted.
NJ_ERR_CANDIDATE_TOO_LONG	A value greater than NJ_MAX_RESULT_LEN was specified in the argument (result).
NJ_ERR_DIC_NOT_FOUND	A learning dictionary is not set in the argument (iwnn->dicSet).
NJ_ERR_PROTECTION_ERR	A function that operates on learning dictionaries has performed an operation on protected memory.
NJ_ERR_NO_RULE_DIC	A rule dictionary is not set in the argument (iwnn->dicSet).

9.22 Set Options Function (NjxSetOption)

This function sets operational parameters for iWnn.

Do not make settings during prediction, kana-kanji conversion, or morphological analysis.

For details on which options can be set, see 8.8 Option Settings (IWNN_OPTION).

Code 9-22 Option Settings Function (NjxSetOption) Declaration

```
s16 NjxSetOption(
 IWNN_CLASS* iwnn,              // Analysis information class
 const IWNN_OPTION* option      // Option settings
)
```

Table 9-74 Option Settings Function (NjxSetOption) Arguments

Input/ Output	Argument	Description
IN	IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
IN	const IWNN_OPTION* option	Option settings. The specified setting is stored in the analysis information class. The default value is set in the analysis information class, if NULL is specified. **Note:** Option setting (default value). Number of candidates to automatically start multiple phrase conversion: NJ_MAX_CANDIDATE.

Table 9-75 Option Settings Function (NjxSetOption) Return Values

Return Value	Description
s16	Negative value: Error. Other: Normal exit.

Table 9-76 Option Settings Function (NjxSetOption) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).

9.23 Set State Function (NjxSetState)

This function sets state parameters for situational adaptive prediction.

Do not make settings during prediction, kana-kanji conversion, or morphological analysis.

Code 9-23 Set State Function (NjxSetState) Declaration

```
s16 NjxSetState(
 IWNN_CLASS* iwnn,       // Analysis information class
 const IWNN_STATE* state // State setting
)
```

Table 9-77 Set State Function (NjxSetState) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
IN	const IWNN_STATE* state	State setting. If NULL is specified, the default value (0) is set for all categories. For details on settings, see section 8.10 State Setting (IWNN_STATE).

Table 9-78 Set State Function (NjxSetState) Return Values

Return Value	Description
s16	Negative value: Error. Other: Normal exit.

Table 9-79 Set State Function (NjxSetState) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_PARAM_INVALID_STATE	A setting outside the allowable range was made for the bias value in the argument (state).

9.24 Get State Setting Function (NjxGetState)

This function returns the current state setting parameter, maintained internally by iWnn.

Code 9-24 Get State Setting Function (NjxGetState) Declaration

```
s16 NjxGetState(
 IWNN_CLASS* iwnn,             // Analysis information class
 IWNN_STATE* state       // State setting
)
```

Table 9-80 Get State Setting Function (NjxGetState) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
OUT	IWNN_STATE* state	状況設定 Buffer storing the state setting. An error results if NULL is specified. For details on settings, see section 8.10 State Setting (IWNN_STATE).

Table 9-81 Get State Setting Function (NjxGetState) Return Values

Return Value	Description
s16	Negative value: Error. Other: Normal exit.

Table 9-82 Get State Setting Function (NjxGetState) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_PARAM_STATE_NULL	A NULL pointer was specified in the argument (state).

9.25 Get Word Information Function (NjxGetWordInfo)

This function gets information used to add words from the process result structure.

This function is used when you want to get word information (IWNN_WORD_INFO), required by the add word function (NjxAddWord), from the results (IWNN_RESULT) of the search word, prediction, or conversion processes.

Code 9-25 Get Word Information Function (NjxGetWordInfo) Declaration

```
s16 NjxGetWordInfo(
 IWNN_WORD_INFO* info,          // Word information
 const IWNN_CLASS* iwnn,              // Analysis information class
 const IWNN_RESULT* result            // Process result structure
)
```

Table 9-83 Get Word Information Function (`NjxGetWordInfo`) Arguments

Input/ Output	Argument	Description
`OUT`	`IWNN_WORD_INFO* info`	Word information. Stores information required to register words extracted from the process result structure (`result`).
`IN`	`const IWNN_CLASS* iwnn`	Analysis information class. An error results if `NULL` is specified.
`IN`	`const IWNN_RESULT* result`	Process result structure. Specifies results obtained from search, conversion, and other processes.

Table 9-84 Get Word Information Function (`NjxGetWordInfo`) Return Values

Return Value	Description
s16	Negative value: Error. Other: Normal exit.

Table 9-85 Get Word Information Function (NjxGetWordInfo) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_PARAM_RESULT_NULL	A NULL pointer was specified in the argument (result).
NJ_ERR_WORD_INFO_NULL	A NULL pointer was specified in the argument (info).
NJ_ERR_INVALID_RESULT	• Unsupported code was set for operations on the argument (result). • The dictionary to get candidates from did not return a reading string for reverse lookup results.
NJ_ERR_DIC_TYPE_INVALID	Unsupported code was set for the dictionary type obtained from the single-word dictionary address given by the argument (result).
NJ_ERR_DIC_BROKEN	Returned when one of the following situations occur, when an uncompressed dictionary is the target in the argument (result): • The queue data to be stored, given by the queue ID, is destroyed. • A reading string longer than NJ_MAX_USER_LEN is found in the user dictionary. • The link information, when one word uses multiple keys, is destroyed.
NJ_ERR_READING_TOO_LONG	The reading string length in the argument (result) exceeds NJ_MAX_LEN.
NJ_ERR_CANDIDATE_TOO_LONG	The candidate string length in the argument (result) exceeds NJ_MAX_RESULT_LEN.

9.26 Get No Conversion Candidates Function (NjxGetStrokeWord)

This function generates information for candidates that have the same notation as the reading.

If you directly confirm an input string without performing conversion operations (NjxConversion, NjxAnalyze, or NjxAllCandidates), better relationship prediction candidates can be obtained by creating no conversion candidates with this function and performing word learning.

In order to handle phrase conversion internally, the same dictionary used at time of phrase conversion is required.

Code 9-26 Get No Conversion Candidate Function (NjxGetStrokeWord) Declaration

```
s16 NjxGetStrokeWord(
 IWNN_CLASS* iwnn,              // Analysis information class
 IWNN_RESULT* result,                 // Conversion result storage buffer
 const wchar_t* reading         // reading string
)
```

Table 9-86 Get No Conversion Candidate Function (NjxGetStrokeWord) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
OUT	IWNN_RESULT* Result	Conversion result storage buffer. Enough memory to store one phrase must be allocated.
IN	const wchar_t* reading	Reading string to perform no conversion. Be sure to add a terminator to the end of the string. Do not change this value until conversion operations have ended, because this string region is used (overwritten) internally by iWnn during kana-kanji conversion.

Table 9-87 Get No Conversion Candidate Function (NjxGetStrokeWord) Return Values

Return Value	Description
s16	1: Number of phrases stored in the conversion result storage buffer (=1). Negative value: Error.

Table 9-88 Get No Conversion Candidate Function (NjxGetStrokeWord) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_DIC_BROKEN	• Cannot get the location to add learning dictionary information or user dictionary information from the learning dictionary or user dictionary specified in the argument (iwnn->dicSet). • The number of registered words given in the learning dictionary or user dictionary specified in the argument (iwnn->dicSet) is larger than the maximum number of registerable words. • The queue data inside the learning dictionary or user dictionary specified in the argument (iwnn->dicSet) is corrupted.
NJ_ERR_PARAM_RESULT_NULL	A NULL pointer was specified in the argument (result).
NJ_ERR_PARAM_READING_NULL	A NULL pointer was specified in the argument (reading).
NJ_ERR_PARAM_READING_SIZE	An empty character string or reading string longer than NJ_MAX_LEN was specified in the argument (reading).
NJ_ERR_NO_RULE_DIC	A rule dictionary is not set in the argument (iwnn->dicSet).
NJ_ERR_NO_PARTS_OF_SPEECH	Cannot get the required part of speech information from the rule dictionary.
NJ_ERR_DIC_TYPE_INVALID	An unsupported dictionary was specified in the argument (iwnn->dicSet).

9.27 Merge Candidate Lists Function (NjxMergeWordList)

This function creates a single candidate list by combining multiple candidate lists.

A list with duplications removed can be created by gathering together words that have the same notation.

In addition, you can easily change the order of candidates in the list, because you can select the merge method to use for candidates with the same notation.

This function is used, for example, when creating a candidate list where pseudo candidates must be located at the end.

If an abnormal termination occurs, the contents of the candidate list (wordList) are undefined.

Code 9-27 Merge Candidate Lists Function (NjxMergeWordList) Declaration

```
s16 NjxMergeWordList(
 IWNN_MERGE_RESULT* wordList,  // Candidate list
 s32 listMax,                        // Maximum number of storable
                                        candidates
 u32 mode,                     // Merge method
 const IWNN_CLASS* iwnn,             // Analysis information class
 const IWNN_RESULT* result,          // Candidate array to be merged
 u32 num                       // Number of candidates to be merged
)
```

Table 9-89 Merge Candidate Lists Function (NjxMergeWordList) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_MERGE_RESULT* wordList	Candidate list. Memory region for storing the list order. Because pointers to iwnn and result are maintained internally, the memory pointed to by iWnn and result must not be written to or freed until use of the candidate list has ended.
IN	s32 listMax	Maximum number of storable candidates. Represents the maximum number of candidates that can be stored in wordList. A specification greater than NJ_MAX_CANDIDATE cannot be made.
IN	u32 mode	Merge method. NJ_MERGE_INIT: Clears the list and then adds candidates. (Use before creating the first candidate list.) NJ_MERGE_NOT_EXIST: Adds only candidates that do not exist in the candidate list, to the end of the list. Any candidates beyond the maximum number of storable candidates are not added. NJ_MERGE_NOT_EXIST_FORCE: Adds only candidates that do not exist in the candidate list, to the end of the list. Any candidates beyond the maximum number of storable candidates are added, by deleting candidates that already exist in the candidate list from the end of the list. NJ_MERGE_FORCE: Adds all candidates in result to the end of the list. Candidates that do not exist in the candidate list are moved to the end. Any candidates beyond the maximum number of storable candidates are added by deleting candidates that already exist in the candidate list from the end of the list.
IN	const IWNN_CLASS* iwnn	Analysis information class. Used to get word information from result.
IN	const IWNN_RESULT* result	Candidate array to be merged. The candidate array to be added to the candidate list. This array must not include duplicate candidates.
IN	u32 num	Number of candidates to be merged. The number of candidates to be merged with the candidate list. If a value larger than list_num is specified, operations continue as if list_num was specified.

Table 9-90 Merge Candidate Lists Function (NjxMergeWordList) Return Values

Return Value	Description
s16	The number of candidates stored in the candidate list. Negative value: Error.

Table 9-91 Merge Candidate Lists Function (NjxMergeWordList) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_PARAM_RESULT_NULL	A NULL pointer was specified in the argument (wordList, result).
NJ_ERR_PARAM_MODE	An invalid value was specified in the argument (mode).
NJ_ERR_BUFFER_NOT_ENOUGH	A value greater than NJ_MAX_CANDIDATE was specified in the argument (listMax).
NJ_ERR_INVALID_RESULT	Candidate information specified in the argument (result) is corrupted. Or, candidate information registered in the argument (wordlist) is corrupted.

9.28 Manage Learning Dictionary Function (NjxManageLearnDic)

This function performs various operations on learning dictionaries. It is used when merging learning dictionaries.

Code 9-28 Manage Learning Dictionary Operation Function (NjxManageLearnDic) Declaration

```
s16 NjxManageLearnDic(
 const IWNN_CLASS* iwnn, // Analysis information class
 u32 operation                 // Operation type
)
```

Table 9-92 Manage Learning Dictionary Operation Function (`NjxManageLearnDic`) Arguments

Input/ Output	Argument	Description
IN	const IWNN_CLASS* iwnn	Analysis information class. The learning dictionary registered in the dictionary set represented by this structure is the target of operations.
IN	u32 operation	Operation type. Specifies the operation to be performed on the learning dictionary. NJ_MLD_OPERATION_COMMIT: Records (commits) word registration range at that time. The commit range is protected from being written by functions, such as the learning function. NJ_MLD_OPERATION_COMMIT_TO_TOP: Words added after the previous commit position are moved ahead of the previous commit range and re-committed. NJ_MLD_OPERATION_COMMIT_CANCEL: Deletes words added after the previous commit position. NJ_MLD_OPERATION_GET_SPACE: Gets the maximum number of candidates that can be registered outside the commit range.

Table 9-93 Manage Learning Dictionary Operation Function (`NjxManageLearnDic`) Return Values

Return Value	Description
s16	Negative value: Error. The return values for normal termination vary depending on the operation type. NJ_MLD_OPERATION_GET_SPACE: The maximum number of candidates that can be registered. Other cases: 0

Table 9-94 Manage Learning Dictionary Operation Function (NjxManageLearnDic) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_PARAM_MODE	An illegal value was specified in the argument (operation).
NJ_ERR_DIC_BROKEN	Dictionary corruption detected.
NJ_ERR_DIC_NOT_FOUND	The learning dictionary specified by the argument (iwnn->dicSet) does not exist in the dictionary set.

The order (oldest to newest) in which words were added (learned) by the learning dictionary is recorded. The newer the candidate, the higher the priority level (frequency level). When adding words from another learning dictionary to the learning dictionary currently being used, you can treat added words as candidates older than (having lower priority than) words already registered by using the NJ_MLD_OPERATION_COMMIT_TO_TOP operation.

Table 9-95 Learning Dictionary Operations

Learning Dictionary Contents		Operation:
Old →	→ New	
Already Learned Area (A)	(Empty)	NJ_MLD_OPERATION_COMMIT Commits the Already Learned Area.
Already Learned Area (A)	Added Candidates (B)	Registers candidates in a different learning dictionary using the add word or other function.
Added Candidates (B)	Already Learned Area (A)	NJ_MLD_OPERATION_COMMIT_TO_TOP Moves added Candidates (B) in front of the Already Learned Area (A). Both areas (B) and (A) are redefined as the committed area at this time.

The committed range is protected from being over-written by the add/delete word functions and the learning function. If the available memory outside the commit range is insufficient, the add word function and learning function return an error.

The commit range specification can be canceled by executing NJ_MLD_OPERATION_COMMIT_CANCEL or executing the initialization API (NjxInit).

9.29 Get Additional Information String Function (NjxGetAdditionalInfo)

This function is used to retrieve the additional information string of candidates from the process result structure.

If the return value indicates an error, undefined values may be stored in the buffer for storing the additional information string (buf).

Code 9-29 Get Additional Information String Function (NjxGetAdditionalInfo) Declaration

```
s32 NjxGetAdditionalInfo (
 wchar_t* buf,                  // Buffer for storing the obtained
additional
                                  information string
 const IWNN_CLASS* iwnn,        // Analysis information class
 const IWNN_RESULT* result,     // Process result structure
 s8 index,                 // Index of the additional information to be
                                  obtained
 u32 size                  // Buffer size
)
```

Table 9-96 Get Additional Information String Function (NjxGetAdditionalInfo) Arguments

Input/ Output	Argument	Description
OUT	wchar_t* buf	Buffer for storing the obtained additional information string. Allocates a buffer and specifies a pointer. For process results that include learnable additional information, allocate a wchar_t array having the size given by NJ_MAX_ADDITIONAL_LEN + NJ_TERM_SIZE. For processing results that include unlearnable additional information, contact Nintendo (support@noa.com) regarding the required array size. A maximum of 65,536 Bytes may be required. (For information on whether additional information can be learned, see section 8.2 Dictionary Sets (IWNN_DIC_SET, IWNN_DIC_INFO, and IWNN_FLASH_DIC_INFO).) An error results if NULL is specified.
IN	const IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
IN	const IWNN_RESULT* result	Process result structure. An error results if NULL is specified.
IN	s8 index	Index of the additional information to be obtained. Specifies the index of the additional information to be obtained. Specify a value from 0 to the maximum number of additional information entries that can be mounted (NJ_MAX_ADDITIONAL_INFO).
IN	u32 size	Byte size of the buffer storing the additional information string. Specify the size, including the string terminator, in terms of number of bytes.

Table 9-97 Get Additional Information String Function (NjxGetAdditionalInfo) Return Values

Return Value	Description
s32	String length of the obtained string. (Not including the terminator.) Negative value: Error.

Table 9-98 Get Additional Information String Function (NjxGetAdditionalInfo) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_PARAM_RESULT_NULL	A NULL pointer was specified in the argument (result).
NJ_ERR_PARAM_INDEX_INVALID	The maximum number of additional information entries is mounted in the argument (index). A number of entries greater than NJ_MAX_ADDITIONAL_INFO was specified.
NJ_ERR_BUFFER_NOT_ENOUGH	A NULL pointer was specified in the argument (buf). 0 was specified in the argument (size). The additional information string length in the argument (result) is longer than the argument (size).
NJ_ERR_INVALID_RESULT	Unsupported code was set for operations on the argument (result).

9.30 Check Additional Information Function (NjxCheckAdditionalInfo)

This function checks whether the specified additional information region corresponds to the dictionary handle.

If additional information is included in the dictionary set, this function is used to check if the additional information corresponds correctly with the dictionary handle. In the case of user dictionaries and learning dictionaries, this function performs a check of additional information equivalent to that made by the check dictionary function (**NjxCheckDic**).

If an error is returned for the check made by this function, it indicates the dictionary handle does not correspond correctly with additional information, so be sure not to use them as a pair.

Code 9-30 Check Additional Information Function (NjxCheckAdditionalInfo) Declaration

```
s16 NjxCheckAdditionalInfo(
 IWNN_CLASS* iwnn,             // Analysis information class
 u8 dicType,                   // Dictionary handle type
 const IWNN_DIC_HANDLE handle, // Dictionary handle
```

```
 const void* additionalInfo,   // Additional information region
 u32 size                      // Additional information region size
)
```

Table 9-99 Check Additional Information Function (NjxCheckAdditionalInfo) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
IN	u8 dicType	Dictionary handle type of the dictionary handle to be checked.
IN	const IWNN_DIC_HANDLE handle	Dictionary handle. Stores the handle of the dictionary corresponding to the additional information region to be specified. An error results if NULL is specified. If a dictionary handle that does not correspond to the additional information region is specified, this function returns 0 (the check is not necessary).
IN	const void* additionalInfo	Additional information region. Stores the start pointer to the additional information region. An error results if NULL is specified.
IN	u32 size	Additional information region size. Stores the size of the region specified in additionalInfo.

Table 9-100 Check Additional Information Function (NjxCheckAdditionalInfo) Return Values

Return Value	Description
s16	Negative value: Error. 1: Normal exit. 0: The check is not necessary.

Table 9-101 Check Additional Information Function (NjxCheckAdditionalInfo) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_PARAM_DIC_NULL	A NULL pointer was specified in the argument (handle).
NJ_ERR_PARAM_ADD_INFO_NULL	A NULL pointer was specified in the argument (additionalInfo).
NJ_ERR_PARAM_ ADD_INFO _INVALID_SIZE	• 0 was specified in the argument (size). • Cannot verify the integrity of the size maintained in the argument (additionalInfo) with the argument (size).
NJ_ERR_BUFFER_NOT_ENOUGH	A value smaller than the size required for the additional information region was specified in the argument (size).
NJ_ERR_FORMAT_INVALID	• An invalid area of memory was specified in the argument (additionalInfo). • Learnable additional information was specified in the argument (additionalInfo), and the maximum additional information size stored in the argument (additionalInfo) is greater than NJ_MAX_ADDITIONAL_LEN.

9.31 Get FLASH Dictionary Cache Size Function (NjxGetFlashDicCacheSize)

This function gets the cache size required by the FLASH dictionary.

Code 9-31 Get FLASH Dictionary Cache Size Function (NjxGetFlashDicCacheSize) Declaration

```
s16 NjxGetFlashDicCacheSize(
 const IWNN_CLASS* iwnn,              // Analysis information class
 const IWNN_FILE* fileStream          // Dictionary file pointer
)
```

Table 9-102 Get FLASH Dictionary Cache Size Function (NjxGetFlashDicCacheSize) Arguments

Input/ Output	Argument	Description
IN	const IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
IN	const IWNN_FILE* fileStream	Dictionary file pointer. Specifies the dictionary file pointer for which to get the cache size. An error results if NULL is specified.

Table 9-103 Get FLASH Dictionary Cache Size Function (NjxGetFlashDicCacheSize) Return Values

Return Value	Description
s16	Negative value: Error. 0 or higher: The cache size.

Table 9-104 Get FLASH Dictionary Cache Size Function (NjxGetFlashDicCacheSize) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_PARAM_STREAM_NULL	A NULL pointer was specified in the argument (fileStream).
NJ_ERR_PARAM_TYPE_INVALID	A dictionary file pointer, other than that for a FLASH dictionary, was specified.
NJ_ERR_STREAM_SEEK_ERR	An error was generated, by the SEEK function, that used the file pointer.
NJ_ERR_STREAM_READ_ERR	An error was generated, by the READ function that used the file pointer.
NJ_ERR_FORMAT_INVALID	An invalid dictionary file pointer was specified.

9.32 Set FLASH Dictionary Information Function (NjxSetFlashDicInfo)

This function sets information required by the FLASH dictionary, in a FLASH dictionary information structure.

Code 9-32 Set FLASH Dictionary Information Function (NjxSetFlashDicInfo) Declaration

```
s16 NjxSetFlashDicInfo(
 IWNN_FLASH_DIC_INFO* flashDicInfo,  // FLASH dictionary information
 const IWNN_CLASS* iwnn,       // Analysis information class
 const IWNN_FILE* fileStream,  // Dictionary file pointer
 const u8* cacheArea,          // Cache region
 u32 cacheSize                       // Cache region size
)
```

Table 9-105 Set FLASH Dictionary Information Function (NjxSetFlashDicInfo) Arguments

Input/ Output	Argument	Description
OUT	IWNN_FLASH_DIC_INFO* flashDicInfo	FLASH dictionary information. Stores the dictionary file pointer and cache region, required by the FLASH dictionary. An error results if NULL is specified.
IN	const IWNN_CLASS* iwnn	Analysis information class. An error results if NULL is specified.
IN	const IWNN_FILE* fileStream	Dictionary file pointer. Stores the dictionary file pointer corresponding to the cache region. An error results if NULL is specified.
IN	const u8* cacheArea	Cache region. Stores the start pointer to the cache region. An error results if NULL is specified.
IN	u32 cacheSize	Cache region size. Stores the size of the region specified in cacheArea.

Table 9-106 Set FLASH Dictionary Information Function (NjxSetFlashDicInfo) Return Values

Return Value	Description
s16	Negative value: Error. 0: Normal exit.

Table 9-107 Set FLASH Dictionary Information Function (NjxSetFlashDicInfo) Errors

Error Code	Description
NJ_ERR_PARAM_ENVIRONMENT_NULL	A NULL pointer was specified in the argument (iwnn).
NJ_ERR_PARAM_STREAM_NULL	A NULL pointer was specified in the argument (fileStream).
NJ_ERR_PARAM_NULL	A NULL pointer was specified in the argument (flashDicInfo or cacheArea).
NJ_ERR_PARAM_TYPE_INVALID	A dictionary file pointer that does not represent a FLASH dictionary was specified.
NJ_ERR_STREAM_SEEK_ERR	An error was generated, by the SEEK function, that used the file pointer.
NJ_ERR_STREAM_READ_ERR	An error was generated, by the READ function that used the file pointer.
NJ_ERR_FORMAT_INVALID	An invalid dictionary file pointer was specified.

10 Errors

The following errors, generated internally by iWnn, are returned as the return values of functions. Error values are defined in `mw\iwnn\engine\nj_err.h`.

A defined constant representing the cause of the error can be retrieved from the error value. Get the cause of error by specifying the error value to the following defined macro.

```
NJ_GET_ERR_CODE(error_value)
```

Table 10-1 Argument Errors Caused by the Application

Cause of Error	Description and Resolution
`NJ_ERR_PARAM_ENVIRONMENT_NULL`	`NULL` was specified for the analysis information class.
`NJ_ERR_PARAM_DIC_NULL`	`NULL` was specified for the dictionary set.
`NJ_ERR_PARAM_READING_NULL`	`NULL` was specified for the reading string.
`NJ_ERR_PARAM_RESULT_NULL`	`NULL` was specified for the result storage buffer.
`NJ_ERR_PARAM_PROCESS_LEN_NULL`	A `NULL` pointer was specified for the analysis end string length storage buffer.
`NJ_ERR_PARAM_KANJI_NULL`	`NULL` was specified for the kanji string.
`NJ_ERR_PARAM_CURSOR_NULL`	`NULL` was specified for the cursor.
`NJ_ERR_DIC_HANDLE_NULL`	`NULL` was specified for the dictionary handle.
`NJ_ERR_PARAM_READING_SIZE`	An invalid reading string size was specified. Verify that the specified string is an empty string or exceeds `NJ_MAX_LEN` in length.
`NJ_ERR_PARAM_DIVISION`	An invalid division position was specified. Verify that the division position does not exceed the length of the specified string.
`NJ_ERR_PARAM_ILLEGAL_LEVEL`	An illegal number of phrases for analysis was specified. Verify that the number of phrases to analyze is 1 or `NJ_MAX_LEN`.
`NJ_ERR_PARAM_ILLEGAL_LIMIT`	An illegal maximum number of prediction candidates or learning candidates was specified. Verify that the number of prediction candidates to get is within legal limits.
`NJ_ERR_PARAM_OPERATION`	An illegal search method was specified. Verify that a prefix match search, complete match search, or derived search is specified for the search method.
`NJ_ERR_PARAM_MODE`	An invalid search candidate retrieval order was specified. Verify that frequency order or reading order is specified for the retrieval order.

Cause of Error	Description and Resolution
NJ_ERR_PARAM_TYPE_INVALID	An unsupported type was specified.
NJ_ERR_DIC_TYPE_INVALID	An invalid dictionary type was specified. Verify that there is a mistake in the specified dictionary type. The dictionary type inside the dictionary storing the dictionary set structure may also be corrupted.
NJ_ERR_DIC_NOT_FOUND	Cannot find target dictionary. Verify that a dictionary handle corresponding to the specified dictionary type is stored in the dictionary set structure.
NJ_ERR_READING_TOO_LONG	A reading string longer than NJ_MAX_LEN was specified.
NJ_ERR_NO_RULE_DIC	A rule dictionary was not specified. Be sure to store a dictionary handle for a rule dictionary in the dictionary set structure.
NJ_ERR_NO_CANDIDATE_LIST	There is no all candidate list. Make sure that NULL is not specified in the process result for the target phrase, when getting all candidates the first time. If getting all candidates for the target phrase, specify the process result the first time, and specify NULL after the first time.
NJ_ERR_AREA_SIZE_INVALID	An invalid dictionary size was specified.
NJ_ERR_BUFFER_NOT_ENOUGH	The buffer size is insufficient. Increase the specified buffer size and verify that process results can be obtained.
NJ_ERR_PARTS_OF_SPEECH_GROUP_INVALID	An invalid part of speech group was specified.
NJ_ERR_CREATE_TYPE_INVALID	An invalid create dictionary type was specified. Verify that a user dictionary or learning dictionary has been specified for the dictionary type.
NJ_ERR_INVALID_FLAG	An invalid flag was specified. Verify that a 1 or 0 was specified for the associative learning flag.
NJ_ERR_INVALID_RESULT	The specified process results are corrupted. Verify that the process result structure obtained by one of the various functions has been changed.
NJ_ERR_NOT_CONVERTED	The get all candidates function was called before using the kana-kanji conversion function. Be sure to use the kana-kanji conversion function before getting all candidates.
NJ_ERR_NOT_SELECT_YET	There is no pre-confirmation information. Be sure not to specify an empty string for the reading string, when using the get prediction candidate function for the first time.
NJ_ERR_WORD_INFO_NULL	NULL was specified for word information.

Cause of Error	Description and Resolution
NJ_ERR_USER_DIC_FULL	The user dictionary is full.
NJ_ERR_SAME_WORD	The same word is already registered in the user dictionary.
NJ_ERR_USER_READING_INVALID	An invalid reading string was specified when adding a word. Check if the reading string given for word information is NULL or an empty string. Check whether the string length exceeds NJ_MAX_LEN (learning dictionary) or NJ_MAX_USER_LEN (user dictionary).
NJ_ERR_USER_CANDIDATE_INVALID	An invalid candidate was specified when adding words. Check if the candidate string given for word information is NULL or an empty string. Check whether the string length exceeds NJ_MAX_RESULT_LEN (learning dictionary) or NJ_MAX_USER_CANDIDATE_LEN (user dictionary).
NJ_ERR_WORD_NOT_FOUND	The word may already be deleted. Perform a word search and try again.
NJ_ERR_DIC_VERSION_INVALID	An older version dictionary (V1.11 or earlier) was specified to the change dictionary type function.
NJ_ERR_CACHE_BROKEN	Be sure to initialize the cache management region, because it is corrupted.
NJ_ERR_PARAM_INDEX_INVALID	An additional information index greater than or equal to the maximum additional information mount count (NJ_MAX_ADDITIONAL_INFO) was specified.
NJ_ERR_PARAM_ADD_INFO_INVALID_SIZE	Zero, or a value that does not match the one stored in the additional information region, was specified for the additional information region size.
NJ_ERR_PARAM_NULL	A NULL pointer was specified in an argument. Check arguments.
NJ_ERR_STREAM_SEEK_ERR	An error was generated, by the SEEK function, that used the file pointer. Verify that the file pointer value is correct.
NJ_ERR_STREAM_READ_ERR	An error was generated, by the READ function that used the file pointer. Verify that the file pointer value is correct.

Table 10-2 Errors Requiring a Change to the Environment

Cause of Error	Description and Resolution
NJ_ERR_CANDIDATE_TOO_LONG	There are candidates longer than NJ_MAX_RESULT_LEN. Either increase the value of NJ_MAX_RESULT_LEN or delete candidates longer than NJ_MAX_RESULT_LEN from the dictionary.

Table 10-3 Errors Requiring a Dictionary to be Checked or Created

Cause of Error	Description and Resolution
NJ_ERR_NO_PARTS_OF_SPEECH	Cannot get part of speech number for the conversion process. Use the check dictionary API to check the integrity of each dictionary.
NJ_ERR_FORMAT_INVALID	The dictionary is corrupted. Use the check dictionary API to check the integrity of each dictionary.
NJ_ERR_CANNOT_GET_QUEUE	Cannot get word information. Recreate the corrupted user dictionary or learning dictionary, using the create dictionary function.
NJ_ERR_DIC_BROKEN	The dictionary is corrupted. Be sure to perform an integrity check on each dictionary, using the check dictionary function. Also, be sure to restore the dictionary if it is a user dictionary or learning dictionary.
NJ_ERR_CANNOT_RESTORE	Cannot restore dictionary. Re-create the corrupted user dictionary or learning dictionary using the create dictionary function.

11 Pseudo Dictionaries

11.1 Overview

Under iWnn, in addition to normal dictionaries created as data files (such as a standard dictionary), you can treat a program module that meets dictionary interface specifications as a pseudo dictionary.

With a standard dictionary, you must overwrite data in the dictionary file in order to change words registered in it, but with a pseudo dictionary you can change stored words dynamically. Also, there is a limit on the number of words that can be registered in a standard dictionary, but because a pseudo dictionary is a program, an unlimited number of words can be mechanically generated.

Table 11-1 Pseudo Dictionary Examples

Type of Pseudo Dictionary	Description
Pseudo Dictionary for Katakana Conversion	Converts Hiragana into Katakana. For example, “う゛ぁいおりん” is converted to “ヴァイオリン”.
Pseudo Dictionary for Expressing Numbers	Converts number readings into numeric characters. (Used for conversion.) For example, “にせんろっぴゃく” is converted into “2600”. Converts numeric characters into a reading. (Used for morphological analysis.) For example, “12/20” is converted into “じゅうにがつはつか”.
Pseudo Dictionary for Address Books	Accesses the address book database built into a mobile telephone, and uses mail addresses, telephone numbers, and the kanji notation for names as conversion candidates based on name readings.
Pseudo Dictionary for Relative Dates	Generates conversion candidates using the terminal clock. For example, “きょう” is converted into “2007 年 11 月 11 日” and “いま” is converted into “9: 00“.
Pseudo Dictionary for Calculators	Entered formulas are converted into conversion candidates, using the calculated result. For example, “120*2” is converted into “240”.
Pseudo Dictionary for Numeric Relations	Generates relationship prediction candidates corresponding to the number just confirmed. For example, “12” results in candidates “時”, “分”, ”月”, ”週”, “日”. “ 24” results in candidates “時”, “分”, “週”, “日”. “30” results in candidates “分”, “週”, “日”.

With Advanced Wnn, candidates that must be created dynamically are created by a pseudo candidate process inside the engine. But with iWnn, such candidates are provided using pseudo dictionaries, treated as external modules.

Macros and functions required to create candidates are defined in `nj_lib.h`, `nj_ext.h`, `njd.h`, `nj_dicif.h`, and `iwnn.h`.

Be sure to include these files when creating a pseudo dictionary.

11.2 Pseudo Dictionary Interface (IWNN_PROGRAM_DIC_IF)

The interface function for connecting a pseudo dictionary to iWnn is defined as follows.

Code 11-1 Pseudo Dictionary Interface Function (IWNN_PROGRAM_DIC_IF) Declaration

```
#define IWNN_PROGRAM_DIC_IF \
       (*program_dic_operation)(IWNN_CLASS* iwnn, \
                                u16 request, \
                                INN_PROGRAM_DIC_MESSAGE *message)
```

Table 11-2 Pseudo Dictionary Interface Function (IWNN_PROGRAM_DIC_IF) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CLASS* Iwnn	Analysis information class. Stores internal information of the iWnn engine. Analysis status is retrieved from here for a pseudo dictionary. **Note:** Usually not written to iwnn.
IN	u16 request	Process request. Content of the process request sent from iWnn to the pseudo dictionary. The following modes are available: NJG_OPERATION_SEARCH: Search for first candidate. NJG_OPERATION_SEARCH_NEXT: Search for next candidate. NJG_OPERATION_GET_WORD_INFO: Get word information. NJG_OPERATION_GET_STROKE: Get reading string. NJG_OPERATION_GET_STR: Get candidate string. NJG_OPERATION_GET_ADDITIONAL: Get additional information string. NJG_OPERATION_LEARN: Learn word. NJG_OPERATION_UNDO_LEARN: Undo learning. NJG_OPERATION_ADD_WORD: Add word. NJG_OPERATION_DELETE_WORD: Delete word. With a pseudo dictionary, processing is performed using iwnn and message information for each request.
IN/ OUT	IWNN_PROGRAM_DIC_MESSAGE *message	Dictionary interface message. Stores information required for processing between the iWnn engine and pseudo dictionaries. Information generated by a pseudo dictionary is stored in this structure and returned to the iWnn engine.

Table 11-3 Pseudo Dictionary Interface Function (IWNN_PROGRAM_DIC_IF) Return Values

Return Value	Description
s16	[When using NJG_OPERATION_SEARCH/NJG_OPERATION_SEARCH_NEXT]: 1: Candidates present. 0: No candidates. Negative value: Error. [When using NJG_OPERATION_GET_WORD_INFO]: 0: Normal exit. Negative value: Error. [When using NJG_OPERATION_GET_STROKE/NJG_OPERATION_GET_STR/NJG_OPERATION_GET_ADDITIONAL]: Stored string length. Negative value: Error. [When using NJG_OPERATION_LEARN]: 0: Normal exit. Negative value: Error.

Table 11-4 Pseudo Dictionary Interface Function (IWNN_PROGRAM_DIC_IF) Errors

Error Code	Description
Any	Returned when an error occurs in the pseudo dictionary (when the pseudo dictionary returns an error). If subsequent processing can continue, iWnn handles the error internally and the iWnn process continues to execute. If subsequent processing cannot continue, iWnn returns an error to the application. Errors returned to the application as an error code with the error value in the lower 7 bits.

11.3 Pseudo Dictionary Message (IWNN_PROGRAM_DIC_MESSAGE)

Code 11-2 Pseudo Dictionary Message (IWNN_PROGRAM_DIC_MESSAGE) Structure Configuration

```
struct IWNN_PROGRAM_DIC_MESSAGE {
 IWNN_SEARCH_CONDITION* condition;   // Search conditions
 IWNN_SEARCH_LOCATION_SET* location;// Search cursor
 IWNN_DIC_SET* dicSet;         // Dictionary set to be searched
 IWNN_WORD* word;              // Word information
 IWNN_LEARN_WORD* learnWord;   // Word registration information
 wchar_t* stroke;              // Storage for obtained reading string
 wchar_t* string;              // Storage for obtained notation string
 wchar_t* additional;          // Storage for obtained additional
                                        information string
 u16 strokeSize;               // stroke size
 u16 strSize;                        // string size
```

```
 u32 additionalSize;                // additional size
 u16 dicIdx;                        // Dictionary mount location
};
```

Table 11-5 Pseudo Dictionary Message (`IWNN_PROGRAM_DIC_MESSAGE`) Structure Members

Member	Description
`IWNN_SEARCH_CONDITION*` Condition	Search conditions. Includes information such as the search method (forward lookup prefix match search, derived search, and so forth), the reading string to search for, the order in which to return candidates (reading order or frequency order).
`IWNN_SEARCH_LOCATION_SET*` `location`	Search cursor. Includes information such as the status of search results (candidate found/not found) and the ID of candidates returned by the dictionary. This is used to store which candidates in the dictionary are being accessed.
`IWNN_DIC_SET*` `dicSet`	Dictionary set to be searched. Includes dictionary set information, such as the dictionary handle to use in the search and search limits. This is used to access dictionary information based on the dictionary mount location.
`IWNN_WORD*` `word`	Word information. Includes information such as the length of the independent word/ancillary word parts, the part of speech, the frequency value, and the dictionary cursor for the word searched for.
`IWNN_LEARN_WORD *` `learnWord`	Word registration information. Includes information such as the part of speech, connect flag for the candidate previously confirmed, and undo flag for the word searched for.
`wchar_t*` `stroke`	Reading string. Used when getting a reading string and during learning.
`wchar_t*` `string`	Candidate string. Used when getting a candidate string and during learning.
`u16` `strokeSize`	Stroke size. Specified in bytes.
`u16` `strSize`	String size. Specified in bytes.
`u16` `dicIdx`	Dictionary mount location Mount location in `IWNN_DIC_SET` of the dictionary that sent the message.

Note: For details, refer to section 11.5 Pseudo Dictionary Processing Specifications.

11.4 Word Registration Information (IWNN_LEARN_WORD)

Code 11-3 Word Registration Information (IWNN_LEARN_WORD) Structure Configuration

```
struct IWNN_LEARN_WORD {
 u16 forePartsOfSpeech;        // Previous part of speech number
 u16 backPartsOfSpeech;        // Later part of speech number
 u16 connect;                        // Connect flag
 u16 undo;                     // Undo flag
};
```

Table 11-6 Word Registration Information (IWNN_LEARN_WORD) Structure Members

Member	Description
u16 forePartsOfSpeech	Previous part of speech number. Includes previous part of speech number information for the candidate to be learned or added.
u16 backPartsOfSpeech	Later part of speech number. Includes later part of speech number information for a candidate to be learned or added.
u16 connect	Connect flag. Includes flag information representing connectivity between the candidate to be learned or added, and the previously confirmed candidate.
u16 undo	Undo flag. Includes information on the flag representing the undo start location for the candidate to be added, and how many learning operations to undo.

Note: For details, see 11.5 Pseudo Dictionary Processing Specifications.

11.5 Pseudo Dictionary Processing Specifications

The processing to be performed by the pseudo dictionary for each processing request is as follows.

11.5.1 First Search (NJG_OPERATION_SEARCH)

First search returns the single highest priority candidate of all candidates that match the search string.

Table 11-7 First Search (NJG_OPERATION_SEARCH) Input Parameters

message->condition	
message->condition ->reading	**Search string.** Represents the reading string during prediction or conversion, or the string to be analyzed during morphological analysis. Stores the reading of the pre-confirmation phrase during a derived search. This string is NULL terminated.
message->condition ->readingLen	**Search string length.** Represents the length of the string to be searched for.
message->condition ->operation	**Search method.** NJ_CURSOR_OPERATION_COMPLETE: Forward lookup complete match search. NJ_CURSOR_OPERATION_FORE: Forward lookup prefix match search. NJ_CURSOR_OPERATION_REVERSE: Reverse lookup complete match search. NJ_CURSOR_OPERATION_REVERSE_FORE: Reverse lookup prefix match search. NJ_CURSOR_OPERATION_LINK: Derived search.
message->condition ->mode	**Search order.** NJ_CURSOR_MODE_FREQUENCY: Frequency order. NJ_CURSOR_MODE_READING: Reading order.
message->condition ->partsOfSpeech	**Part of speech restriction.** Restriction condition for the part of speech to be created. Use NjdConnectTest to determine whether the candidate to be created conforms to the part of speech restriction.
message->condition ->kanji	**Pre-confirmed phrase notation string (only when using *NJ_CURSOR_OPERATION_LINK).** Represents the notation string of the pre-confirmed phrase to be accessed during relationship prediction. The string is NULL terminated.
Message->condition ->noReadingFore	**Part of speech restriction (only when using *NJ_CURSOR_OPERATION_LINK).** Part of speech restriction condition used during relationship prediction.
message->condition ->ancillaryConnect	**Ancillary word connection status.** Indicates whether the candidate connects to an ancillary word.
message->location	
message->location ->dicFrequency	**Dictionary frequency.** Sets the upper and lower limits on the frequency of candidates to be returned. The pseudo dictionary gives a frequency to candidates in the range from dicFrequency.base to dicFrequency.high. dicFrequency.base is guaranteed to be equal to or less than dicFrequency.high.
message->dicIdx	
message->dicIdx	**Dictionary index number.**

message->dicSet	
message->dicSet	**Dictionary set to be searched.** Defines a valid dictionary set. Used to get the rule dictionary handle (dicSet.ruleHandle[0]).
Iwnn	
iwnn->njcMode	**Function calling mode.** Represents the status of the iWnn function called from the application. 0: Initialized status or currently calling a prediction function. 1: Multiple phrase conversion status. 2: Getting all candidates status. 3: Morphological analysis status.
iwnn ->environment.type	**Detailed status during prediction conversion.** Represents the internal status when a prediction function is called. NJ_ANALYZE_INITIAL: Initial setting status. NJ_ANALYZE_NEXT_PREDICTION: Derived search status. NJ_ANALYZE_FORWARD_SEARCH: Pre-fix search status. NJ_ANALYZE_FORWARD_SEARCH_WITH_NO_READING: Forward match search status, including no reading prediction dictionaries. NJ_ANALYZE_CONVERSION_MULTIPLE: Multiple conversion result retrieval status. NJ_ANALYZE_CONVERSION_SINGLE: Single phrase conversion result retrieval status. NJ_ANALYZE_COMPLETE: All candidate analysis result retrieval status. NJ_ANALYZE_END: Conversion complete status.
iwnn ->state	**State setting.** Represents current state information. May be accessed as necessary.

Table 11-8 First Search (NJG_OPERATION_SEARCH) Output

message->location	
message->location ->location.current	**Candidate ID.** Represents the number used to identify the candidate. Be sure to assign a unique ID number to each candidate generated by the dictionary. Information such as a reading string, notation string, and part of speech is generated based on this ID in subsequent processing.
message->location ->cacheFrequency	**Frequency value.** Represents the priority level of the candidate.
message->location ->location.currentInfo	**Phrase count information.** Be sure to specify the fixed value 0x10.
message->location ->location.status	**Search result status.** Stores one of the following values according to the search results. NJ_STATUS_SEARCH_READY: Complete match present. NJ_STATUS_SEARCH_END: No complete match (prefix match present). NJ_STATUS_SEARCH_END_EXTENSION: No complete match (no prefix match).

Return Values	
1:	**Search match candidate present.**
0:	**No search match candidate.**
Negative value:	**Error.**

11.5.2 Search Next Candidate (NJG_OPERATION_SEARCH_NEXT)

Search next candidate returns the candidate with the next highest priority that matches the search string.

Table 11-9 Search Next Candidate (NJG_OPERATION_SEARCH_NEXT) Input Parameters

message->condition	
Message->condition ->reading	**Search string.** Represents the reading string during prediction or conversion, or the string to be analyzed during morphological analysis. Stores the reading of the pre-confirmation phrase during a derived search. This string is NULL terminated.
message->condition ->readingLen	**Search string length.** Represents the length of the string to be searched for.
message->condition ->operation	**Search method.** NJ_CURSOR_OPERATION_COMPLETE: Forward lookup complete match search. NJ_CURSOR_OPERATION_FORE: Forward lookup prefix match search. NJ_CURSOR_OPERATION_REVERSE: Reverse lookup complete match search. NJ_CURSOR_OPERATION_REVERSE_FORE: Reverse lookup prefix match search. NJ_CURSOR_OPERATION_LINK: Derived search.
message->condition ->mode	**Search order.** NJ_CURSOR_MODE_FREQUENCY: Frequency order. NJ_CURSOR_MODE_READING: Reading order.
message->condition ->partsOfSpeech	**Part of speech restriction.** Restriction condition for the part of speech to be created. Use NjdConnectTest to determine whether the candidate to be created conforms to the part of speech restriction.
message->condition ->kanji	**Pre-confirmed phrase notation string (only when using *NJ_CURSOR_OPERATION_LINK).** Represents the notation string of the pre-confirmed phrase to be accessed during relationship prediction. The string is NULL terminated.
message->condition ->noReadingFore	**Part of speech restriction (only when using *NJ_CURSOR_OPERATION_LINK).** Part of speech restriction condition used during relationship prediction.
message->condition ->ancillaryConnect	**Ancillary word connection status.** Indicates whether the candidate connects to an ancillary word.

message->location	
message->location ->location.current	**Previous Candidate ID.** Represents the ID number of the previously returned candidate. Basically, the next candidate is searched for based on this ID number.
message->location ->dicFrequency	**Dictionary frequency.** Sets the upper and lower limits on the frequency of candidates to be returned. The pseudo dictionary gives a frequency to candidates in the range from dicFrequency.base to dicFrequency.high. dicFrequency.base is guaranteed to be equal to or less than dicFrequency.high.
message->dicIdx	
message->dicIdx	**Dictionary index number.**
message->dicSet	
message->dicSet	**Dictionary set to be searched.** Defines a valid dictionary set. Used to get the rule dictionary handle (dicSet.ruleHandle[0]).
Iwnn	
iwnn->njcMode	**Function calling mode.** Represents the status of the iWnn function called from the application. 0: Initialized status or currently calling a prediction function. 1: Multiple phrase conversion status. 2: Getting all candidates status. 3: Morphological analysis status.
iwnn ->environment.type	**Detailed status during prediction conversion.** Represents the internal status when a prediction function is called. NJ_ANALYZE_INITIAL: Initial setting status. NJ_ANALYZE_NEXT_PREDICTION: Derived search status. NJ_ANALYZE_FORWARD_SEARCH: Pre-fix search status. NJ_ANALYZE_FORWARD_SEARCH_WITH_NO_READING: Forward match search status, including no reading prediction dictionaries. NJ_ANALYZE_CONVERSION_MULTIPLE: Multiple conversion result retrieval status. NJ_ANALYZE_CONVERSION_SINGLE: Single phrase conversion result retrieval status. NJ_ANALYZE_COMPLETE: All candidate analysis result retrieval status. NJ_ANALYZE_END: Conversion complete status.
iwnn ->state	**State setting.** Represents current state information. May be accessed as necessary.

Table 11-10 Search Next Candidate (NJG_OPERATION_SEARCH_NEXT) Output

message->location	
message->location ->location.current	**Candidate ID.** Represents the number used to identify the candidate. Be sure to assign a unique ID number to each candidate generated by the dictionary. Information such as a reading string, notation string, and part of speech is generated based on this ID in subsequent processing.
message->location ->cacheFrequency	**Frequency value.** Represents the priority level of the candidate.
message->location ->location.currentInfo	**Phrase count information.** Be sure to specify the fixed value 0x10.
message->location ->location.status	**Search result status.** Stores one of the following values according to the search results: NJ_STATUS_SEARCH_READY: Complete match present. NJ_STATUS_SEARCH_END: No complete match (prefix match present).
Return Values	
1:	**Search match candidate present.**
0:	**No search match candidate.**
Negative value:	**Error.**

11.5.3 Get Word Information (NJG_OPERATION_GET_WORD_INFO)

Get word information creates word information (message->word) based on search cursor (message->location) information.

Table 11-11 Get Word Information (NJG_OPERATION_GET_WORD_INFO) Input Parameters

message->location	
message->location ->location.current	**Candidate ID.** Represents the ID number of the candidate to be obtained. Candidate information corresponding to this ID is created. The location created by NJG_OPERATION_SEARCH or NJG_OPERATION_SEARCH_NEXT is specified. A location created by another dictionary is never specified.
message->dicIdx	
message->dicIdx	**Dictionary index number.**
message->dicSet	
message->dicSet	**Dictionary set to be searched.** Defines a valid dictionary set. Used to get the rule dictionary handle (dicSet.ruleHandle[0]).
message->word	
message->word ->reading	**Search string.** Represents the search string used during candidate search.
NJ_GET_READING_LEN_FROM_STEM (message->word)	**Search string length.** Represents the length of the string to be searched for.

Table 11-12 Get Word Information (NJG_OPERATION_GET_WORD_INFO) Output

message->word	
message->word ->stem.loc	**Search cursor.** Stores message->location->location as is.
message->word ->stem	**Independent word information.** This information is set as follows based on search cursor information. Reading string length: NJ_SET_READING_LEN_TO_STEM (message->word, string length). This is the complete reading string of the candidate, not the search string. Notation string length: NJ_SET_CANDIDATE_LEN_TO_STEM (message->word, string length). Part of speech number: NJ_SET_FORE_PARTS_OF_SPEECH_TO_STEM (message->word, fore part of speech). NJ_SET_BACK_PARTS_OF_SPEECH_TO_STEM (message->word, back part of speech). Pseudo candidate type: NJ_TYPE_UNDEFINE fixed. NJ_SET_WORD_TYPE_TO_STEM (message->word, NJ_TYPE_UNDEFINE). Frequency level: NJ_SET_FREQUENCY_TO_STEM (message->word, frequency). Set to the value of message->location->cacheFrequency.

Return Values	
0:	**Normal exit.**
Negative value:	**Error.**

The part of speech number used to set independent word information is obtained using the get part of speech function (**`NjdRuleGetPartsOfSpeech`**).

Be sure to get and set both the fore and back part of speech numbers for the part of speech in question.

For example, get and set information as follows for a typical noun.

Code 11-4 Get Part of Speech Function (`NjdRuleGetPartsOfSpeech`) Get/Set Example

```
NJ_SET_FORE_PARTS_OF_SPEECH_TO_STEM(message->word,
 NjdRuleGetPartsOfSpeech(iwnn->dicSet.ruleHandle[NJ_MODE_TYPE_CONVERSION],
                         NJ_PARTS_OF_SPEECH_NOUN_NO_CONJUGATION_FORE));

NJ_SET_BACK_PARTS_OF_SPEECH_TO_STEM(message->word,
 NjdRuleGetPartsOfSpeech(iwnn->dicSet.ruleHandle[NJ_MODE_TYPE_CONVERSION],
                         NJ_PARTS_OF_SPEECH_NOUN_NO_CONJUGATION_BACK));
```

Code 11-5 Get Part of Speech Function (`NjdRuleGetPartsOfSpeech`) Declaration

```
s16 ret = NjdRuleGetPartsOfSpeech(
 IWNN_DIC_HANDLE ruleHandle,         // Rule dictionary handle
 u8 type                        // Part of speech type
)
```

Table 11-13 Get Part of Speech Function (`NjdRuleGetPartsOfSpeech`) Arguments

Input/ Output	Argument	Description
IN	`IWNN_DIC_HANDLE ruleHandle`	Rule dictionaryhandle. Specifies the handle of the rule dictionary to be used. Use `iwnn->dicSet.ruleHandle[NJ_MODE_TYPE_CONVERSION]` for pseudo dictionaries.
IN	`u8 type`	Part of speech type. `NJ_PARTS_OF_SPEECH_NOUN_NO_CONJUGATION_FORE` regular noun (fore). `NJ_PARTS_OF_SPEECH_NOUN_NO_CONJUGATION_BACK` regular noun (back). `NJ_PARTS_OF_SPEECH_NOUN_FORE` noun conjugating with –suru (fore). `NJ_PARTS_OF_SPEECH_NOUN_BACK` noun conjugating with –suru (back). `NJ_PARTS_OF_SPEECH_PERSON_NAME_FORE` person's name (fore). `NJ_PARTS_OF_SPEECH_PERSON_NAME_BACK` person's name (back). `NJ_PARTS_OF_SPEECH_PLACE_NAME_FORE` place name (fore). `NJ_PARTS_OF_SPEECH_PLACE_NAME_BACK` place name (back). `NJ_PARTS_OF_SPEECH_SYMBOL_FORE` symbol (fore). `NJ_PARTS_OF_SPEECH_SYMBOL_BACK` symbol (back). `NJ_PARTS_OF_SPEECH_NUMERIC_BACK` number (fore) `NJ_PARTS_OF_SPEECH_SINGLE_KANJI_FORE` single kanji (fore). `NJ_PARTS_OF_SPEECH_SINGLE_KANJI_BACK` single kanji (back). `NJ_PARTS_OF_SPEECH_PSEUDO_FORE` pseudo (fore). `NJ_PARTS_OF_SPEECH_PSEUDO_BACK` pseudo (back).

Table 11-14 Get Part of Speech Function (`NjdRuleGetPartsOfSpeech`) Return Values

Return Value	Description
s16	Part of speech number (0 if there is an error).

11.5.4 Get Reading (`NJG_OPERATION_GET_STROKE`)

Get reading generates a reading string from word information (`message->word`).

In the case of prediction candidates, the complete reading string of the candidate is returned, rather than the input reading string.

Table 11-15 Get Reading (NJG_OPERATION_GET_STROKE) Input Parameters

message->word	
message->word	**Word information.** Represents the word information of the candidate to be obtained. Word information created by NJG_OPERATION_GET_WORD_INFO is specified. Words created by another dictionary are never specified.
message->word ->reading	**Search string.** Represents the search string used during candidate search. Because a complete reading string is included in the search string, in the case of a complete match candidate, a reading string can be obtained from the search string.
message->word ->stem.location.current	**Candidate ID.** Represents the ID number of the candidate to be obtained. Because a complete reading string cannot be obtained from the search string, in the case of a prefix match candidate, reading data for the word must be obtained from the dictionary.
NJ_GET_READING_LEN_FROM_STEM (message->word)	**Reading string length.** Represents the complete reading string length. A reading string of this length is always returned in the output.
message->strokeSize	
message->strokeSize	**Reading string storage region byte size.** Represents the size of the region (message->stroke) storing the reading string.

Table 11-16 Get Reading (NJG_OPERATION_GET_STROKE) Output

message->stroke	
message->stroke	**Reading string.** Stores the complete reading string for the specified word.
Return Values	
Integer equal to or greater than 0:	**Stored reading string length.**
Negative value:	**Error.**

11.5.5 Get Notation (NJG_OPERATION_GET_STR)

Get notation generates a notation string from word information (message->word).

Table 11-17 Get Notation (NJG_OPERATION_GET_STR) Input Parameters

message->word	
message->word	**Word information.** Represents the word information of the candidate to be obtained. Word information created by NJG_OPERATION_GET_WORD_INFO is specified. Words creates by another dictionary are never specified.
message->word ->reading	**Search string.** Represents the search string used during candidate search. Because a complete reading string is included in the search string, in the case of a complete match candidate, a reading string can be obtained from the search string.
message->word ->stem.location.current	**Candidate ID.** Represents the ID number of the candidate to be obtained.
NJ_GET_READING_LEN_FROM_STEM (message->word)	**Reading string length.** Represents the complete reading string length.
NJ_GET_CANDIDATE_LEN_FROM_STEM (message->word)	**Candidate string length.** Represents the candidate string length of the word. A candidate string of this length is always returned in the output.
message->strSize	
message->strSize	**Notation string storage region byte size.** Size of the region (message->string) storing the notation string.

Table 11-18 Get Notation (NJG_OPERATION_GET_STR) Output

message->string	
message->string	**Notation string.** Stores the notation string for the word.
Return Values	
Integer equal to or greater than 0:	**Stored notation string length.**
Negative value:	**Error.**

11.5.6 Get Additional Information (NJG_OPERATION_GET_ADDITIONAL)

Get additional information generates an additional information string based on word information (message->word).

Table 11-19 Get Additional Information (NJG_OPERATION_GET_ADDITIONAL) Input Parameters

message->word	
message->word	**Word information.** Represents the word information of the candidate to be obtained. Word information created by NJG_OPERATION_GET_WORD_INFO is specified. Words creates by another dictionary are never specified.
message->word ->reading	**Search string.** Represents the search string used during candidate search. Because a complete reading string is included in the search string, in the case of a complete match candidate, a reading string can be obtained from the search string.
message->word ->stem.location.current	**Candidate ID.** Represents the ID number of the candidate to be obtained.
NJ_GET_READING_LEN_FROM_STEM (message->word)	**Reading string length.** Represents the complete reading string length.
NJ_GET_CANDIDATE_LEN_FROM_STEM (message->word)	**Candidate string length.** Represents the candidate string length of the word. A candidate string of this length is always returned in the output.
message->dicIdx	**Additional information index number.** Used when getting additional information.
message->additionalSize	
message->additionalSize	**Additional information string storage region byte size.** Represents the size of the region (message->additional) storing the additional information string.

Table 11-20 Get Additional Information (NJG_OPERATION_GET_ADDITIONAL) Output

message->additional	
message->additional	**Additional information string.** Stores the additional information string of the word.
Return Values	
Integer equal to or greater than 0:	**Stored additional information string length.**
Negative value:	**Error.**

11.5.7 Learn (NJG_OPERATION_LEARN)

This operation learns words according to input parameters.

This operation message is sent even when learning a word other than those obtained from a pseudo dictionary. When learning a word obtained from a pseudo dictionary, be sure to determine that the word was obtained from a pseudo dictionary before executing learning.

This operation message is not sent when learning by morphological analysis (**MmxSelect**).

Table 11-21 Learn (NJG_OPERATION_LEARN) Input Parameters

message->dicIdx	
message->dicIdx	**Dictionary index number.** Used when getting a dictionary handle.
message->dicSet	
message->dicSet	**Dictionary set to be searched.** Defines a valid dictionary set. Used to get a rule dictionary handle (dicSet.ruleHandle[0]) or dictionary handle.
message->word	
message->word	**Word information.** Represents the word information of the candidate to be obtained. Word information created by NJG_OPERATION_GET_WORD_INFO is specified. Words created by another dictionary are never specified.
message->word ->stem.location.current	**Candidate ID.** Represents the ID number of the candidate to be learned.
message->word ->stem.location.handle	**Dictionary handle.** Represents the dictionary handle from which the candidate to be learned was obtained.
message->word ->stem.location.type	**Dictionary handle type.** Represents the dictionary handle type from which the candidate to be learned was obtained. IWNN_DIC_INFO: : type is stored.
message->learnWord	
message->learnWord ->forePartsOfSpeech	**Fore part of speech number.** The fore part of speech of the candidate to be learned.
message->learnWord ->backPartsOfSpeech	**Back part of speech number.** The back part of speech of the candidate to be learned.
message->learnWord ->connect	**Connection flag.** The flag indicating connectivity between the candidate to be learned and the pre-confirmation candidate.

message->dicIdx	
message->learnWord ->undo	**Undo flag.** The flag indicating whether the candidate to be learned represents the undo start location.
message->stroke	
message->stroke	**Reading string storage region.** Stores the reading string of the candidate to be learned.
message->strokeSize	
message->strokeSize	**Reading string storage region string length.** Represents the reading string length of the candidate to be learned.
message->string	
message->string	**Notation string storage region.** Stores the notation string of the candidate to be learned.
message->strSize	
message->strSize	**Notation string storage region string length.** Represents the notation string length of the candidate to be learned.
message->additional	
message->additional	**Additional information string storage region.** Stores the additional information string for the candidate to be learned.
message->additionalSize	
message->additionalSize	**Additional information string storage region string length.** Represents the additional string length for the candidate to be learned.

Table 11-22 Learn (NJG_OPERATION_LEARN) Output

Return Values	
Integer equal to or greater than 0:	**Normal exit.**
Negative value:	**Error.**

11.5.8 Undo Learning (NJG_OPERATION_UNDO_LEARN)

Undo learning undoes learning for the number of times specified in word registration information (message->learnWord).

Table 11-23 Undo Learning (NJG_OPERATION_UNDO_LEARN) Input Parameters

message->dicIdx	
message->dicIdx	**Dictionary index number.** Used when getting a dictionary handle.
message->dicSet	
message->dicSet	**Dictionary set to be searched.** Enabled dictionary set definition. Used to get a rule dictionary handle (dicSet.ruleHandle[0]) or dictionary handle.
message->learnWord	
message->learnWord ->undo	**Undo count.** Represents the number of times to undo learning.

Table 11-24 Undo Learning (NJG_OPERATION_UNDO_LEARN) Output

Return Values	
Integer equal to or greater than 0:	**Normal exit.**
Negative value:	**Error.**

11.5.9 Add Word (NJG_OPERATION_ADD_WORD)

Add word adds words according to the input parameters.

Table 11-25 Add Word (NJG_OPERATION_ADD_WORD) Input Parameters

message->dicIdx	
message->dicIdx	**Dictionary index number.** Used when getting a dictionary handle.
message->dicSet	
message->dicSet	**Dictionary set to be searched.** Defines a valid dictionary set. Used to get a rule dictionary handle (dicSet.ruleHandle[0]) or dictionary handle.
message->learnWord	
message->learnWord -> forePartsOfSpeech	**Fore part of speech number.** The fore part of speech of the candidate to be learned.
message->learnWord -> backPartsOfSpeech	**Back part of speech number.** The back part of speech of the candidate to be learned.

message->dicIdx	
message->learnWord ->connect	**Connection flag.** The flag indicating connectivity between the candidate to be learned and the pre-confirmation candidate.
message->learnWord ->undo	**Undo flag.** The flag indicating whether the candidate to be learned represents the undo start location.
message->stroke	
message->stroke	**Reading string storage region.** Stores the reading string of the candidate to be learned.
message->strokeSize	
message->strokeSize	**Reading string storage region string length.** Represents the reading string length of the candidate to be learned.
message->string	
message->string	**Notation string storage region.** Stores the notation string of the candidate to be learned.
message->strSize	
message->strSize	**Notation string storage region string length.** Represents the notation string length of the candidate to be learned.
message->additional	
message->additional	**Additional information string storage region.** Stores the additional information string for the candidate to be learned.
message->additionalSize	
message- >additionalSize	**Additional information string storage region string length.** Represents the additional string length for the candidate to be learned.

Table 11-26 Add Word (NJG_OPERATION_ADD_WORD) Output

Return Values	
Integer equal to or greater than 0:	**Normal exit.**
Negative value:	**Error.**

11.5.10 Delete Word (NJG_OPERATION_DELETE_WORD)

Delete word deletes words according to the input parameters.

Table 11-27 Delete Word (NJG_OPERATION_DELETE_WORD) Input Parameters

message->word	
message->word	**Word information.** Represents the word information of the candidate to be learned. Word information created by NJG_OPERATION_GET_WORD_INFO is specified. Words created by another dictionary are never specified.
message->word ->stem.location.current	**Candidate ID.** Represents the ID number of the candidate to be learned.
message->word ->stem.location.handle	**Dictionary handle.** Represents the dictionary handle from which the candidate to be learned was obtained.
message->word ->stem.location.type	**Dictionary handle type.** Represents the dictionary handle type from which the candidate to be learned was obtained. IWNN_DIC_INFO: : type is stored.

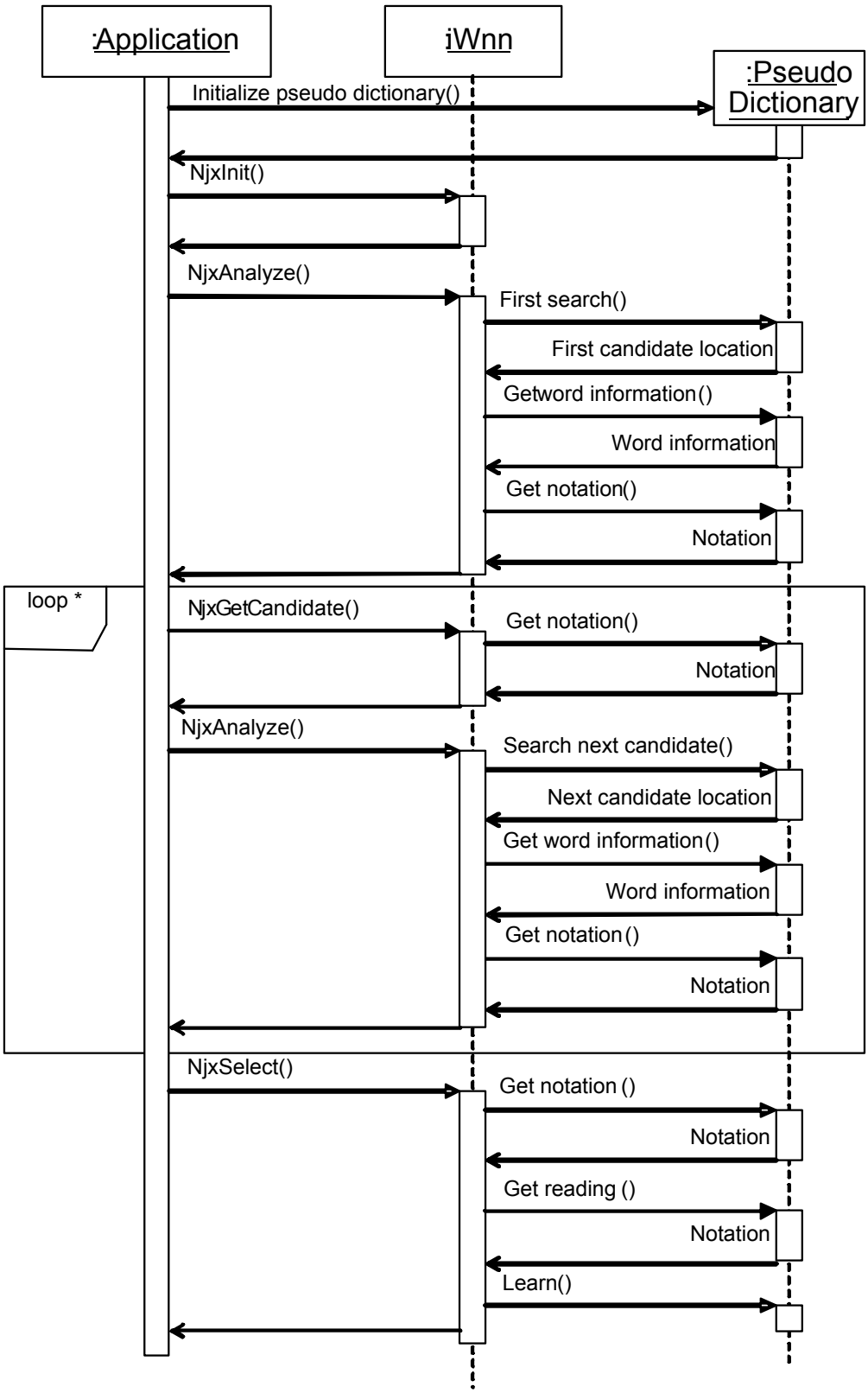
Table 11-28 Delete Word (NJG_OPERATION_DELETE_WORD) Output

Return Values	
Integer equal to or greater than 0:	**Normal exit.**
Negative value:	**Error.**

11.6 Basic Operation Sequence

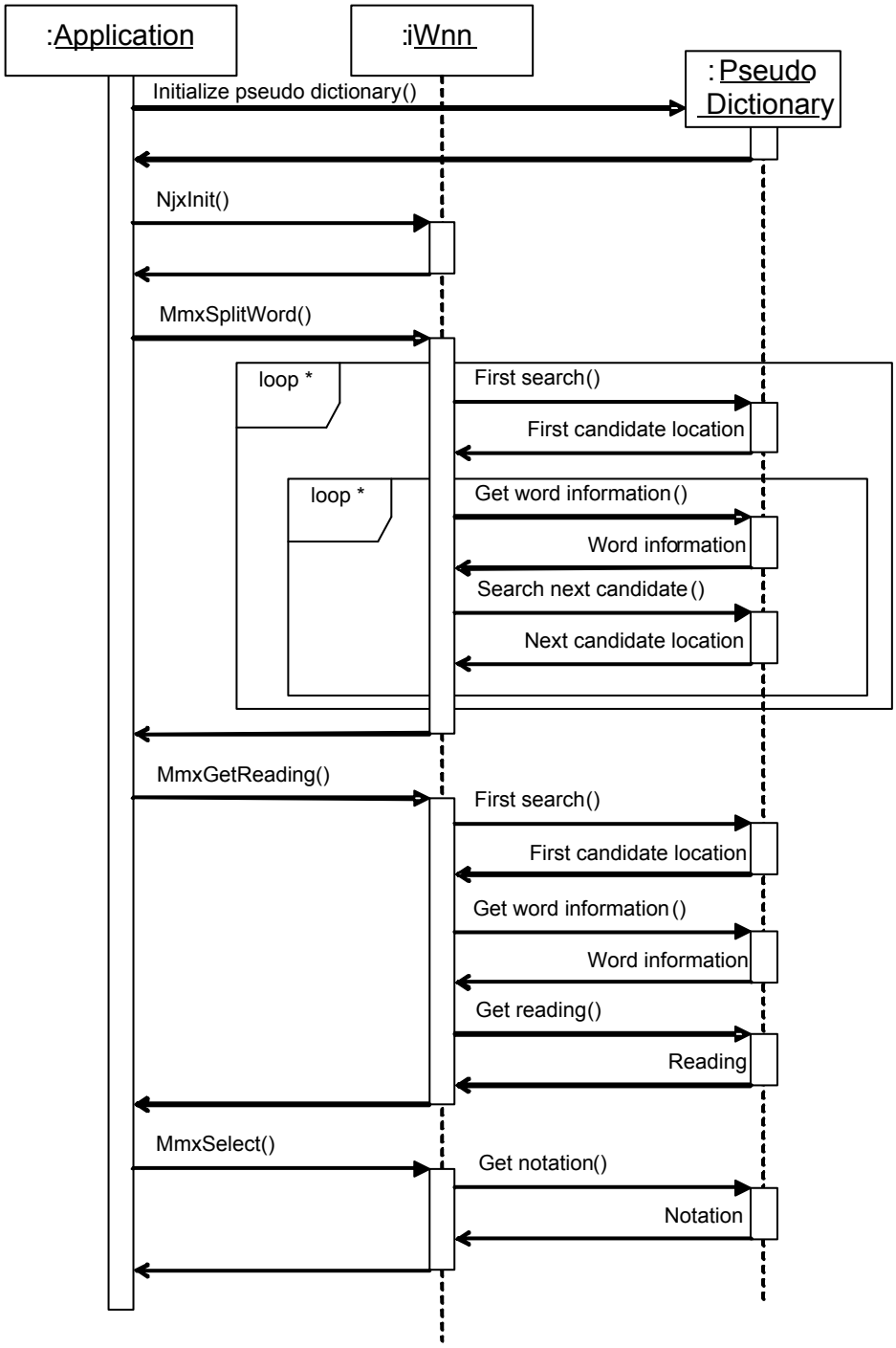
The flow from Prediction Conversion to Candidate Confirmation is illustrated below (same for Regular Conversion and Get All Candidates).

Figure 11-1 Flow From Prediction Conversion to Candidate Confirmation



The flow from Morphological Analysis to Morphological Learning is illustrated below.

Figure 11-2 Flow from Morphological Analysis to Morphological Learning



12 Candidate/Dictionary Lookup Filtering

12.1 Overview

With iWnn, process results output for prediction, multiple phrase conversion, get all candidates, dictionary search, and morphological analysis can be filtered to exclude certain candidates.

For example, filtering can be used for the following applications.

- To exclude candidates that include strings not allowed by applications.
  This includes filtering so that half-width katakana candidates are unusable, when creating e-mail or making pictographs unusable in file name input fields.
- To limit the length of candidate strings.
  This includes getting only candidates that fit inside the character count limitations, when entering text into input fields that have a character limit.

To use filters, set a function pointer to the filter function in the option settings structure (`IWNN_OPTION`) and enable it by executing the set options function (**`NjxSetOption`**) or the initialize function (**`NjxInit`**). iWnn calls the filter function once per candidate. The filter function determines whether or not each candidate passed from iWnn should be used, and notifies iWnn with its return value.

There are two types of filter: a dictionary lookup filter and a candidate filter. The phase for executing these two types of filters differs as follows.

Table 12-1 Phase for Executing Dictionary Lookup Filter and Candidate Filter

Filter	Phase in Which to Execute
Lookup Filter	Filter used during the dictionary search stage. By excluding candidates using a filter at this stage, processing continues with excluded candidates treated internally by iWnn as if they are not registered in the dictionary. This is used in cases such as when excluding a certain character type.
Candidate Filter	Filter used after phrase information has been created. With iWnn, additional processing such as for ancillary words is performed after dictionary lookup and the creation of phrase information. After that, the candidate filter is called to determine whether or not a given result is to ultimately be used as a candidate. This is used in cases such as when you want to limit the length of candidate strings. A candidate filter cannot be used on the first candidate returned by the kana-kanji conversion function (**`NjxConversion`**), split word function (**`MmxSplitWord`**), or get all candidates function (**`NjxAllCandidates`**).

Macro functions required to create filter functions are defined in `nj_lib.h`, `nj_ext.h`, `njfilter.h`, and `iwnn.h`. Be sure to include these files when creating a filter function.

12.2 Dictionary Lookup Filter Interface (IWNN_PHASE1_FILTER_IF)

The interface function of connecting a lookup filter with iWnn is defined as follows.

Code 12-1 Lookup Filter Interface Function (IWNN_PHASE1_FILTER_IF) Definition

```
typedef s16 (*IWNN_PHASE1_FILTER_IF)\
       (IWNN_CLASS* iwnn, IWNN_PHASE1_FILTER_MESSAGE *message);
```

Table 12-2 Lookup Filter Interface Function (IWNN_PHASE1_FILTER_IF) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CLASS* iwnn	Analysis information class. Stores information internal to the iWnn engine. **Note:** Data is not usually written to iwnn.
IN	IWNN_PHASE1_FILTER_MESSAGE* message	Dictionary lookup filter message. This includes candidate information to be filtered. **Note:** For details, see section 12.3 Dictionary Lookup Filter Messages (IWNN_PHASE1_FILTER_MESSAGE).

Table 12-3 Lookup Filter Interface Function (IWNN_PHASE1_FILTER_IF) Return Values

Return Value	Description
s16	Other than 0: Use candidate. 0: Do not use candidate.

Table 12-4 Lookup Filter Interface Function (IWNN_PHASE1_FILTER_IF) Errors

Error Code	Description
None	There are no error codes for IWNN_PHASE1_FILTER_IF.

12.3 Dictionary Lookup Filter Messages (IWNN_PHASE1_FILTER_MESSAGE)

Code 12-2 Dictionary Lookup Filter Messages (IWNN_PHASE1_FILTER_MESSAGE) Structure Configuration

```
struct IWNN_PHASE1_FILTER_MESSAGE {
 IWNN_SEARCH_CONDITION* condition;   // Search conditions
 wchar_t* stroke;              // Reading string buffer
 wchar_t* string;              // Notation string buffer
 s16 strokeLen;                // Reading string length
 s16 strLen;                         // Notation string length
 IWNN_RESULT* result;          // Word information
```

```
 void* option;                  // Filter option
};
```

Table 12-5 Dictionary Lookup Filter Messages (`IWNN_PHASE1_FILTER_MESSAGE`) Structure Members

Member	Description
`IWNN_SEARCH_CONDITION* condition`	Search conditions. Includes information such as the search method (forward lookup prefix match search, derived search, and so forth), the reading string to be searched, and the order in which to return candidates (reading order or frequency order). Usually not used.
`wchar_t* stroke`	Reading string. Represents the reading string of the word. Sometimes this string is not terminated by a `NULL` character.
`wchar_t* string`	Candidate string. Represents the candidate string of the word. Sometimes this string is not terminated by a `NULL` character.
`s16 strokeLen`	Reading string length. Specifies the reading string length (number of array elements).
`s16 strLen`	Candidate string length. Specifies the candidate string length (number of array elements).
`IWNN_RESULT* result`	Word information. Stores information such as part of speech and frequency level. Normally not used.
`void* option`	Dictionary lookup filter option. Represents the dictionary lookup filter option specified by option settings (`IWNN_OPTION`). This can be defined and used indepenedently for each filter relationship.

12.4 Candidate Filter Interface (IWNN_PHASE2_FILTER_IF)

This interface function connects a candidate filter with iWnn. It is defined as follows.

Code 12-3 Candidate Filter Interface Function (IWNN_PHASE2_FILTER_IF) Definition

```
typedef s16 (*IWNN_PHASE2_FILTER_IF) \
       (IWNN_CLASS* iwnn, IWNN_PHASE2_FILTER_MESSAGE *message);
```

Table 12-6 Candidate Filter Interface Function (IWNN_PHASE2_FILTER_IF) Arguments

Input/ Output	Argument	Description
IN/ OUT	IWNN_CLASS* iwnn	Analysis information class. Stores internal information of the iWnn engine. **Note:** Data is not usually written to iwnn.
IN	IWNN_PHASE2_FILTER_MESSAGE* message	Candidate filter message. This includes candidate information to be filtered. **Note:** For details, see section 12.5 Candidate Filter Messages (IWNN_PHASE2_FILTER_MESSAGE).

Table 12-7 Candidate Filter Interface Function (IWNN_PHASE2_FILTER_IF) Return Values

Return Value	Description
s16	Other than 0: Use candidate. 0: Do not use candidate.

Table 12-8 Candidate Filter Interface Function (IWNN_PHASE2_FILTER_IF) Errors

Error Code	Description
None	There are no error codes for IWNN_PHASE2_FILTER_IF.

12.5 Candidate Filter Messages (IWNN_PHASE2_FILTER_MESSAGE)

Code 12-4 Candidate Filter Message (IWNN_PHASE2_FILTER_MESSAGE) Structure Configuration

```
struct IWNN_PHASE2_FILTER_MESSAGE {
 IWNN_RESULT* result;                    // Phrase information
 void* option;                       // Filter option
};
```

Table 12-9 Candidate Filter Message (`IWNN_PHASE2_FILTER_MESSAGE`) Structure Members

Member	Description
`IWNN_RESULT* result`	Phrase information. Includes information such as the phrase's reading string, candidate string, independent/ancillary word part of speech, and frequency. As with the process result structure obtained by the prediction and multiple phrase conversion processes, iWnn functions can be used to get the reading string and candidate string.
`void* option`	Candidate filter option. Represents the candidate filter option specified by option settings (`IWNN_OPTION`). This may be freely defined and used for each filter function.

13 Standard Extension Module

13.1 Overview

Equipped with basic functions in the core, independent of the use environment (the language being used, embedded mobile device, and so forth), iWnn is designed so that functions dependent on the environment can be added as extension modules. Functions dependent on the environment are, therefore, provided as a standard extension module, even if they are required as standard by the text input system.

13.2 Standard Pseudo Candidate Dictionary Module (`NjexPseudoDic`)

13.2.1 Dictionary Overview

Pseudo dictionaries are used to generate standard pseudo candidates.

The standard pseudo candidate dictionary executes candidate generation processes (such as conversion from hiragana to katakana and conversion from number readings to numeric characters) that cannot be implemented using a standard, static dictionary. When performing conversion/prediction for Japanese, always set this dictionary type in the dictionary set structure.

External functions are defined in `iwnn.h`.

When using a standard pseudo candidate dictionary, the following types of pseudo candidates can be generated.

Table 13-1 Pseudo Candidate Types

Pseudo Candidate Type	Description
`NJ_TYPE_HIRAGANA`	Candidates with a notation the same as the reading. For example: “あいうえお” → ”あいうえお”
`NJ_TYPE_KATAKANA`	Full-width katakana candidates. For example: “あいうえお” → ”アイウエオ”
`NJ_TYPE_HALF_KATAKANA`	Half-width kana candidates. For example: “あいうえお” → “ｱｲｳｴｵ” This is not needed for models that do not allow for half-width kana candidate input.
`NJ_TYPE_FULL_KANJI_NUMERIC_GRADE_READING`	Kanji candidates based on the kana reading. For example: “いちまんごせん” → ”一万五千”
`NJ_TYPE_FULL_NUMERIC_READING`	Full-width Arabic number candidates based on the kana reading. For example: “いちまんごせん” → ”１５０００”

Pseudo Candidate Type	Description
NJ_TYPE_HALF_NUMERIC_READING	Half-width Arabic number candidates based on the kana. For example: “いちまんごせん” → ” 15000 “
NJ_TYPE_FULL_KANJI_NUMERIC_READING	Kanji number candidates (including 〇) based on the kana reading. For example: “いちまんごせん” → ”一五〇〇〇”

Whether to use or not to use each pseudo candidate and settings for priority level are defined in the IWNN_PSEUDO_SET structure. The pseudo dictionary work area for the standard pseudo candidate dictionary module is specified for the dictionary set (IWNN_DIC_SET and IWNN_DIC_INFO). If NULL is specified for the pseudo dictionary work area, the default setting is used.

Candidates such as alphanumeric pseudo candidates and date/time pseudo candidates, that can be generated by Advanced Wnn, can also be used, as usual, by customizing the conversion table according the key assignments of the terminal. For details, contact Nintendo support (support@noa.com).

13.2.2 IWNN_PSEUDO_SET Structure

Code 13-1 IWNN_PSEUDO_SET Structure Configuration

```
struct IWNN_PSEUDO_SET {
 s16 count;              // Number of pseudo candidates to be set
 u8 type[NJ_PSEUDO_SET_MAX];   // Type of pseudo candidates to be generated
}
```

Table 13-2 IWNN_PSEUDO_SET Structure Members

Member	Description
s16 count	Number of pseudo candidates to be set. Sets the number of pseudo candidates stored in type[].
u8 type	Type of pseudo candidates to be generated. Stores the pseudo candidate type to be generated in order of highest priority.

The default setting is as follows.

```
static IWNN_PSEUDO_SET pseudoSet = {
 7,
       {
       NJ_TYPE_FULL_KANJI_NUMERIC_GRADE_READING,
       NJ_TYPE_FULL_NUMERIC_READING,
       NJ_TYPE_HALF_NUMERIC_READING,
       NJ_TYPE_FULL_KANJI_NUMERIC_READING,
       NJ_TYPE_HIRAGANA,
       NJ_TYPE_KATAKANA,
       NJ_TYPE_HALF_KATAKANA,
 }
};
```

13.3 Standard Filtered Prediction Search Dictionary Module (NjexPredictionPseudoDic)

This pseudo dictionary is for using candidates obtained when calling the get prediction candidate function without a reading (reading = “”), during filtered relationship prediction searches.

The standard filtered prediction search dictionary performs operations that allow the use of multiple phrase relationship prediction results that cannot be used during a usual filtered relationship prediction search.

External functions are defined in `iwnn.h`.

13.4 Mixed Number Conversion Dictionary Module (NjexNumericCharPseudoDic)

This pseudo dictionary is used to correctly convert a string such as “12 がつ” to “12 月”, by appropriately identifying counters included in reading strings with mixed numbers.

External functions are defined in `iwnn.h`.

13.5 Number Relationship Prediction Dictionary Module (NjexNumericForecastPseudoDic)

This pseudo dictionary is used to generate counters as no reading prediction candidates, immediately after a number string has been confirmed, so that “月” and ”日” appearing as candidates after “12” is confirmed and ”日” appearing as a candidate after “ 13 ” is confirmed. A suitable prediction candidate is generated based on information such as the range of values in which the confirmed number falls and words previously confirmed.

External functions are defined in `iwnn.h`.

Appendix A Character Type Definitions

Table A-1 Definition of Hiragana Characters

	ぁ	あ	ぃ	い	ぅ	う	ぇ	え	ぉ	お	か	が	き	ぎ	く
ぐ	け	げ	こ	ご	さ	ざ	し	じ	す	ず	せ	ぜ	そ	ぞ	た
だ	ち	ぢ	っ	つ	づ	て	で	と	ど	な	に	ぬ	ね	の	は
ば	ぱ	ひ	び	ぴ	ふ	ぶ	ぷ	へ	べ	ぺ	ほ	ぼ	ぽ	ま	み
む	め	も	ゃ	や	ゅ	ゆ	ょ	よ	ら	り	る	れ	ろ	ゎ	わ
ゐ	ゑ	を	ん												

Table A-2 Definition of Katakana Characters

	ァ	ア	ィ	イ	ゥ	ウ	ェ	エ	ォ	オ	カ	ガ	キ	ギ	ク
グ	ケ	ゲ	コ	ゴ	サ	ザ	シ	ジ	ス	ズ	セ	ゼ	ソ	ゾ	タ
ダ	チ	ヂ	ッ	ツ	ヅ	テ	デ	ト	ド	ナ	ニ	ヌ	ネ	ノ	ハ
バ	パ	ヒ	ビ	ピ	フ	ブ	プ	ヘ	ベ	ペ	ホ	ボ	ポ	マ	ミ
ム	メ	モ	ャ	ヤ	ュ	ユ	ョ	ヨ	ラ	リ	ル	レ	ロ	ヮ	ワ
ヰ	ヱ	ヲ	ン	ヴ	ヵ	ヶ									

Table A-3 Half-width Katakana Characters

	ｧ	ｱ	ｨ	ｲ	ｩ	ｳ	ｪ	ｴ	ｫ	ｵ	ｶ	ｶﾞ	ｷ	ｷﾞ	ｸ
ｸﾞ	ｹ	ｹﾞ	ｺ	ｺﾞ	ｻ	ｻﾞ	ｼ	ｼﾞ	ｽ	ｽﾞ	ｾ	ｾﾞ	ｿ	ｿﾞ	ﾀ
ﾀﾞ	ﾁ	ﾁﾞ	ｯ	ﾂ	ﾂﾞ	ﾃ	ﾃﾞ	ﾄ	ﾄﾞ	ﾅ	ﾆ	ﾇ	ﾈ	ﾉ	ﾊ
ﾊﾞ	ﾊﾟ	ﾋ	ﾋﾞ	ﾋﾟ	ﾌ	ﾌﾞ	ﾌﾟ	ﾍ	ﾍﾞ	ﾍﾟ	ﾎ	ﾎﾞ	ﾎﾟ	ﾏ	ﾐ
ﾑ	ﾒ	ﾓ	ｬ	ﾔ	ｭ	ﾕ	ｮ	ﾖ	ﾗ	ﾘ	ﾙ	ﾚ	ﾛ	ﾜ	ﾜ
		ｦ	ﾝ	ｳﾞ											

Note: Although voiced and semi-voiced sounds are listed in the same table locations to make the correspondence with full-width characters easier to see, there are no voiced or semi-voiced characters in the case of half-width katakana. For half-width katakana, voiced and semi-voiced sounds are indicated by using the pure sound character in combination with a separate symbol to indicate voicing (゛) or semi-voicing (゜).

Appendix B Frequently Asked Questions

B.1 FAQ Regarding Characters and Character Strings

B.1.1 Q: What range of character codes can be processed?

All codes can be used except the terminator code (0x0000 when using UTF-16BE).

For coding methods, such as for sequence codes, refer to the *Custom Dictionary Creation Tool Manual*.

B.1.2 Q: How do I register pictograph characters specific to mobile telephones?

If pictograph characters cannot be registered in a dictionary under the development environment (such as when a pictograph code outside the code range is assigned, or when there is no font including pictographs available in the development environment), use octal notation to express the pictograph code to register the pictograph in the dictionary.

Create dictionaries according to the I/O code used by the system. For an example, see the following table.

Table B-1 Octal Notation to Express Pictograph Code to Register the Pictograph in the Dictionary

Reading	Candidate (Pictograph)	Part of Speech	Frequency
かお	\366\131	Symbol	0

For coding methods, such as those for sequence codes, refer to the *Custom Dictionary Creation Tool Manual*.

B.1.3 Q: How do I register the ‘ \ ’ (backslash) character?

Convert the ‘ \ ’ code to octal, to register the backslash character.

Create dictionaries according to the I/O code used by the system. For an example, see the following table.

Table B-2 Octal Notation to Register the Backslash Character

Reading	Candidate (Pictograph)	Part of Speech	Frequency
えん	\134	Symbol	0

For coding methods, such as those for sequence codes, refer to the *Custom Dictionary Creation Tool Manual*.

B.1.4 Q: How do I use lines to indicate elongated vowels?

Elongated lines such as "—" are not included in auxiliary characters. To handle this exactly like vowel elongation, the character type identification process must be changed.

B.2 FAQ Regarding Dictionaries

B.2.1 Q: What is the effect of the number and size of dictionaries in the dictionary set?

If the size of a dictionary is doubled, the speed of a dictionary search decreases by 10 to 20%.

If the number of dictionaries of the same size is doubled, the speed of a dictionary search is two times slower.

Although increasing dictionaries containing more than several tens of thousands of words affects the conversion process speed, increasing dictionaries containing a few hundred words has only a slight impact on the conversion processing speed.

B.2.2 Q: What about various sizes of custom dictionaries?

The resulting sizes when creating various compressed custom dictionaries from a 45,000-word conversion dictionary are listed in the table below.

Table B-3 Various Sizes of Custom Dictionaries

Dictionary Format	Size
AdvancedWnn V1.1 Compressed custom dictionary (forward lookup complete match dictionary)	615 KB
Forward lookup complete match compressed custom dictionary	511 KB (83.11% compared to V 1.1)
Forward lookup prefix match compressed custom dictionary	635 KB (103.19% compared to V 1.1)
Reverse lookup complete match compressed custom dictionary	801 KB (130.19% compared to V 1.1)

B.2.3 Q: Can the maximum number of mountable dictionaries be set to 21 or more?

A value of up to 100 can be set for this parameter.

Note, however, that you can expect slightly poorer performance in terms of processing speed as this value increases.

Performance and other tests have been performed for iWnn with a maximum value of 20.

B.3 FAQ Regarding Processing Methods

B.3.1 Q: Are there any precautions for using the associative learning flag with the add word function?

There are two functions for executing learning in the learning dictionary: the learn function and the add word function.

Since the associative learning flag for the learning and add word functions represents connection information versus previously learned phrases, alternating use of these functions will result in associative learning with unexpected results.

For example, when adding “関係” and ”学習” to the learning dictionary to learn the relationship between them, the reading “かんけい” is converted on the registration screen and “関係” is input into the input field.

1. At this time, learning information of the following type is registered in the learning dictionary when learned using the learning function.

Table B-4 Learning Information Registered in the Learning Dictionary by the Learning Function

Reading	Candidate	Relationship Information
かんけい	関係	No

2. “関係” is registered in the learning dictionary using the add word function.

Table B-5 Pictograph Code Registered in the Dictionary by Using the Add Word Function

Reading	Candidate	Relationship Information
かんけい	関係	No
かんけい	関係	No

3. The reading “がくしゅう” is converted on the registration screen and “学習” is input in the input field. At this time, learning information of the following type is registered in the learning dictionary when learned using the learning function.

Table B-6 Learning Information Is Registered in the Learning Dictionary

Reading	Candidate	Relationship Information
かんけい	関係	No
かんけい	関係	No
がくしゅう	学習	No

4. ”学習” is registered in the learning dictionary using the add word function.

Table B-7 Pictograph Code Connected Through Associative Learning

Reading	Candidate	Relationship Information
かんけい	関係	No
かんけい	関係	No
がくしゅう	学習	No
がくしゅう	学習	Yes

In this case, “学習” and “学習” are connected through associative learning.

When using the associative learning flag of the add word function, be sure to add words to the learning dictionary using only the add word function.

Because functions within the process result structure expiration period and provision range may affect the learning dictionary and user dictionary, be careful when using the learning function and add word function in conjunction with each other.

B.3.2 Q: How do I obtain a prediction candidate consisting of the same string as the input string?

Get the candidate string using the get candidate string function and compare it to the input string.

B.3.3 Q: What data should be saved?

The learning dictionary and user dictionary must be saved.

Memory is allocated for these libraries by the application, and specified in the dictionary set. Save these allocated memory regions as is.

B.3.4 Q: How do I implement an English-kana conversion feature (such as where “かさた” converts to “adg”)?

An English-kana conversion feature for converting a string such as “かさた” to “adg”, “234“, or “2 時 34 分”, according to the characters assigned to the ten-key keypad, is widespread among international mobile telephones when entering text.

This type of conversion can be supported when using iWnn by creating a pseudo dictionary.

Although this is not provided as a standard feature because key assignments vary depending on the model, sample source code for a pseudo dictionary that implements English-kana conversion is available.

Contact Nintendo support (support@noa.com) for details on changing the key table.

B.3.5 Q: How do I implement a function equivalent to the pseudo candidate function under Advanced Wnn Ver. 2.3?

Under iWnn, the function for creating pseudo candidates has been moved to the extension module. Although pseudo candidate-related functions found in Advanced Wnn have been deleted along with this move, equivalent functions can be implemented as given below.

B.3.5.1 Set Pseudo Candidate Set Function (`nj_set_gijiset`)

When the standard pseudo candidate dictionary module is mounted to the dictionary set structure, it can be substituted by setting the pseudo candidate set structure (`IWNN_PSEUDO_SET`) in the pseudo dictionary work area.

`IWNN_PSEUDO_SET` is defined under Advanced Wnn in the standard pseudo candidate dictionary module.

B.3.5.2 Get Pseudo Candidate Function (`nj_get_giji`)

Use of the get pseudo candidate function (**`NjexGetPseudo`**), compatible with Advanced Wnn Ver. 2.3, is included in the standard pseudo candidate dictionary module.

B.3.6 Q: How do I reduce registration speed when batch registering to the learning dictionary?

Processing time can be reduced by suppressing optimization performed each time a word is registered in the learning dictionary. Be sure to specify `NJ_ADD_WORD_OPTIMIZE_OFF` in the `extensionMode` member. (Refer to section 8.8 Option Settings (IWNN_OPTION).)

If you do so, remove the associated flag before registering any words after the first, and perform optimization of the dictionary only once at the end. Otherwise, processing speed will drop when using the dictionary in question.

B.4 Miscellaneous FAQ

B.4.1 Q: Does iWnn support multi-thread processing?

As long as writable dictionaries are not being shared, processes such as kana-kanji conversion, morphological analysis, and dictionary search can be executed as multiple threads.

Because exclusive write control is not supported for learning dictionaries and user dictionaries, you must prepare a dictionary region for each thread, in order to use writable dictionaries.

B.4.2 Q: Can I use a prediction conversion dictionary?

iWnn does not support prediction conversion dictionaries.

B.4.3 Q: In what order are candidates obtained by the get prediction candidate function (`NjxAnalyze`)?

Although candidates are arranged in order of prefix match search process result, multiple phrase conversion process result, single phrase conversion process result, and then get all candidates process result, the order of candidates within each process is determined in consideration of relationship among the frequency set for each word, the priority level of the dictionary (dictionary frequency setting), the state setting bias value, and pre-confirmation information.

This also makes it easier to prioritize candidates closest to the input string, when using fuzzy searches.

For example, even if “学校(がっこう)” is recorded in the learning dictionary, once “か” is entered, a candidate that starts with “か”, such as “必ず” will appear as the first candidate.

If you want the priority of candidates to be determined based solely on frequency, rather than how close it is to the input string, be sure to specify `IWNN_OPTION_FORECAST_TOP_FREQUENCY` in the extensionMode member. (Refer to 8.8 Option Settings (IWNN_OPTION).)

Appendix C Dictionary Frequency Settings When Using Multi-lingual Features

C.1 Introduction

This section describes the following, when using the multi-lingual prediction conversion features of iWnn.

- Dictionary frequency settings.
- List of available functions.

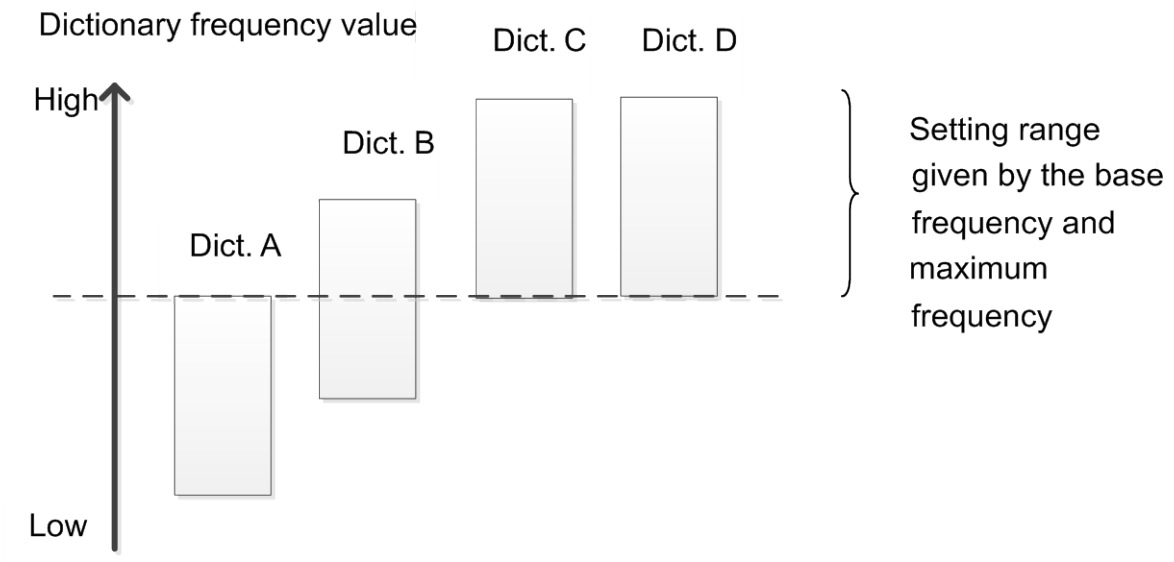
C.2 Dictionary Frequency Settings

These settings allow you to specify the priority relationship among dictionaries according to a dictionary frequency value.

Dictionary frequency values are set as a pair of values representing base frequency and high frequency. iWnn handles the priority of stored words within a specified frequency range.

In addition, dictionary frequencies can be set separately for normal conversion (kana-kanji conversion and get all candidates), prediction conversion, and morphological analysis.

Figure C-1 Dictionary Frequency Setting Example



In the example shown in Figure C-1, words stored in Dictionary C are always prioritized above words stored in Dictionary A. In addition, by overlapping setting ranges as shown for Dictionary A and Dictionary B, you can prioritize only those words stored in Dictionary B that have a higher priority than words in Dictionary A. If the setting ranges for two dictionaries are the same, as shown for Dictionary C and Dictionary D, priority is determined based solely on the frequency information stored for each word.

However, depending on the state setting during connection, prediction, and situational adaptive prediction, the priority of a word may be changed because it lies outside the setting range of its dictionary.

Limitations on Base Frequency/High Frequency by Dictionary Handle

Specify the base frequency and maximum frequency by dictionary handle in the range from 0 to 1000. If the base frequency is greater than the maximum frequency, that dictionary will not be used.

Table C-1 Frequency Settings for English Dictionaries

Dictionary Type	Fuzzy Search	For Conversion		For Prediction		For Morphological Analysis	
		Base	High	Base	High	Base	High
Standard Prediction Dictionary	Yes	100	400	100	400	10	0
No Reading Prediction Dictionary	No	100	244	100	244	10	0
Relationship Prediction Dictionary	Yes	245	245	245	245	10	0
Custom Dictionary	No	0 ~ 400	0 ~ 400	0 ~ 400	0 ~ 400	10	0
User Dictionary	No	500	500	10	0	10	0
Learning Dictionary	Yes	501	1000	501	1000	10	0

For dictionaries not used in any particular mode, set a base frequency of 10 and a maximum frequency of 0 (base frequency less than maximum frequency) to exclude them from use.

Table C-2 Frequency Settings for Korean Dictionaries

Dictionary Type	Fuzzy Search	For Conversion		For Prediction		For Morphological Analysis	
		Base	High	Base	High	Base	High
Conversion Dictionary	No	400	500	10	0	10	0
Standard Prediction Dictionary	No	10	0	100	400	10	0
No Reading Prediction Dictionary	No	10	0	100	400	10	0
Relationship Prediction Dictionary	No	245	245	245	245	10	0
Custom Dictionary	No	0 ~ 400	0 ~ 400	0 ~ 400	0 ~ 400	10	0
Single-Kanji Dictionary	No	0	10	10	0	10	0
User Dictionary	No	500	500	10	0	10	0
Learning Dictionary	No	501	1000	501	1000	10	0

For dictionaries not used in any particular mode, set a base frequency of 10 and a maximum frequency of 0 (base frequency less than maximum frequency) to exclude them from use.

Table C-3 Frequency Settings for Chinese (Simplified) Dictionaries

Dictionary Type	Fuzzy Search	For Conversion		For Prediction		For Morphological Analysis	
		Base	High	Base	High	Base	High
Conversion Dictionary	No	100	400	10	0	10	0
Prediction Conversion Dictionary	No	10	0	100	560	10	0
Custom Dictionary	No	0 ~ 400	0 ~ 400	0 ~ 400	0 ~ 400	10	0
Single-Kanji Dictionary	No	100	400	10	0	10	0
User Dictionary	No	500	500	10	0	10	0
Learning Dictionary	No	501	1000	501	1000	10	0

For dictionaries not used in any particular mode, set a base frequency of 10 and a maximum frequency of 0 (base frequency less than maximum frequency) to exclude them from use.

Table C-4 Frequency Settings for Chinese (Traditional) Dictionaries

Dictionary Type	Fuzzy Search	For Conversion		For Prediction		For Morphological Analysis	
		Base	High	Base	High	Base	High
Conversion Dictionary	No	100	400	10	0	10	0
Prediction Conversion Dictionary	No	10	0	100	560	10	0
Custom Dictionary	No	0 ~ 400	0 ~ 400	0 ~ 400	0 ~ 400	10	0
Single-Kanji Dictionary	No	100	400	10	0	10	0
Cangjie Prediction Dictionary	No	10	0	100	400	10	0
User Dictionary	No	500	500	10	0	10	0
Learning Dictionary	No	501	1000	501	1000	10	0

For dictionaries not used in any particular mode, set a base frequency of 10 and a maximum frequency of 0 (base frequency less than maximum frequency) to exclude them from use.

C.3 List of Usable Functions

iWnn provides the following functions for use during multilingual prediction conversion.

Table C-5 List of Usable Functions (○: Usable, —: Not Usable)

Function Name	English	Korean	Chinese (Simplified)	Chinese (Traditional)
Initialize (**`NjxInit`**)	○	○	○	○
Get reading string (**`NjxGetStroke`**)	○	○	○	○
Get candidate string (**`NjxGetCandidate`**)	○	○	○	○
Get dictionary handle (**`NjxGetDicHandle`**)	○	○	○	○
Create dictionary region (**`NjxCreateDic`**)	○	○	○	○
Check dictionary (**`NjxCheckDic`**)	○	○	○	○
Get character type (**`NjxGetCharType`**)	○	○	○	○
Change dictionary type (**`NjxChangeDicType`**)	○	○	○	○
Get prediction candidate (**`NjxAnalyze`**)	○	○	○	○

Function Name	English	Korean	Chinese (Simplified)	Chinese (Traditional)
Kana-kanji conversion (**NjxConversion**)	○	○	○	○
Get all candidates (**NjxAllCandidates**)	○	○	○	○
Learn (**NjxSelect**)	○	○	○	○
Undo learning (**NjxUndo**)	○	○	○	○
Search word (**NjxSearchWord**)	○	○	○	○
Get word (**NjxGetWord**)	○	○	○	○
Add word (**NjxAddWord**)	○	○	○	○
Delete word (**NjxDeleteWord**)	○	○	○	○
Split words (**MmxSplitWord**)	—	—	—	—
Get part of speech group (**MmxGetPartsOfSpeech**)	—	—	—	—
Get reading string for morphological analysis (**MmxGetReading**)	—	—	—	—
Learn by morphological analysis (**MmxSelect**)	—	—	—	—
Set options (**NjxSetOption**)	○	○	○	○
Set state (**NjxSetState**)	—	—	—	—
Get state (**NjxGetState**)	—	—	—	—
Get registered word information (**NjxGetWordInfo**)	○	○	○	○
Get no conversion candidate (**NjxGetStrokeWord**)	—	—	—	—
Merge candidate lists (**NjxMergeWordList**)	○	○	○	○
Manage learning dictionary (**NjxManageLearnDic**)	○	○	○	○
Delete word (**NjxDeleteWord**)	○	○	○	○

Appendix D Notes

This appendix gives a supplemental description of sections already described in this manual and special terminology.

D.1 Defined Values Set at Compile Time (Data Types)

Data types listed in section 3.1 Defined Values Set at Compile Time cannot be changed by the user.

D.2 Defined Values Set at Compile Time (Defined Values)

Except for some parameters, the maximum, minimum, and default values for defined values listed in section 3.1 Defined Values Set at Compile Time cannot be changed by the user.

The default values for the following defined values can be changed.

- Maximum number of registerable words in a user dictionary (`NJ_MAX_USER_COUNT`).
- Maximum number of registerable fuzzy characters (`NJ_MAX_CHARSET`).

D.3 Creating Dictionaries

The phrase “when power is turned on the first time after shipment from the factory” found in section 7.1 Startup means the same thing as “the first time an application that uses this middleware is started.”

D.4 Cold Start/Hot Start

The following are descriptions of a cold start and a hot start.

- Cold start.
  The act of starting a device in which software is embedded from a completely power-off state.
- Hot start.
  The act of starting a device in which software is embedded using a software reset, that omits some hardware checks.

D.5 FLASH Dictionaries/Non- FLASH Dictionaries

The following are descriptions of a FLASH dictionary and a non-FLASH dictionary.

- Non-FLASH dictionary.
  Indicates a dictionary where all dictionary data is deployed in memory.
- FLASH dictionary.
  Indicates a dictionary where only some dictionary data is deployed in memory, in order to conserve memory being used by the application. Only an integrated dictionary can be used as a FLASH dictionary.

D.6 Automatic Learning When Replying to E-mail

Use of the automatic learning feature when replying to e-mail, described in section 7.12 Automatic Learning When Replying to E-mail (Learning Morphological Analysis Results), assumes that the hardware device in question is equipped with e-mail capability.

Revision History

Version	Revision Date	Category	Description
1.0.9	2013/04/11	Changed	• 9.4 Create Dictionary Function (**`NjxCreateDic`**) Added information on the number of learned phrases that can be used.
1.0.8	2012/12/21	Added	• 3.3 Access to Dictionary Files (OnMemory Version Only) Added description of OnMemory operations.
1.0.7	2012/03/22	Changed	• 3.2 Including Header Files Corrected the filename to `iwnnCTR.h`. • 8.3 Dictionary Frequency Value Settings Added that using the dictionary with the setting of the base frequency greater than the maximum frequency for a word search function results in an error.

1.0.6	2010/10/04	Changed	• Miscellaneous
			Changed the document version from 1.0.5 to 1.0.6.
			• Overall
			Corrected distorted fonts in figures.
			• 3.1 Defined Values Set at Compile Time
			Added a note about changing variable types.
			Added a note about changing defined values.
			7.1 Startup
			Added a note about cold starts and hot starts.
			Added a note about FLASH dictionaries and non-FLASH dictionaries.
			Added a note about the timing for creating a dictionary region in memory.
			• 7.12 Automatic Learning When Replying to E-mail (Learning Morphological Analysis Results)
			Added a note on the anticipated use of the learning function when replying to e-mail.
			• 9.4 Create Dictionary Function (**`NjxCreateDic`**)
			Changed the setting value for the type argument of the `NjxCreateDic` function from a range of 0 to 4 to any of the following values.
			`NJ_CREATE_DIC_TYPE_USER`
			`NJ_CREATE_DIC_TYPE_LEARN_AWNN`
			`NJ_CREATE_DIC_TYPE_LEARN`
			`NJ_CREATE_DIC_TYPE_USER_ADDITIONAL`
			`NJ_CREATE_DIC_TYPE_LEARN_ADDITIONAL`
			• 17 Appendix D: Notes
			Newly added as a notes section.

1.0.5	2010/09/03	Changed	• Miscellaneous Changed the document version from 1.0.4 to 1.0.5. • Overall Corrected a problem of styles missing from the style guide. Removed the interface for frequency learning.

Version	Revision Date	Category	Description
1.0.4	2010/07/30	Changed	• Miscellaneous Changed the document version from 1.0.3 to 1.0.4. Changed the document name to"iWnn Programming Manual". (Deleted "memory savings version" from the title.) Added section numbers to PDF bookmarks. Deleted a description of ShiftJIS character codes. Deleted unnecessary broken lines due to by disabling MS-Word spell checking. Corrected a problem of styles missing from the style guide. Integrated this document with the separate document titled"iWnn Appendix". • 2.7 Complete Match Searches (Forward/Reverse Lookup), Prefix Match Searches (Forward Lookup) and Relationship Searches Changed the section title • 2.8 Various Dictionaries (Integrated, Single Kanji, Ancillary Word, User, Learning, Rule, Custom, No Reading Prediction) Changed the section title. • 3 Notes on Using iWnn Changed the section title. • 5 Expiration Period and Applicable Scope of the Process Result Structure Changed the section title. Changed the Japanese term "使用期限" to"有効期限". • 7.2 From Getting Candidates to Learning Corrected the omission of words from "Figure 7-2 From Getting Candidates to Learning". • 8.7 fuzzy character set (`IWNN_CHARSET`) Made revisions to the example of a fuzzy character set structure. • 9.4 Create Dictionary Function (**`NjxCreateDic`**) Changed the examples of calculating sizes for the user dictionary and learning dictionary to use UTF-16BE. • 9.24 Get State Setting Function (**`NjxGetState`**) Corrected the omission of words from "Code 9-27 Get State Setting Function (**`NjxGetState`**) Declaration". • 11.1 Overview Removed the unnecessary example of a pseudo dictionary.
1.0.3	2010/06/30	Changed	• Miscellaneous Changed the document version from 1.0.0 to 1.0.3. • 7.1 Startup Added how to distinguish a non-FLASH dictionary from a FLASH dictionary. • 3.1 Defined Values Set at Compile Time Corrected the format of the description of `NJ_MAX_USER_COUNT`.
1.0.0	2010/05/07	Changed	• Miscellaneous Changed the document version from 0.3.1 to 1.0.0. Corrected typos.

Version	Revision Date	Category	Description
0.3.1	2010/04/05	Changed	• 11.5 Pseudo Dictionary Processing Specifications Made revisions according to coding conventions. ・Changed “loct” to “location.” ・Changed “rhandle” to “ruleHandle.” • 13 Standard Extension Module Changed each section title so the module name appears in parentheses. • 13.2 Standard Pseudo Candidate Dictionary Module (`NjexPseudoDic`) Changed from “CellBodyBullet” style to the “CellBody” style in Table 13 1 Pseudo Candidate Types • Miscellaneous Made revisions for supporting the document style guides of developers. ・Support for ”8.3 Using Numbers”. ・Support for ” 3.12.3 Between Full-width and Half-width”. Revised typos.
0.3.0	2010/03/31	Changed	• Miscellaneous Applied styles to the entire document.
		Added	• 18 Revision History Newly added. • Colophon • Newly added.
0.2.0	2010/03/19	-	Initial Version.