Technical Note

# Basis Sentence Breaking Libraries

Version 10.10

# Technical Note

Revision 0

This document describes how to use the HP Autonomy Basis Sentence Breaking Libraries 10.10.

The Basis Rosette Linguistics Platform libraries allow you to use the Basis systems for sentence breaking and stemming in IDOL Server. This library takes the place of the standard sentence breaking libraries that Autonomy provides with IDOL Server for Chinese, Japanese, and Korean.

You can use the Basis libraries in the package to perform sentence breaking in your IDOL Server. You can also optionally create a custom dictionary to specify custom sentence breaking rules for particular words and phrases.

**Contents**




Copyright © 2015 Hewlett-Packard Development Company, L.P.

# Supported Platforms

The following operating system platforms are supported by Basis Sentence Breaking Libraries10.10.

- Linux
- Windows

The following sections provide more information about the supported versions of these platforms.

**Linux (64-bit only)**

- glibc 2.5

**Windows (64-bit and 32-bit)**

- Windows 7
- Windows 8
- Windows Server 2008
- Windows Server 2012

# Install the Basis Libraries

The Basis zip package contains the library files for Chinese, Japanese, and Korean, along with the required supporting files and dictionaries.

**To install the Basis libraries on your IDOL Server instance**

- Extract the contents of the `langfiles` directory in the zip package to the `langfiles` directory in your IDOL Server installation.

# Environment Variables

To use the Basis RLP, you must set up the following environment variables on the machine:

- BT_ROOT. The Root directory that contains the RLP SDK. This is the langfiles directory where you extracted the Basis zip package.
- BT_BUILD. The platform that the script is running on. Set this variable to the appropriate value from the following table:

| Platform | BT_BUILD Value |
|---|---|
| 64-bit Linux | amd64-glibc25-gcc41 |
| 64-bit Windows | amd64-w64-msvc110 |
| 32-bit Windows | ia32-w32-msvc110 |

You must also configure the LD_LIBRARY_PATH (for Linux platforms), or PATH (for Microsoft Windows platforms).

On Linux, you must set:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BT_ROOT/rlp/bin/$BT_BUILD:$BT_
ROOT/rlp/lib/$BT_BUILD
```

On Windows you must set:

```
set PATH=%PATH%;%BT_ROOT%\rlp\bin\%BT_BUILD%
```

# Configuration

To use the Basis Rosette Linguistics Platform, you must configure a context file for each sentence breaking library, and you must configure IDOL Server to use the sentence breaking libraries.

# RLP Configuration

You can configure the Basis Rosette Linguistics Platform separately for each sentence breaking library by using a context file. The following example shows a context file for Chinese sentence breaking:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!DOCTYPE contextconfig PUBLIC
      "-//basistech.com//DTD RLP Context Config 7.1//EN"
      "urn:basistech.com:7.1:contextconfig.dtd">
<contextconfig>
   <properties>
      <!-- When a word is in the system dictionary use user dictionary -->
      <property name="com.basistech.cla.favor_user_dictionary" value="true"/>
      <property name="com.basistech.cla.decomposecompound" value="true"/>
   </properties>
   <languageprocessors>
      <languageprocessor>Unicode Converter</languageprocessor>
      <languageprocessor>Language Identifier</languageprocessor>
```

```
      <languageprocessor>Encoding and Character Normalizer</languageprocessor>
      <languageprocessor>Sentence Breaker</languageprocessor>
      <languageprocessor>Script Region Locator</languageprocessor>
      <languageprocessor>Chinese Language Analyzer</languageprocessor>
      <languageprocessor>Word Breaker</languageprocessor>
      <languageprocessor>Lemmatizer</languageprocessor>
   </languageprocessors>
</contextconfig>
```

You must place the context file in the `langfiles` directory, with the file name `chinese-context.xml`, `japanese-context.xml`, or `korean-context.xml` (as appropriate). For more information about how to configure the Basis RLP, see "Appendix: Basis RLP " on page 9.

# IDOL Server Language Configuration

To use the Basis RLP libraries in IDOL Server, you must update the language configuration section for the appropriate language.

**To configure IDOL Server to use the Basis Sentence Breaking library**

1. In the appropriate language configuration section of the configuration file, set the `SentenceBreaking` parameter to the name of the Basis library that you want to use. For example:

   ```
   SentenceBreaking=japanesebreaking_basis
   ```

2. Set the `SentenceBreakingOptions` parameter to a comma-separated list of the options that you want to use. The following options are available for the Basis sentence breaking libraries:

   | Option | Description |
   |---|---|
   | `DecomposeCompound` | For tokens that are a compound of several components, separate the individual components by white space in the output buffer. |
   | `Logging` | Enable logging. Logs are output to the basis-logs directory, to the *`Language`*`.log` file, where *`Language`* is `Chinese`, `Japanese`, or `Korean` (as appropriate). |
   | `Stemming` | Include any stem terms that the Basis RLP provides for a token in the output buffer. The Basis libraries provide stemming for some words in Chinese, Japanese, and Korean. |

   You can also use the standard IDOL Server sentence breaking options. For available options, refer to the *IDOL Server Reference*.

3. Set the other configuration options for the language as usual. For more information about the available configuration options, refer to the *IDOL Server Reference*.

**Tip:** The default IDOL Server configuration file has the `Normalise` option set to `true` for Chinese, which normalizes traditional Chinese characters to simplified. If you want to use a custom dictionary file that contains rules with traditional Chinese characters, you must set `Normalise` to `false` for IDOL Server to use these rules.

**Note:** The `Stemming` configuration parameter in the `[MyLanguage]` or `[LanguageTypes]` configuration section does not have any effect on the `SentenceBreakingOptions=stemming` option. For Chinese, Japanese, and Korean, all the stemming is performed in the sentence breaking libraries, and not in IDOL Server, so the IDOL stemming configuration does not affect the stemming that the Basis library performs.

4. Modify the configuration for any other languages that you want to use the Basis libraries for.
5. Save and close the configuration file. Restart IDOL Server for your changes to take effect.

The following example configuration sets options for Japanese and Chinese:

```
[japanese]
Encodings=UTF8:japaneseUTF8
Stemming=false
SentenceBreaking=japanesebreaking_basis
SentenceBreakingOptions=logging,stemming,decomposecompound

[chinese]
Encodings=UTF8:chineseUTF8
Stemming=false
SentenceBreaking=chinesebreaking_basis
SentenceBreakingOptions=logging
```

# User Dictionaries

You can create custom dictionaries for the Basis RLP libraries, which define how to add white space to compound terms that you define. These user dictionaries are compiled.

**Note:** You must set up the `BT_ROOT` and `BT_BUILD` environment variables before you run the scripts to compile your custom dictionaries. See "Environment Variables" on page 2.

## Create the Dictionary File

The dictionary files contain a list of terms and phrases, and descriptions of how you want the library to add white space.

Each entry that you want to add is a single line, in the following format.

*word*[tab]*POS*[tab]*DecompositionPattern*

where,

| | |
|---|---|
| `[tab]` | The ASCII horizontal tab character with code 9. |
| *word* | The noun to add to the dictionary. |
| *POS* | One of the user-dictionary part of speech tags, listed in the documentation. See "Appendix: Basis RLP " on page 9. |
| *DecompositionPattern* | A comma-separated list of numbers to specify the number of characters from the word to use in each component of the decomposition. You can use a zero (0) to specify that you do not want to split the term.<br><br>**Note:** If you want to use decomposition patterns, you must set the `SentenceBreakingOptions` configuration parameter to include the `DecomposeCompound` option in the IDOL Server configuration file language configuration. In addition, if you supply a custom XML context file, you must not turn off the `decomposecompound` property. For more information, see "RLP Configuration" on page 3, and "Appendix: Basis RLP " on page 9.<br><br>If you use a decomposition pattern of 0, the sentence breaking library does not split the term, even if `DecomposeCompound` is turned on in the IDOL Server configuration file. |

For example:

```
深圳发展銀行   noun    2,4
北京人    noun    0
```

The dictionary files must be UTF-8 encoded. You can add comment lines, by starting a line with a hash (#) character. The libraries ignore any empty lines.

The following sections provide more detail about creating the dictionary file for each language:

- Chinese: "Creating the User Dictionary" on page 21
- Japanese: "Creating the Source File" on page 25
- Korean: "Editing the Dictionary Source File" on page 30

# Compile the Dictionary Files

The following sections describe how to compile the custom dictionaries on Microsoft Windows and Unix operating systems.

## *Compile the Dictionary File on Windows*

**To compile the dictionary file on Microsoft Windows**

1. Ensure that you have set the `BT_BUILD` and `BT_ROOT` environment variables. For details, see "Environment Variables" on page 2.
2. Open a Cygwin command-line window, and change directory to the location of the compilation script:

| Language | Compilation Script |
|---|---|
| Chinese | `BT_ROOT/rlp/cma/source/samples/` |
| Japanese | `BT_ROOT/rlp/jma/source/samples/` |
| Korean | `BT_ROOT/rlp/kma/source/samples/` |

3. Copy the text file containing your user dictionary to the same directory as the compilation script.
4. Run the following command to compile the text file.

   `./build_user_dict.sh` *`CustomDictionary`*`.txt` *`OutputFile`*`.bin`

   where,

   - *`CustomDictionary`* is the file name of your text file.
   - *`OutputFile`* is the name of the compiled dictionary file that you want to create.

   For example:

   ```
   ./build_user_dict.sh user_chinese.txt user_chinese.bin
   ```

## Compile the Dictionary File on Unix

**To compile the dictionary file on Unix platforms**

1. Ensure that you have set the `BT_BUILD` environment variable. For details, see "Environment Variables" on page 2.
2. Open the command-line, and change directory to the location of the compilation script:

| Language | Compilation Script |
|---|---|
| Chinese | `BT_ROOT/rlp/cma/source/samples/` |
| Japanese | `BT_ROOT/rlp/jma/source/samples/` |
| Korean | `BT_ROOT/rlp/kma/source/samples/` |

3. Copy the text file containing your user dictionary to the same directory as the compilation script.

4. `./build_user_dict.sh CustomDictionary.txt OutputFile.bin`

   where,

   - *CustomDictionary* is the file name of your text file.
   - *OutputFile* is the name of the compiled dictionary file that you want to create.

   For example:

   ```
   ./build_user_dict.sh user_chinese.txt user_chinese.bin
   ```

# Enable the Custom Dictionary

After you have created and compiled the custom dictionary, you must move it to the appropriate location, and update the Language Analyzer configuration file.

## *Update the Language Analyzer Configuration File*

After you have compiled the custom dictionary and moved it to the appropriate directory, you must update the appropriate Language Analyzer configuration file.

The following table lists the locations where you must place the compiled dictionary files, and the location of the language analyzer configuration file. When you add a new custom dictionary, you must update this language analyzer configuration file to configure the Basis libraries to use it.

| Language | Dictionary Location | Language Analyzer Configuration File |
|---|---|---|
| Chinese | `BT_ROOT/rlp/cma/dicts/` | `BT_ROOT/rlp/etc/cla-options.xml` |
| Japanese | `BT_ROOT/rlp/jma/dicts/` | `BT_ROOT/rlp/etc/jla-options.xml` |
| Korean | `BT_ROOT/rlp/kma/dicts/` | `BT_ROOT/rlp/etc/kla-options.xml` |

- For Chinese you can add one or more `<dictionarypath>` properties, containing the path to the custom user dictionary. See "Updating the Chinese Language Analyzer Configuration File" on page 24.
- For Japanese, you can add one or more `<DictionaryPath>` properties (case-sensitive), containing the path to the custom user dictionary. See "Updating the Japanese Language Analyzer Configuration File" on page 29.
- For Korean, you can add one `<UserDictionaryPath>` property, containing the path to the custom user dictionary. For Korean you can add only one custom dictionary. See "Notes on the Name and Location of the User Dictionary" on page 32.

## Configure IDOL Server for the Custom Dictionary

When you are using a custom dictionary, you must consider the following factors for your IDOL Server configuration:

- The Content component normalizes text before it sends the text to the sentence breaking library. To prevent normalization interfering with sentence breaking, you must either ensure that your custom dictionaries contain normalized text in the entries, or turn off normalization in IDOL Server. To turn off normalization, you must set the `Normalise` parameter to `false` in your language configuration, and remove any normalization-related options from the `SentenceBreakingOptions` parameter.
- If you want to decompose compound words, ensure that your `SentenceBreakingOptions` parameter includes the `DecomposeCompound` option. For more details, see "IDOL Server Language Configuration" on page 4.

## Configure the Basis Libraries to Override Values

By default, the Basis libraries use the custom user dictionaries in conjunction with the standard dictionaries, and in the event of a conflict, it uses the standard dictionary value. You can configure the library to override the standard dictionary value with your custom values, by creating a context XML configuration file.

# Appendix: Basis RLP

The following sections provide more detail about the Basis RLP Language Analyzers, and the user dictionaries.

Copyright © 2015 Basis Technology Corporation. All rights reserved. This document is property of and is proprietary to Basis Technology Corporation. It is not to be disclosed or reproduced in whole or in part without the express written consent of Basis Technology Corporation.



# Chinese Language Analyzer

**Name**

Chinese Language Analyzer

**Dependencies**

Sentence Breaker, Script Region Locator

**Language Dependent**

Chinese (Simplified and Traditional)

**XML-Configurable Options**

The options for the Chinese Language Analyzer are described by the ***BT_ROOT*/rlp/etc/claoptions.xml** file. For example:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE claconfig SYSTEM "claconfig.dtd">

<claconfig>
   <dictionarypath>
      <env name="root"/>/cma/dicts/zh_lex_<env
name="endian"/>.bin</dictionarypath>
   <readingdictionarypath>
      <env name="root"/>/cma/dicts/zh_reading_<env
name="endian"/>.bin</readingdictionarypath>
   <stopwordspath><env name="root"/>/cma/dicts/zh_stop.utf8</stopwordspath>
</claconfig>
```

The configuration file conforms to **claconfig.dtd**:

```
<!ENTITY % pathname "(#PCDATA | env)+"
<!ELEMENT claconfig (dictionarypath, posdictionarypath,
          readingdictionarypath, stopwordspath)>
<!ELEMENT dictionarypath (#PCDATA | env)*>
<!ELEMENT readingdictionarypath (#PCDATA | env)*>
<!ELEMENT stopwordspath (#PCDATA | env)*>
<!ELEMENT lockdictionary EMPTY >
<!ATTLIST lockdictionary value (yes | no) 'no'>
<!ELEMENT env EMPTY >
<!ATTLIST env name CDATA #REQUIRED>
```

The `dictionarypath` specifies the path name to the main dictionary used for segmentation. Users must use the main dictionary that comes with the analyzer. In addition, users can create and employ user dictionaries (see "Chinese User Dictionaries" on page 21). This option must be specified at least once. Users can specify one main dictionary and zero or more user dictionaries.

The `readingdictionarypath` specifies the path to the analyzer's reading dictionary, which is used to look up readings for segmented tokens.

The `stopwordspath` specifies the pathname to the stopwords list used by the analyzer.

The `lockdictionary` value indicates whether or not the pages containing the dictionary are locked in RAM.

**Context Properties**

The following table lists the context properties supported by the Chinese Language Analyzer. Note that for brevity the `com.basistech.cla` prefix has been removed from the property names in the first column. Hence the full name for `break_at_alphanum_intraword_punct` is `com.basistech.cla.break_at_alphanum_intraword_punct`.

| Property | Type | Default | Description |
|---|---|---|---|
| `break_at_alphanum_intraword_punct` | boolean | false | Ignored when `consistent_latin_segmentation` is true (the default). If true, Chinese Language Analyzer considers punctuation between alphanumeric characters as a break. For example, the text "www.basistech.com" is segmented as a single token when the option is false, but as five tokens when it is true: "www", ".", "basistech", ".", and "com". |
| `consistent_latin_segmentation` | boolean | true | If true, Chinese Language Analyzer provides consistent segmentation of embedded text in Latin script; the `break_at_alphanum_intraword_punct` and `whitespace_is_number_sep` settings are ignored. Set to false to revert to pre-RLP 7.1 segmentation behavior. |
| `decomposecompound` | boolean | true | If true, Chinese Language Analyzer decomposes compound words into their components. A word is subject to decomposition if it is a user-dictionary entry with a decomposition pattern (see "Creating the User Dictionary" on page 21) or a noun in the Chinese Language Analyzer dictionary that contains more than 4 characters. Words are never decomposed into a sequence of single-character units. |
| `favor_user_dictionary` | boolean | false | If true, resolve conflicts between a user dictionary (see "Chinese User Dictionaries" on page 21) and a system dictionary in favor of the user dictionary. |
| `generate_all` | boolean | true | If true and `readings` is true, all the readings for a token are returned. For single characters, these are returned in brackets, separated by semicolons. |

| Property | Type | Default | Description |
|---|---|---|---|
| `ignore_stopwords` | boolean | false | If false, stopwords are returned and the vector of STOPWORD results is instantiated. [1] If true, tokens that are stopwords are not returned to the caller. |
| `limit_parse_length` | non-negative integer | 0 (no limit) | Sets the maximum number of characters, *n* that are processed in a single parse buffer as a sentence. Chinese Language Analyzer normally starts parsing after it detects a sentence boundary. When a limit is set, Chinese Language Analyzer starts parsing within *n* characters, even if a sentence boundary has not yet been detected. Setting a limit avoids delays when the processor encounters thousands of characters but no sentence boundary.<br><br>**Note:** Basis Technology recommends setting the limit n to at least 100. Depending on the type of text being processed, any number less than 100 may degrade tokenization accuracy or cause Chinese Language Analyzer to split a valid token across buffers and not detect the token correctly. |
| `min_length_for_script_change` | positive integer | 10 | Ignored when `consistent_latin_segmentation` is false. The minimum length of non-native text to be considered for a script change. For example, DVD 播放机 does not trigger a script change if the setting is greater than 3.[2] |

[1] If stopwords are returned, you can determine with the C++ API whether a given token is a stopword by calling `BT_RLP_TokenIterator::IsStopword`. In Java, you can use the `List contains` method to see whether the list of stopword references returned by `RLPResultAccess getListResult(RLPConstants.STOPWORD)` contains the `Integer` index of the token.

[2] A script change indicates a boundary between tokens, so the length for script change setting you choose may influence how a mixed-script string is tokenized.

| Property | Type | Default | Description |
|---|---|---|---|
| `normalize_result_token` | boolean | false | If true, Chinese Language Analyzer generates LEMMA results with normalized number tokens: full-width Latin digits and punctuation are converted to their half-width counterparts, grouping separators are removed (e.g., 2,000 becomes 2000), and Hanzi numerals and mixed Hanzi/Latin numeric expressions are converted to Latin. |
| `pos` | boolean | true | If true, PART_OF_SPEECH results are calculated. |
| `reading_by_character` | boolean | false | If true and `readings` is true, the reading for a polysyllabic token is determined on a percharacter basis, instead of by the token as a whole. Generally speaking, this usage is problematic for polyphonic Hanzi, such as 都, which can be read as dou1 or du1 depending on the context. For example, when followed by 市, it is pronounced du1 (as in du1shi4), but is pronounced dou1 when used alone. |
| `reading_type` | string | "tone_marks" | Sets the representation of tones. Possible values:<br>• "tone_marks" [1]-- diacritics over the appropriate vowels<br>• "tone_numbers" -- a number from 1-4, suffixed to each syllable<br>• "no_tones" -- pinyin without tone presentation<br>• "cjktex" -- pinyin generated as macros for the CJKTeX pinyin.sty style |
| `readings` | boolean | false | If true, READING results are calculated. These results contain the pinyin transcription of the word in most cases, and alternative pinyin transcriptions when the recognized word has more than one way to be pronounced. |

[1]The readings are generated in Unicode, and not all Unicode fonts include glyphs for the codepoints used to represent "tone_marks".

| Property | Type | Default | Description |
|---|---|---|---|
| `separate_syllables` | boolean | false | If true, the syllables in the reading for a polysyllabic token are separated by a vertical line ("\|"). |
| `use_v_for_u_diaresis` | boolean | false | If true, v is used instead of ü. The value is implicitly true when `reading_type` is "cjktex," and is ignored when `reading_type` is "tone_marks". The substitution of v is common in environments that lack diacritics. It is probably most useful when `reading_type` is "tone_numbers". |
| `whitespace_is_number_sep` | boolean | true | Ignored when `consistent_latin_segmentation` is true (the default). Whether the Chinese language processor treats whitespace (horizontal and vertical) as a number separator. If true, the text "1995 1996" is segmented as two tokens; if false, the same text is segmented as a single token.<br><br>**Note:** The default behavior (whitespace is a numeric separator) yields different behavior than Chinese Language Analyzer versions prior to Release 4.3. |

**Description**

The Chinese Language Analyzer segments Chinese text into separate tokens (words and punctuation) and assigns part-of-speech (POS) tags to each token. Chinese Language Analyzer also reports offsets for each token, and alternative readings, if any, for Hanzi or Hanzi compounds.

The Chinese Language Analyzer returns the following result types:

- TOKEN
- TOKEN_OFFSET
- SENTENCE_BOUNDARY
- PART_OF_SPEECH — if `com.basistech.cla.pos` is true (the default)
- COMPOUND — if `com.basistech.cla.decomposecompound` is true (the default)
- READING (pinyin transcriptions) — if `com.basistech.cla.readings` is true (the default is false)

- LEMMA — if `com.basistech.cla.normalize_result_token` is true (the default is false)
- STOPWORD — if `com.basistech.cla.ignore_stopwords` is false (the default)

**Chinese User Dictionaries**

You can create user dictionaries for words specific to an industry or application. User dictionaries allow you to add new words, personal names, and transliterated foreign words. In addition, you can specify how existing words are segmented. For example, you may want to prevent a product name from being segmented even if it is a compound. For more information, see "Chinese User Dictionaries" on page 21.

# Japanese Language Analyzer

**Name**

Japanese Language Analyzer

**Dependencies**

Sentence Breaker, Script Region Locator

**Language Dependent**

Japanese

**XML-Configurable Options**

Settings for the Japanese Language Analyzer are specified in **`BT_ROOT`/rlp/etc/jla-options.xml**. This file includes pathnames for the main dictionary used for tokenization and POS tagging, the reading dictionary (with *yomigana* pronunciation aids expressed in *Hiragana*), a stopwords list, and may include one or more user dictionaries.

The user can edit the stopwords list and create user dictionaries (see"Japanese User Dictionaries" on page 24).

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE jlaconfig SYSTEM "jlaconfig.dtd">
<jlaconfig>
<DictionaryPaths>
   <DictionaryPath><env name="root"/>/jma/dicts/JP_<env
name="endian"/>.bin</DictionaryPath>
   <!-- Add a DictionaryPath for each user dictionary -->
</DictionaryPaths>

<!-- We only support one Japanese Language Analyzer reading dictionary -->
<ReadingDictionaryPath><env name="root"/>/jma/dicts/JP_<env name="endian"/>_
Reading.bin
  </ReadingDictionaryPath>

<StopwordPath><env name="root"/>/jma/dicts/JP_stop.utf8</StopwordPath>
</jlaconfig>
```

The *<env name="endian"/>* in the dictionary name is replaced at runtime with either "BE" or "LE" to match the platform byte order: big-endian or little-endian. For example, Sun's SPARC and Hewlett Packard's PA-RISC are big-endian, whereas Intel's x86 CPUs are little-endian.

The `StopwordPath` specifies the pathname to the stopwords list used by the analyzer.

**Context Properties**

The following table lists the context properties supported by the Japanese Language Analyzer processor. Note that for brevity the `com.basistech.jla` prefix has been removed from the property names in the first column. Hence the full name for `decomposecompound` is `com.basistech.jla.decomposecompound`.

| Property | Type | Default | Description |
|---|---|---|---|
| `consistent_latin_segmentation` | boolean | true | If true, Japanese Language Analyzer provides consistent segmentation of embedded text in Latin script; the `segment_non_japanese` setting is ignored. Set to false to revert to pre-RLP 7.1 segmentation behavior. |
| `decomposecompound` | boolean | true | If true, Japanese Language Analyzer decomposes compound words into their components.[1] |
| `deep_compound_decomposition` | boolean | false | If true, Japanese Language Analyzer recursively decomposes into smaller components the components marked in the dictionary as being decomposable.[2] |
| `favor_user_dictionary` | boolean | false | If true, Japanese Language Analyzer favors words in the user dictionary (over the standard Japanese dictionary) during tokenization. |
| `generate_token_sources` | boolean | false | If true, Japanese Language Analyzer generates a `TOKEN_SOURCE_ID` result for each token. You can use the `TOKEN_SOURCE_ID` to get the `TOKEN_SOURCE_NAME` of the dictionary. |
| `ignore_separators` | boolean | true | If true, Japanese Language Analyzer ignores whitespace separators when tokenizing input text. If false, Japanese Language Analyzer treats whitespace separators as token delimiters. Note that Japanese orthography allows a newline to occur in the middle of a word. |

[1]To access the components that make up the compound, use the COMPOUND result.
[2]To access the components that make up the compound, use the COMPOUND result.

| Property | Type | Default | Description |
|---|---|---|---|
| `ignore_stopwords` | boolean | false | If true, tokens that are stopwords are not returned to the caller. If false, tokens that are stopwords are returned and the vector of STOPWORD results is instantiated.[1] |
| `limit_parse_length` | 0 or positive integer | 0 (no limit) | Sets the maximum number of characters, *n* that are processed in a single parse buffer as a sentence. Japanese Language Analyzer normally starts parsing after it detects a sentence boundary. When a limit is set, Japanese Language Analyzer starts parsing within *n* characters, even if a sentence boundary has not yet been detected. Setting a limit avoids delays when the processor encounters thousands of characters but no sentence boundary.<br><br>**Note:** Basis Technology recommends setting the limit *n* to at least 100. Depending on the type of text being processed, any number less than 100 may degrade tokenization accuracy or cause Japanese Language Analyzer to split a valid token across buffers and not detect the token correctly. |
| `min_length_for_script_change` | positive integer | 10 | Ignored when `consistent_latin_segmentation` is false. The minimum length of non-native text to be considered for a script change. For example, DVD レコーダー does not trigger a script change if the setting is greater than 3. |
| `normalize_result_token` | boolean | false | If true, when a dictionary form is not available and a normalized form is, Japanese Language Analyzer returns the normalized form as LEMMA. Middle dots are removed from words (e.g., ワールド・ミュージック is normalized to ワールドミュージック) and numbers are normalized: zenkaku (fullwidth) Arabic numerals such as １０，０００ are converted to half-width numerals; commas are removed (e.g., 2,000 becomes 2000); Kanji numerals are converted to halfwidth numerals (e.g., 四千 becomes 4000). Note: independent of this setting, the dictionary form, if available, is always returned as LEMMA. |

[1] If stopwords are returned, you can determine with the C++ API whether a given token is a stopword by calling `BT_RLP_TokenIterator::IsStopword`. In Java, you can use the `List contains` method to see whether the list of stopword references returned by `RLPResultAccess getListResult(RLPConstants.STOPWORD)` contains the `Integer` index of the token.

| Property | Type | Default | Description |
|---|---|---|---|
| `readings` | boolean | false | If true, READING [113] results are calculated. These results contain Furigana transcriptions rendered in Hiragana. |
| `segment_non_japanese` | boolean | true | Ignored when `consistent_latin_segmentation` is true (the default). When true, non-Japanese text is segmented at Latin script and number boundaries. For example, 206xs is tokenized as 206 and xs. If false, 206xs is tokenized as 206xs. |
| `separate_numbers_from_counters` | boolean | true | If true, Japanese Language Analyzer returns numbers and their counters as separate tokens. Warning: If you set it to false, you degrade the accuracy of the Base Noun Phrase Locator and Statistical Entity Extractor. |
| `separate_place_name_from_suffix` | boolean | true | If true, Japanese Language Analyzer separates place names from their suffixes (e.g., 岡山県 is tokenized to 岡山 and 県). Warning: If you set it to false, you degrade the accuracy of the Base Noun Phrase Locator and Statistical Entity Extractor. |
| `whitespace_tokenization` | boolean | false | If true, Japanese Language Analyzer tokenizes on whitespace boundaries only. This can be useful in search applications when parsing short query strings. Such queries typically do not have enough context for accurate segmentation based on morphological analysis. |

**Description**

The Japanese Language Analyzer tokenizes Japanese text into separate words and assigns a Part-of-Speech (POS) tag to each word; see Japanese POS Tags [286] . The Japanese Language Analyzer returns the following result types:

- TOKEN
- TOKEN_OFFSET
- SENTENCE_BOUNDARY
- PART_OF_SPEECH
- LEMMA — dictionary form when available, or normalized form if available and `com.basistech.jla.normalize_result_token` is true (the default is false)
- COMPOUND — if `com.basistech.jla.decomposecompound` is true (the default)

- READING (Furigana transcriptions rendered in Hiragana) if `com.basistech.jla.readings` is true (the default is false)
- STOPWORD — if `com.basistech.jla.ignore_stopwords` is false (the default)
- TOKEN_SOURCE_ID — if `com.basistech.jla.generate_token_sources` is true (the default is false)
- TOKEN_SOURCE_NAME — if `com.basistech.jla.generate_token_sources` is true (the default is false)

**Japanese User Dictionaries**

Japanese Language Analyzer includes the capability to create and use one or more segmentation (tokenization) user dictionaries for words specific to an industry or application. User dictionaries allow you to add new words, personal names, and transliterated foreign words. In addition, you can specify how compound words are tokenized. For example, you may want to prevent a product name from being segmented even if it is a compound.

You can also create reading user dictionaries with Furigana transcriptions rendered in Hiragana.You can use a reading user dictionary to override readings returned by the standard JLA reading dictionaryand to override readings guessed from a standard segmentation (tokenization) user dictionary.

For more information, see "Japanese User Dictionaries" on page 24.

# Korean Language Analyzer

**Name**

Korean Language Analyzer

**Dependencies**

Sentence Breaker, Script Region Locator

**Language Dependent**

Korean

**XML-Configurable Options**

The options for the Korean Language Analyzer are defined in ***BT_ROOT*/rlp/etc/kla-options.xml**.

For example:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE klaconfig SYSTEM "klaconfig.dtd">
<klaconfig>
<dictionarypath><env name="root"/>/kma/dicts/</dictionarypath>
<utilitiesdatapath><env name="root"/>/utilities/data/</utilitiesdatapath>
<stopwordspath><env name="root"/>/kma/dicts/kr_stop.utf8</stopwordspath>
</klaconfig>
```

The configuration file must conform to **klaconfig.dtd**:

```
<!ELEMENT klaconfig (dictionarypath, utilitiesdatapath)>
<!ELEMENT dictionarypath (#PCDATA)>
<!ELEMENT utilitiesdatapath (#PCDATA)>
<!ELEMENT stopwordspath (#PDCATA)>
```

Note that for the Korean Language Analyzer, the `dictionarypath` points to the directory that contains the required dictionaries. This is different from the Japanese and Chinese Language Analyzer behavior, which requires a path to each dictionary that you are including.

The `utilitiesdatapath` must specify **utilities/data**. This contains internal transcription tables.

The `stopwordspath` specifies the pathname to the stopwords list used by the analyzer.

**Context Properties**

The following table lists the context property supported by the Korean Language Analyzer processor. Note that for brevity the `com.basistech.kla` prefix has been removed from the property names in the first column. Hence the full name for `ignore_stopwords` is `com.basistech.kla.ignore_stopwords`.

| Property | Type | Default | Description |
|---|---|---|---|
| `consistent_ latin_ segmentation` | boolean | true | If true, Korean Language Analyzer provides consistent segmentation of embedded text in Latin script. Set to false to revert to pre-RLP 7.1 segmentation behavior. |
| `ignore_ stopwords` | boolean | false | If false, stopwords are returned and the vector of STOPWORD results is instantiated. [1]If true, tokens that are stopwords are not returned to the caller. |
| `min_length_ for_script_ change` | positive integer | 10 | Ignored when `consistent_latin_segmentation` is false. The minimum length of non-native text to be considered for a script change. For example, DVD 플레이어 does not trigger a script change if the setting is greater than 3. |

**Description**

The Korean Language Analyzer segments Korean text into separate words and compounds, reports the length of each word and the lemma, and assigns a Part-of-Speech (POS) tag to each word. Korean Language Analyzer also returns a list of compound analyses (may be empty).

The Korean Language Analyzer returns the following result types:

- TOKEN
- TOKEN_OFFSET

[1]If stopwords are returned, you can determine with the C++ API whether a given token is a stopword by calling `BT_RLP_TokenIterator::IsStopword`. In Java, you can use the `List contains` method to see whether the list of stopword references returned by `RLPResultAccess getListResult(RLPConstants.STOPWORD)` contains the `Integer` index of the token.

- SENTENCE_BOUNDARY
- PART_OF_SPEECH
- COMPOUND
- LEMMA
- STOPWORD — if `com.basistech.kla.ignore_stopwords` is false (the default)

**Korean User Dictionary**

Korean Language Analyzer provides a user dictionary that users can edit and recompile. For more information, see "Korean User Dictionary" on page 30.

# Chinese User Dictionaries

You can create user dictionaries for words specific to an industry or application. User dictionaries allow you to add new words, personal names, and transliterated foreign words. In addition, you can specify how existing words are segmented. For example, you may want to prevent a product name from being segmented even if it is a compound.

For efficiency, Chinese user dictionaries are compiled into a binary form with big-endian or little-endian byte order to match the platform.

**Procedure for Using a Chinese User Dictionary**

1. Create the dictionary. See "Creating the User Dictionary" below.
2. Compile the user dictionary. See "Compiling the User Dictionary" on page 23.
3. Put the dictionary in the ***BT_ROOT*/rlp/cma/dicts** directory. See "Where to Put the User Dictionary" on page 24
4. Edit the Chinese Language Analyzer configuration file to include the user dictionary. See "Updating the Chinese Language Analyzer Configuration File" on page 24

## *Creating the User Dictionary*

The source file for a Chinese user dictionary is UTF-8 encoded (see "Valid Characters for Chinese User Dictionary Entries" on page 23). The file may begin with a byte order mark (BOM). Empty lines are ignored. A comment line begins with #.

Each entry is a single line:

*word* **Tab** *POS* **Tab** *DecompPattern*

where *word* is the noun, *POS* is one of the user-dictionary part-of-speech tags listed below, and *DecompPattern* (optional) is the decomposition pattern: a comma-delimited list of numbers that specify the number of characters from *word* to include in each component of the compound (0 for no

decomposition). The individual components that make up the compound are in the COMPOUND results.

**User Dictionary POS Tags (case-insensitive)**

- NOUN
- PROPER_NOUN
- PLACE
- PERSON
- ORGANIZATION
- FOREIGN_PERSON

For example, the user dictionary entry

```
深圳发展銀行 organization 2,2,2
```

indicates that 深圳发展銀行 should be decomposed into three two-character components:

```
深圳
发展
銀行
```

The sum of the digits in the pattern must match the number of characters in the entry. For example,

```
深圳发展銀行 noun 4,9
```

is invalid because the entry has 6 characters while the pattern is for a 13-character string. The correct entry is:

```
深圳发展銀行 noun 2,4
```

The POS and decomposition pattern can be in Chinese full-width numerals and Roman letters. For example:

```
上海证券交易所 organization １,２,３,１
```

Decomposition can be prevented by specifying a pattern with the special value "0" or by specifying a pattern consisting of a single digit with the length of the entry.

For example:

```
北京人 noun 0
```

or

```
北京人 noun 3
```

Tokens matching this entry will not be decomposed. To prevent a word that is also listed in a system dictionary from being decomposed, set `com.basistech.cla.favor_user_dictionary` to true.

## Valid Characters for Chinese User Dictionary Entries

An entry in a Chinese user dictionary must contain characters corresponding to the following Unicode code points or to valid surrogate pairs. In this listing, .. indicates an inclusive range of valid code points:

0023..002F, 0030..0039, 0040..005A, 005F, 0060..007A, 007E, 00A2..00A5, 00B7, 00C0..00D6,00D8..00F6, 00F8..00FF, 0100..017F, 0180..024F, 09F2..09F3, 0E3F, 17DB, 2010..206F, 2160..216B,2170..217B, 2200..22FF, 2460..24FF, 25A0..25FF, 25CB, 2600..26FF, 2E80..2EF3, 2F00..2FD5, 2FF0..2FFB, 3003..3007, 3012, 3020, 3031..3037, 30FB, 3200..32FF, 3401..4DB5, 4E00..9FA5,F900..FA2D, FDFC, FE69, FF00, FF02..FFEF

For example, the full stop 。(3002), indicates a sentence break and may not be included in a dictionary entry,

## Compiling the User Dictionary

Chinese Language Analyzer requires the dictionary as described above to be in a binary form. The byte order of the binary dictionary must match the byte order of the runtime platform. The platform on which you compile the dictionary determines the byte order. To use the dictionary on both a little-endian platform (such as an Intel x86 CPU) and a big-endian platform (such as a Sun Solaris), generate a binary dictionary on each of these platforms.

**Note:** A compiled user dictionary may also contain entries that include PUA or Supplementary characters. See "Entering Non-Standard Characters in a Chinese or Japanese User Dictionary (or a Gazetteer)" on page 33 .

The script for generating a binary dictionary is ***BT_ROOT*/rlp/cma/source/samples/build_user_dict.sh**.

**Prerequisites**

- Unix or Cygwin (for Windows).
- The `BT_ROOT` environment variable must be set to *`BT_ROOT`* , the Basis root directory. For example, if **RLP SDK** is installed in **/usr/local/basistech**, set the `BT_ROOT` environment variable to **/usr/local/basistech**.
- The `BT_BUILD` environment variable must be set to the platform identifier embedded in your SDK package file name (see "Environment Variables" on page 2).

To compile the dictionary into a binary format, issue the following command:

```
build_user_dict.sh input output
```

For example, if you have a user dictionary named **user_dict.utf8**, build the binary user dictionary **user_dict.bin** with the following command:

```
./build_user_dict.sh user_dict.utf8 user_dict.bin
```

**Note:** If you are making the user dictionary available for little-endian and big-endian platforms, you can compile the dictionary on both platforms, and differentiate the dictionaries by using user_dict_LE.bin for the little-endian dictionary and user_dict_BE.bin for the big-endian dictionary.

## *Where to Put the User Dictionary*

We recommend that you put your Chinese user dictionaries in ***BT_ROOT*/rlp/cma/dicts**, where `BT_ROOT` is the root directory of the **RLP SDK**.

## *Updating the Chinese Language Analyzer Configuration File*

To instruct Chinese Language Analyzer to use your user dictionary, add a `<dictionarypath>` element to **cla-options.xml**. For example, suppose the binary user dictionary is named **user_dict.bin** and is in ***BT_ROOT*/rlp/cma/dicts**. Modify ***BT_ROOT*/rlp/etc/cla-options.xml** to include the new `<dictionarypath>` element.

```
<claconfig>
  ...
  ...
  <dictionarypath><env name="root"/>/cma/dicts/user_dict.bin</dictionarypath>
</claconfig>
```

If you are making the user dictionary available for little-endian and big-endian platforms, and you are differentiating the two files as indicated above ("LE" and "BE"), you can set up the Chinese Language Analyzer configuration file to choose the correct binary for the runtime platform:

```
<claconfig>
...
...
<dictionarypath><env name="root"/>/cma/dicts/user_dict_<env
name="endian"/>.bin</dictionarypath>
</claconfig>
```

The *<env name="endian"/>* in the dictionary name is replaced at runtime with "BE" if the platform byte order is big-endian or "LE" if the platform byte order is little-endian.

**Note:** At runtime, RLP replaces <env name="root"/> with the path to the RLP root directory ( ***BT_ROOT*/rlp**).

You can specify multiple user dictionaries in the options file.

# Japanese User Dictionaries

Japanese Language Analyzer includes the capability to create and use one or more segmentation (tokenization) user dictionaries for nouns specific to an industry or application. User dictionaries allow you to add new nouns, including also personal and organizational names, and transliterated

foreign nouns. In addition, you can specify how compound nouns are tokenized. For example, you may want to prevent a product name from being segmented even if it is a compound.

You can also create user reading dictionaries with Furigana transcriptions rendered in Hiragana. The readings can override the readings returned from the JLA reading dictionary and override readings that are otherwise guessed from segmentation (tokenization) user dictionaries.

Japanese segmentation (tokenization) user dictionaries and reading user dictionaries are compiled into separate binary forms with big-endian or little-endian byte order to match the platform. Both dictionary types can be compiled from the same source file.

**Procedure for Creating and Using a Japanese User Dictionary**

1. Create the user dictionary. See "Creating the Source File" below.
2. Compile the source file. See "Compiling the User Dictionary" on page 27.
3. Put the user dictionary in the Japanese Language Analyzer dictionary directory. See "Where to Put the User Dictionary" on page 29.
4. Edit the Japanese Language Analyzer configuration file to include the user dictionary. See "Updating the Japanese Language Analyzer Configuration File" on page 29.

## *Creating the Source File*

The source file for a Japanese user dictionary is UTF-8 encoded (see "Valid Characters for Japanese User Dictionary Entries" on page 27). The file may begin with a byte order mark (BOM). Empty lines are ignored. A comment line begins with #.

If you want to identify the dictionary where Japanese Language Analyzer found each token, you must assign each user dictionary a name, and you must compile the dictionary (see "Compiling the User Dictionary" on page 27). At the top of the file, enter

!DICT_LABEL **Tab** *Dictionary Name*

where *Dictionary Name* is the name you want to assign to the dictionary.

Each entry in the dictionary is a single line:

*word* **Tab** *POS* **Tab** *DecompPattern* **Tab** *Reading1*,*Reading2*,...

where *word* is the noun, *POS* is one of the user-dictionary part-of-speech tags listed below, and *DecompPattern* is the decomposition pattern: a comma-delimited list of numbers that specify the number of characters from word to include in each component of the compound (0 for no decomposition), and *Reading1*,...is a comma-delimited list of one or more Furigana transcriptions rendered in Hiragana or Katakana.

The individual components that make up the compound are in the `COMPOUND` results. The decomposition pattern and readings are optional, but you must include a decomposition pattern if you include readings. In other words, you must include all elements to include the entry in a reading user dictionary, even though the reading user dictionary does not use the POS tag or decomposition pattern. To include an entry in a segmentation (tokenization) user user dictionary, you only need POS tag and an optional decomposition pattern. Keep in mind that those entries that include all

elements can be included in both a segmentation (tokenization) user dictionary and a reading user dictionary.

**User Dictionary POS Tags**

- NOUN
- PROPER_NOUN
- PLACE
- PERSON
- ORGANIZATION
- GIVEN_NAME
- SURNAME
- FOREIGN_PLACE_NAME
- FOREIGN_GIVEN_NAME
- FOREIGN_SURNAME
- AJ (adjective)
- AN (adjectival noun)
- D (adverb)
- HS (honorific suffix)
- V1 (vowel-stem verb)
- VN (verbal noun)
- VS (suru-verb)
- VX (irregular verb)

Examples (the last three entries include readings)

```
!DICT_LABEL New Words 2014
デジタルカメラ NOUN
デジカメ NOUN 0
東京証券取引所 ORGANIZATION 2,2,3
狩野 SURNAME 0
安倍晋三 PERSON 2,2 あべしんぞう
麻垣康三 PERSON 2,2 あさがきこうぞう
商人 NOUN 0 しょうにん, あきんど
```

The POS and decomposition pattern can be in Japanese full-width numerals and Roman letters. For example:

東京証券取引所　organization　２,２,３

The "2,2,3" decomposition pattern instructs Japanese Language Analyzer to decompose this compound entry into

東京
証券
取引所

# Valid Characters for Japanese User Dictionary Entries

An entry in a Japanese user dictionary must contain characters corresponding to the following Unicode code points, to valid surrogate pairs, or to letters or decimal digits in Latin script. In this listing, .. indicates an inclusive range of valid code points:

0025..0039, 0040..005A, 005F..007A, 007E, 00B7, 0370..03FF, 0400..04FF, 2010..206F, 2160..217B, 2200..22FF, 2460..24FF, 25A0..25FF, 2600..26FF, 3003..3007, 3012, 3020, 3031..3037, 3041..3094, 3099..309E, 30A1..30FA, 30FC..30FE, 3200..32FF, 3300..33FF, 4E00..9FFF, D800..DBFF, DC00..DFFF, E000..F8FF, F900..FA2D, FF00, FF02..FFEF

For example, the full stop 。(3002), indicates a sentence break and must not be included in a dictionary entry. The Katakana middle dot ・(30FB) must not appear in a dictionary entry; input strings with this character match the corresponding dictionary entries without the character.

# Compiling the User Dictionary

Japanese Language Analyzer requires the dictionary as described above to be in a binary form. If you have created a reading dictionary, you must compile it separately, even if it is in the same source file as a segmentation dictionary.

The byte order of the binary dictionary must match the byte order of the runtime platform. The platform on which you compile the dictionary determines the byte order. To use the dictionary on both a little-endian platform (such as an Intel x86 CPU) and a big-endian platform (such as a Sun Solaris), generate a binary dictionary on each of these platforms.

**Segmentation (Tokenization) User Dictionary.** The script for generating a binary segmentation (tokenizzation) user dictionary is a shell script for Unix (**build_user_dict.sh**) and a batch file for Windows (**build_user_dict.bat**). The script is in ***BT_ROOT*/rlp/jma/source/samples/**.

The script for generating a binary dictionary is a shell script for Unix (**build_user_dict.sh**) and a batch file for Windows (**build_user_dict.bat**). The script is in ***BT_ROOT*/rlp/jma/source/samples/**.

**Prerequisites**

- The BT_ROOT environment variable must be set to ***BT_ROOT***, the Basis Technology root directory. For example, if **RLP SDK** is installed in **/usr/local/basistech**, set the BT_ROOT environment variable to **/usr/local/basistech**.

- The BT_BUILD environment variable must be set to the platform identifier embedded in your SDK package file name (see "Environment Variables" on page 2).

To compile the dictionary into a binary format that Japanese Language Analyzer can use, issue the following command:

```
build_user_dict.[sh|bat] input output
```

For example, if you have a user dictionary named **user_dict.utf8**, build the binary user dictionary **user_dict.bin** with the following command in a Unix shell:

```
./build_user_dict.sh user_dict.utf8 user_dict.bin
```

**Reading User Dictionary.** Compile the dictionary on the little-endian or big-endian platform on which you plan to use the dictionary. The script for generating a binary reading user dictionary is **build_user_reading_dict.sh** in ***BT_ROOT*/rlp/jma/source/samples/**.

**Prerequisites**

- Unix or Cygwin (for Windows)
- Python
- The BT_ROOT environment variable must be set to ***BT_ROOT***, the Basis Technology root directory. For example, if RLP SDK is installed in /usr/local/basistech, set the BT_ROOT environment variable to **/usr/local/basistech**.
- The BT_BUILD environment variable must be set to the platform identifier embedded in your SDK package file name (see "Environment Variables" on page 2 ).

To compile the reading dictionary into a binary format that Japanese Language Analyzer can use, issue the following command in a Unix shell or Cygwin (Windows):

```
build_user_reading_dict.sh input output
```

For example, if you have a user dictionary named **user_dict.utf8**, build the binary readinguser dictionary **user_reading_dict.bin** with the following command in a Unix shell:

```
./build_user_reading_dict.sh user_dict.utf8 user_reading_dict.bin
```

**Note:** For both segmentation (tokenization) user dictionaries and reading user dictionaries, if you are making the user dictionary available for little-endian and big-endian platforms, you can compile the dictionary on both platforms, and differentiate the dictionaries by using user_dict_LE.bin for the little-endian dictionary and user_dict_BE.bin for the big-endian dictionary.

The extension for the Japanese dictionary files (system and user) does not have to be **.bin**.

## *Non-Compiled User Dictionaries*

For backwards compatibility, Japanese Language Analyzer continues to support non-compiled user dictionaries. Keep in mind that non-compiled dictionaries are less efficient and contain less

information. A non-compiled user dictionary must be in UTF-8 and may contain comments, single-field (word) entries, and double-field entries with a word and a decomposition pattern:

- Comment lines beginning with a pound sign (#).
- Word entries (one word per line with no POS, but may include a **Tab** and a decomposition pattern). The decomposition pattern is a series of one or more digits without commas. For example:

  ```
  東京証券取引所 223
  ```

## *Where to Put the User Dictionary*

We recommend that you put your segmentation (tokenization) Japanese user dictionaries and reading user dictionaries in ***BT_ROOT*/rlp/jma/dicts**, where *BT_ROOT* is the root directory of the **RLP SDK**.

## *Updating the Japanese Language Analyzer Configuration File*

To use **user_dict.bin** and **user_reading_dict.bin** with the Japanese Language Analyzer, modify the **jla-options.xml** file to include them. For example, if you put your user dictionaries in the location we recommend (the directory that contains the system Japanese dictionary), modify it to read as follows:

```
<DictionaryPaths>
  <DictionaryPath><env name="root"/>/jma/dicts/JP_<env
name="endian"/>.bin</DictionaryPath>
  <!-- Add a DictionaryPath for each standard user dictionary -->
  <DictionaryPath>
    <env name="root"/>/jma/dicts/user_dict.bin
  </DictionaryPath>
</DictionaryPaths>
<!-- To supply your own reading dictionary, add the user reading dictionary
before the JLA Reading dictionary -->
<ReadingDictionaryPaths>
  <ReadingDictionaryPath>
    <env name="root"/>/jma/dicts/user_reading_dict.bin</ReadingDictionaryPath>
  <ReadingDictionaryPath>
    <env name="root"/>/jma/dicts/JP_<env name="endian"/>_
Reading.bin</ReadingDictionaryPath>
</ReadingDictionaryPaths>
```

If you are making the user dictionary available for little-endian and big-endian platforms, and you are differentiating the two files as indicated above ("LE" and "BE"), you can set up the Japanese Language Analyzer configuration file to choose the correct binary for the runtime platform:

```
<DictionaryPaths>
  <DictionaryPath><env name="root"/>/jma/dicts/JP_<env
name="endian"/>.bin</DictionaryPath>
```

```
  <!-- Add a DictionaryPath for each user dictionary -->
  <DictionaryPath>
    <env name="root"/>/jma/dicts/user_dict_<env name="endian"/>.bin
  </DictionaryPath>
  </DictionaryPaths>
<ReadingDictionaryPaths>
  <ReadingDictionaryPath>
    <env name="root"/>/jma/dicts/user_reading_dict_<env
name="endian"/>.bin</ReadingDictionaryPath>
  <ReadingDictionaryPath>
    <env name="root"/>/jma/dicts/JP_<env name="endian"/>_
Reading.bin</ReadingDictionaryPath>
</ReadingDictionaryPaths>
```

The *<env name="endian"/>* in the dictionary name is replaced at runtime with "BE" if the platform byte order is big-endian or "LE" if the platform byte order is little-endian.

**Note:** At runtime, RLP replaces <env name="root"/> with the path to the RLP root directory ( ***BT_ROOT*/rlp**).

You can specify multiple standard user dictionaries and reading user dictionaries in the options file.

# Korean User Dictionary

Korean Language Analyzer (see "Korean Language Analyzer" on page 19) provides one dictionary that users can edit and recompile.

**Note:** Prior to Release 6.0, the contents of this dictionary were maintained in two separate dictionaries: a Hangul dictionary and a compound noun dictionary.

As specified in the Korean Language Analyzer options file (see "XML-Configurable Options" on page 19) `dictionarypath` element, this dictionary in its compiled form is in ***BT_ROOT*/rlp/kma/dicts**. If your platform is little-endian, the compiled dictionary filename is **kla-usr-LE.bin**. If your platform is big-endian, the compiled dictionary filename is **kla-usr-BE.bin**. You can modify and recompile this dictionary. Do not change its name.

**Procedure for Modifying the User Dictionary**

1. Edit the dictionary source file. See "Editing the Dictionary Source File" below.
2. Recompile the dictionary. See "Compiling the User Dictionary" on page 32.

## *Editing the Dictionary Source File*

The source file for the compiled user dictionary shipped with RLP is ***BT_ROOT*/rlp/kma/samples/kla-usrdict.u8**. The source file is UTF-8 encoded. A comment line begins with #. The file begins with a number of comment lines that document the format of the dictionary entries.

Each dictionary entry is a single line:

*word* **Tab** *POS* **Tab** *DecompPattern*

*word* is the lemma (dictionary) form of the word. Verbs and adjectives should not include the "-ta" suffix.

*POS* is one or more of the user-dictionary part-of-speech tags listed below. An entry can have multiple parts of speech; simply concatenate the part of speech codes. For example, the POS for a verb that can be used transitively and intransitively is "IT".

*DecompPattern* (optional) is the decomposition pattern for a compound noun: a comma-delimited list of numbers that specify the number of characters from *word* to include in each component of the compound (0 for no decomposition). Korean Language Analyzer uses a decomposition algorithm to decompose compound nouns that contain no *DecompPattern*. The individual components that make up the compound are in the COMPOUND results.

| POS | Meaning |
|---|---|
| N | Noun |
| P | Pronoun |
| U | Auxiliary noun |
| M | Numeral |
| c | Compound noun |
| T | Transitive verb |
| I | Intransitive verb |
| W | Auxiliary verb |
| S | Passive verb |
| C | Causative verb |
| J | Adjective |
| K | Auxiliary adjective |
| B | Adverb |
| D | Determiner |
| L | Interjection (exclamation) |

Examples:

```
개배때기 N
그러더니 B
그러던 D
꿰이 TC
```

```
개인홈페이지 c
경품대축제 c 2,3
```

One compound noun (개인홈페이지) contains no decomposition pattern, so Korean Language Analyzer uses a decomposition algorithm to decompose it. For the other compound noun (경품대축제), the "2,3" decomposition pattern instructs Korean Language Analyzer to decompose it into

```
경품
대축제
```

You can add new entries and modify or delete existing entries.

# Compiling the User Dictionary

Compile the dictionary on the little-endian or big-endian platform on which you plan to use the dictionary.

The script for generating a binary dictionary is ***BT_ROOT*/rlp/kma/source/samples//build_user_dict.sh**.

**Prerequisites**

- Unix or Cygwin (for Windows).
- The `BT_ROOT` environment variable must be set to *`BT_ROOT`*, the Basis Technology root directory. For example, if Korean Language Analyzer is installed in **/usr/local/basistech**, set `BT_ROOT` to **/usr/local/basistech**.
- The `BT_BUILD` environment variables must be set to the platform identifier embedded in your Korean Language Analyzer package name, such as `ia32-glibc22-gcc32`. For a list of the `BT_BUILD` values, see "Environment Variables" on page 2.

To compile the dictionary into a binary format, issue the following command:

```
build_user_dict.sh input output
```

where *input* is the input filename (`kla-userdict.u8`, unless you have changed the name) and *ouput* is `kla-usr-LE.bin` if your platform is little-endian or `kla-usr-BE.bin` if your platform is big-endian.

# Notes on the Name and Location of the User Dictionary

You must put the binary user dictionary in the dictionary directory specified by the `dictionarypath` element in kla-options.xml (see "XML-Configurable Options" on page 19). As shipped, this directory is ***BT_ROOT*/rlp/kma/dicts**. As indicated above, the default filename for the user dictionary is `kla-usr-LE.bin` or `kla-usr-BE.bin`. There can only be one user dictionary, so we recommend you use the default filename. If you want to use a different filename, you must add a `userdictionarypath` element to **kla-options.xml** with the filename (no path). Suppose, for example, that you have compiled the user dictionary with the name `my-kla-usr-LE.bin` and placed that file in the dictionary directory. Edit **kla-options.xml** so it contains `userdictionarypath` as indicated below:

```
<klaconfig>
  <dictionarypath<env name="root"/>/kma/dicts</dictionarypath>
  <userdictionarypath>my-kla-usr-LE.bin</userdictionarypath>
  ..
  ..
<klaconfig>
```

# Entering Non-Standard Characters in a Chinese or Japanese User Dictionary (or a Gazetteer)

In a Chinese or Japanese user dictionary, you may want to include terms that include Unicode Private Use Area (PUA) characters for user-defined characters (UDC) or non-BMP (Supplementary) characters found in personal names.

**PUA Characters**. Characters in the range U+E000 - U+F8FF. Use `\uxxxx` where the `u` is lower-case and each `x` is a hexadecimal character.

**Supplementary Characters**. Unicode characters in the range >U+FFFF. Use `\Uxxxxxxxx` where the `U` is upper-case and each `x` is a hex character.

In fact, you can use `\uxxxx` for any character in the range U+E000 - U+F8FF, and you can use `\Uxxxxxxxx` to represent any Unicode character. You can also use this syntax for entering characters in text gazetteers.