BASIS TECHNOLOGY

Rosette LINGUISTICS PLATFORM

Application Developer's Guide

Release 6.5.2

We Put the World in the World Wide Web®

Application Developer's Guide

Published April 2009

Basis Technology Corp.

One Alewife Center
Cambridge, MA 02140
F 617.386.2020
E ProductSupport@basistech.com

Preface

The Rosette Linguistics Platform (RLP) is designed for document handling systems that need to identify, classify, analyze, index, and search unstructured text in many different languages. By integrating RLP, developers can enable their applications to process raw text data by identifying the language and encoding of a given document, converting the text to Unicode, and performing comprehensive linguistic analysis and entity extraction of text in English and a variety of Asian, European and Middle Eastern languages.

1. In this Guide

This guide explains how to install, configure, and use RLP to process and analyze text in a variety of languages. Major topics include:

- Introduction to the RLP Feature Set and Architecture [1]
- Installing RLP [5]
- Using the RLP Command-Line Utility [6]
- Creating an RLP Application [17]
- Sample RLP Applications [27]
- Named Entities [45]
- Processing Multilingual Text [73]
- Preparing Data for Processing [77]
- Accessing RLP Result Data [83]
- Configuring RLP for Distribution with an Application [105]
- User-Defined Data: Stopwords and User Dictionaries [191]
- Appendixes on Parts of Speech [209] , Morphological and Special Tags [233] , Tcl Regular Expression Syntax [239] , Error Codes [249] , Guidelines for Bug Reporting [253] , and the Windows Rosette Demo [257] .
- A Glossary [269]

2. Other Documentation

This developer's guide is intended to be used with

- Release Notes [RLP-6.5.2-ReadMe.html]

  The *RLP Release Notes* contain up-to-date information about new features and bug fixes in this release.

- API Reference [api-reference/index.html]

  The *API Reference* includes HTML documentation generated from source code for the C++ and Java APIs.

3. What's New

The following features are new in RLP 6.5.

- **Named Entities.** Expanded named entity [45] support to include Russian and three new entity types: TITLE, NUMBER, and DISTANCE. For this release, we have acquired, annotated, and performed statistical training with new data; improved and added more regular expressions; and introduced internal language-specific binary gazetteers (created by Basis Technology). To improve performance, the PERCENT entity type has been removed.

Added Regular Expression [163] support for naming and reusing regular expressions (including expression fragments).

Added Named Entity Redactor [160] support for joining adjacent named entities into a single named entity. By default, adjacent TITLE and PERSON entities are joined into a PERSON entity.

Added the `com.basistech.neredact.prefer_length` property. When set to true (the default), this property instructs the Named Entity Redactor [160] to resolve a conflict between overlapping candidate entities in favor of the longer candidate.

Added the `com.basistech.neredact.max_entity_tokens` property. When a named entity returned by NamedEntityExtractor contains more than this number of tokens (the default is 8), Named Entity Redactor [160] discards the entity.

Added the FragmentBoundaryDetector [143] , which uses whitespace to separate items in fragmentary text (such as lists and tables), so that the NamedEntityExtractor [156] will not combine a series of fragments into a single entity. The Regular Expression [163] processor also contains a new context property (`com.basistech.regexp.respect_boundaries`) that you can set to instruct the processor not to cross fragment boundaries when matching text.

Added support for language-specific text gazetteers. See Gazetteer Dictionary Paths [145] .

Java API Updates in com.basistech.rlp.RLPResultAccess: Added access to `NamedEntityData`. Replaced `getIntegerData()` with `getDetectedLanguage()` and `getDetectedScript`, which return the appropriate `com.basistech.util Enum` type (`LanguageCode` and `ISO639`).

Added API to guarantee that multiple instances of an entity are consistently returned [50] with the same entity type

Added a facility for creating a blacklist [50] of strings that are not to be returned by the Named Entity Extractor for the specified entity type.

- Expanded the .NET API [69] to provide complete access to RLP functionality. The .NET API is modeled on the Java API.
- Enhanced the Base Linguistics (BL1) [129] processor to be run simultaneously in multiple threads.
- Replaced the Japanese Orthographic Normalizer (JON) with the ManyToOneNormalizer [153] , which provides a multi-language utility for using language-specific normalization dictionaries to provide normalized tokens. We continue to distribute a Japanese normalization dictionary. Users can add their own normalization dictionaries for any of the languages we support.
- Moved sample code for integrating **RLP** with **Lucene** and **Solr** into a separate package (**rlplucene-6.0.0-sdk-unix.tar.gz** or **rlplucene-6.0.0-sdk-win.zip**).
- Added support for instantiating multiple Environment objects in the same process. Each of these Environment objects is a wrapper for the same underlying Environment. Accordingly all Environment objects must be initialized with the same environment configuration (normally **rlp-global.xml**).
- Replaced Unix Make files and Windows Visual Studio Solution and Project files with scripts for building the C++ , C, and .NET sample applications. The Unix **.sh** scripts are designed to be run in a Bash Shell. The Windows**.bat** scripts should be run in the Command Prompt. We continue to provide Ant scripts for building and running the Java sample applications, as well as Unix **.sh** scripts and Windows **.bat** scripts for running all the sample applications.
- RLI can now detect UTF-16LE and UTF-16BE, even if the endianness does not match the endianness of the host operating system. If the endianness of the file matches the endiannnes of the host, RLI reports

the encoding as UTF-16. If the endianness of file and host do not match, RLI reports the full encoding: UTF-16LE or UTF-16BE.

- Approximately 24,000 traditional Chinese words have been added to the Chinese dictionary.
- Replaced the Mac OS 10.4 (Darwin 8.9.1) platform for 32-bit Intel platform with the universal Mac OS 10.5 (Darwin 9) platform for 32-bit and 64-bit Intel.
- The `sparc-solaris9-gcc345` platform has been renamed to `sparc-solaris9-gcc34`.
- For Base Linguistics Language Analyzer [129] , added caching of morphological data about commonly used words in English and German to accelerate linguistic processing.

Consult the *RLP Release Notes* (**RLP-6.5.2-ReadMe.html**) for full details about changes to RLP in this release.

Chapter 1. Introduction to the Rosette Linguistics Platform

The Rosette Linguistics Platform (RLP) is the backbone of Basis Technology's text and language analysis technology. RLP provides advanced natural-language processing techniques to help your applications unlock information in unstructured text. RLP includes modules for language and encoding identification, converting text to Unicode, identifying basic linguistic features, and locating key entities like the names of people, places, and objects of interest. RLP supports English and a variety of Asian, European, and Middle Eastern languages. The detailed linguistic information provided by RLP can be used to increase the accuracy and depth of information-retrieval, text-mining, entity-extraction, and other text-analysis applications.

1.1. Key Features

RLP is packaged with modules for *Named Entity Extraction* (NE) and Base Linguistics (BL): *base noun phrase detection*, *tokenization*, *sentence boundary detection*, *part-of-speech tagging*, and morphological analysis including *stemming*, *alternative readings (transcriptions)*, and *compound analysis*. These modules can process Arabic, Chinese, Czech, Dutch, English, Farsi (Persian), French, German, Greek, Hungarian, Italian, Japanese, Korean, Polish, Portuguese, Russian, Spanish, and Urdu.

Language support for each of these operations is indicated in the following table:

Table 1.1. RLP Language Support for Base Linguistics (BL) and Named Entity Extraction (NE)

Language	Base Linguistics							NE
	Tokenization	POS	SBD	BNP	Stemming	Compounds	Readings	
Arabic	✓	✓	✓	✓	✓	n/a		✓
Chinese (Simplified)	✓	✓	✓	✓	n/a	n/a	✓	✓
Chinese (Traditional)	✓	✓	✓	✓	n/a	n/a	✓	✓
Czech	✓	✓	✓		✓	n/a		✓
Dutch	✓	✓	✓	✓	✓	✓		✓
English[a]	✓	✓	✓	✓	✓	n/a	n/a	✓
Farsi (Persian)	✓		✓		✓	n/a		✓
French	✓	✓	✓	✓	✓	n/a		✓
German	✓	✓	✓	✓	✓	✓		✓
Greek	✓	✓	✓		✓	n/a		✓
Hungarian	✓	✓	✓		✓	✓		✓
Italian	✓	✓	✓	✓	✓	n/a		✓
Japanese	✓	✓	✓	✓	✓	✓	✓	✓
Korean	✓	✓	✓		✓	✓		✓
Polish	✓	✓	✓		✓	n/a		✓
Portuguese	✓	✓	✓	✓	✓	n/a		✓
Russian	✓	✓	✓		✓	n/a		✓

Language	**Base Linguistics**							**NE**
	Tokenization	**POS**	**SBD**	**BNP**	**Stemming**	**Compounds**	**Readings**	
Spanish	✓	✓	✓	✓	✓	n/a		✓
Urdu	✓		✓		✓	n/a		✓

[a]RLP also provides specialized support for upper-case English text. When processing English text that is entirely upper case, specify the English Upper-Case language code (`en_uc`).

POS is part-of-speech tagging, *SBD* is sentence-boundary detection, and *BNP* is base-noun-phrase detection. For Chinese, the readings are pinyin transcriptions; for Japanese, the readings are Furigana transcriptions rendered in Hiragana. Blanks in this table indicate that the functionality is not available; *n/a* indicates that the feature does not apply to that language.

If you work with multilingual input data, RLP provides tools for locating regions of contiguous text in a single language, so that you can process each region with the appropriate language processors.

In addition to the languages listed above, the Rosette Language Identifier (RLI) [178] can identify text in the following languages: Albanian, Transliterated Arabic, Bahasa Indonesia, Bahasa Malay, Bengali, Bulgarian, Catalan, Croatian, Danish, Estonian, Finnish, Gujarati, Hebrew, Hindi, Icelandic, Kannada, Kurdish, Latvian, Lithuanian, Malayalam Norwegian, Pashto, Transliterated Pashto, Transliterated Farsi (Persian), Romanian, Serbian (Cyrillic and Latin), Slovak, Slovenian, Somali, Swedish, Tagalog, Telugu, Thai, Turkish, Ukrainian, Transliterated Urdu, Uzbek (Cyrillic and Latin), and Vietnamese.

Other key features:

- RLP is written in a portable subset of ISO/ANSI C++.
- C++, C, Java, and .NET APIs are available. The APIs do not vary from one human language to another.

  Text is internally encoded in Unicode (UTF-16).
- RLP operations are thread safe.

Note

RLP's features are enabled by license keys issued by Basis Technology. Please contact us to obtain the required evaluation or production license file, and refer to Installing RLP [5] for information about where to put the license file.

1.2. Architecture Overview

RLP consists of a collection of language *processors*, tied together by a common runtime environment. Each processor is responsible for performing a particular task, such as linguistic analysis or entity extraction, and generating a set of result data that applications can use and manipulate. The language processors are dynamically loaded; they are shared-object libraries on Unix systems and dynamically linked libraries (DLLs) on Windows.

All of the language processors available to the system -- as defined by the RLP license -- are kept in an *environment*. A *context* specifies an ordered set of language processors used to perform a particular task. A single environment can support multiple concurrent contexts. The language processors and their related data (such as dictionaries or language models) are stored in one location where they may be shared by different contexts.

Each context provides access to the results generated by the language processors in that context.

1.2.1. RLP Language Processors

The RLP language processors perform a variety of roles: detecting the encoding, MIME type (document type), and language of the input; stripping markup (such as HTML tags); converting the input to a standard Unicode form for further processing; normalizing variant spellings; performing various types of linguistic analysis; and generating output that other language processors or the application can use.

An application uses an ordered sequence of language processors (a context) to process the input text. In many cases, one language processor requires the output of another language processor as input for its own operation.

1.2.2. RLP Environments and Contexts

The RLP environment represents the global state of RLP, including license information and the language processors you can use. The environment loads and maintains the various language processors. It tracks all the language processors and can delay the loading and initialization of language processors until they are actually used.

An RLP context specifies an ordered series of language processors. Created by the environment, a single context defines a set of operations to be performed on a document.

For example, a context for extracting Japanese named entities might consist of the Unicode Converter (for converting UTF-8, for example, to the appropriate form of UTF-16 for the platform), the Japanese Language Analyzer (for tokenization and POS tagging), the Sentence Boundary Detector, the Base Noun Phrase detector, the Named Entity Extractor, the Gazetteer (to find named entities listed in gazetteers), the Regular Expression processor (to find named entities, such as dates or email addresses, identified with regular expressions), the Named Entity Redactor (for resolving duplication or overlaps), and the REXML processor (for generating a report).

A context for performing Japanese named entity extraction:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE contextconfig SYSTEM "http://www.basistech.com/dtds/2003/
    contextconfig.dtd">
<contextconfig>
  <languageprocessors>
    <language processor>Unicode Converter</languageprocessor>
    <!-- Japanese Language Analyzer -->
    <languageprocessor>JLA</languageprocessor>
   <languageprocessor>SentenceBoundaryDetector</languageprocessor>
    <languageprocessor>BaseNounPhrase</languageprocessor>
    <languageprocessor>NamedEntityExtractor</languageprocessor>
    <languageprocessor>Gazetteer</languageprocessor>
     <!-- Regular Expression-->
    <languageprocessor>RegExpLP</languageprocessor>
     <!-- Named Entity Redactor -->
    <languageprocessor>NERedactLP</languageprocessor>
    <!-- REXML generates an XML report -->
    <languageprocessor>REXML</languageprocessor>
  </languageprocessors>
</contextconfig>
```

Multiple contexts can be created from a single environment, and contexts can be executed in their own threads, given that all RLP processors are *re-entrant*.

An environment can be used in more than one processing thread with no locking. A context cannot be shared by multiple threads.

1.2.3. RLP Configuration

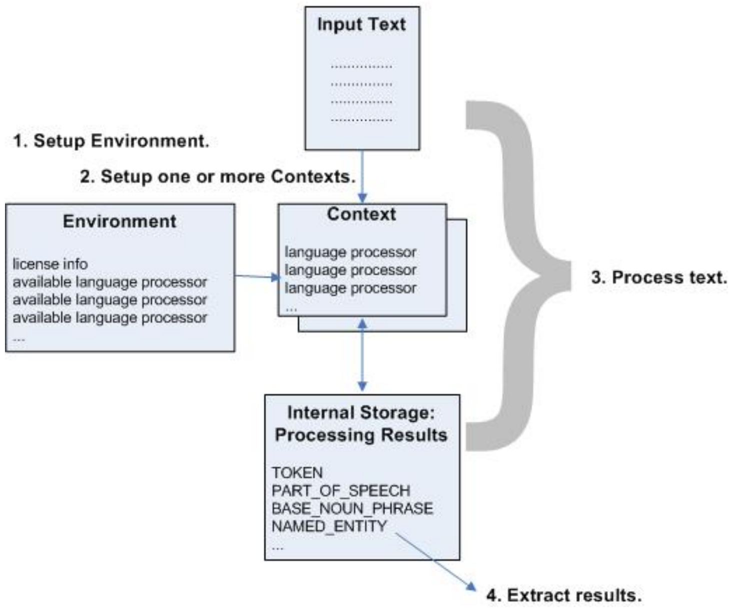
Processing resources are made available to an application by defining an environment and creating contexts within that environment. An environment defines licensing information and available processors. A context defines which processors will be run and in what order when RLP processes input text. You use an XML file or XML string to define the environment, and another XML file or string to define a context.

1.2.4. RLP Result Data

The application of a context to the input text generates a set of result data. You may use a language processor to display this result data to the user. The context also provides programmatic access to this data.

1.2.5. Core of an RLP Application

An RLP application defines an environment and one or more contexts, applies each context to input text, collects the result data of interest, and manipulates that data as required to meet its goals. The combination of environment, contexts, and result data provides a flexible framework for developing natural language processing (NLP) applications.



Chapter 2. RLP Getting Started

This chapter shows you how to download and install RLP, and how to use the RLP command-line utility to process sample data and your own text.

2.1. Downloading RLP

To install and use RLP, you need to download three files:

- A compressed SDK package file

  The SDK package must be the correct package for your platform. See Supported Platforms [13] .

- A compressed documentation package file

  The documentation package includes this book, release notes, and HTML API references for the C++, C, and Java APIs.

- A license file (**rlp-license.xml**)

  The license contains a set of keys that define the language operations you are authorized to perform with RLP.

When you contact Basis Technology to obtain a copy of RLP (for production use or for evaluation), we send you an email with private download links to the license file (**rlp-licenses.xml**), the SDK package, and the documentation package. These links expire after 30 days. If you need an extension, please contact `ProductSupport@basistech.com` .

Click the links to download these files to your workstation.

> The SDK is a **.zip** file for Windows or a compressed archive (**.tar.gz**) for Unix. The file names include version number and platform designation. See Supported Platforms [13] .
>
> The documentation is a **.zip** file for Windows or a compressed archive (**.tar.gz**) for Unix. See Documentation package file name [15] .

2.2. Installing the RLP SDK

Extract the SDK compressed package file you downloaded, and put the RLP license file in the RLP licenses directory.

1. Extract the SDK compressed package file to a directory on your local volume

   Windows example (with **rlp-6.5.2-sdk-ia32-w32-msvc80.zip**):
   a. In **Explorer**, double click the package file.
   b. In the Compressed Folder window that appears, use the **Extraction Wizard** to extract the package to a directory (such as **C:\rlp-6.5.2-sdk-ia32-w32-msvc80**).

   Unix example (with **rlp-6.5.2-sdk-1a32-glibc22-gcc32.tar.gz**):
   a. Put the file you have downloaded in the directory where you want to install the SDK (such as **/usr/local/BasisTech/BT_RLP_6.5.2**).
   b. Extract the package in that directory.

```
mv rlp-6.5.2-sdk-1a32-glibc22-gcc32.tar.gz /usr/local/BasisTech/BT_RLP_6.5.2
cd /usr/local/BasisTech/BT_RLP_6.5.2
gunzip rlp-6.5.2-sdk-1a32-glibc22-gcc32.tar.gz
tar -xf rlp-6.5.2-sdk-1a32-glibc22-gcc32.tar
```

Installation Directory: BT_ROOT.
If you install other Basis Technology SDKs that use the **RLP SDK**, such as the **RLP Name Components(RLP-NC) SDK**, you should install these SDKs in the same installation directory. The documentation uses `BT_ROOT` or Basis root directory to designate the installation directory. RLP applications must set the path to the Basis root directory (for example, **C:\rlp-6.5.2-sdk-ia32-w32-msvc80** or **/usr/local/BasisTech/BT_RLP_6.5.2**), so make a note of it for future use.

2. Copy the license file provided in the installation package into ***BT_ROOT*/rlp/rlp/licenses**. RLP does not run without a valid license file. If you wish to upgrade from an evaluation license or add support for another language or another RLP feature, contact Basis Technology Corp. at `ProductSupport@basistech.com` .

3. **Unix only.** Add the RLP library directory to the `LD_LIBRARY_PATH` environment variable (or its equivalent for your Unix operating system). The RLP library directory is ***BT_ROOT*/rlp/lib/ *BT_BUILD*** , where *BT_BUILD* is the platform identifier embedded in your SDK package file name (see Supported Platforms [13]). For example:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:\
/usr/local/BasisTech/BT_RLP_6.5.2/rlp/lib/ia32-glibc22-gcc32/
```

2.3. Installing the RLP Documentation

Extract the compressed documentation package **.zip** or **.tar.gz** [15] file you downloaded to the ***BT_ROOT*/rlp/doc** directory.

The RLP documentation set includes the following:

- Release Notes (**RLP-6.5.2-ReadMe.html**) with up-to-date information about new features and bug fixes in this release
- The *RLP Application Developer's Guide* (contains this chapter)
- Online reference to the C++, C, Java, and .NET APIs

2.4. Running the RLP Command-line Utility

Before you begin integrating RLP with your applications (see Creating an RLP application [17]), you may want to use the RLP command-line utility to help you understand the capabilities of RLP.

The RLP command-line utility is an executable: **rlp.exe** in Windows and **rlp** in Unix. This utility is in the binary directory:

BT_ROOT*/rlp/bin/*BT_BUILD

where *BT_BUILD* is the platform identifier embedded in your SDK package file name (see Supported Platforms [13]).

Windows example: If you installed **rlp-6.5.2-sdk-ia32-w32-msvc71.msi** to **C:\RLP-SDK-6.5.2**, the command line utility is

C:\RLP-SDK-6.5.2\rlp\bin\ia32-w32-msvc71\rlp.exe

Unix example: If you expanded **rlp-6.5.2-sdk-ia32-glibc22-gcc32.tar.gz** to **/usr/local/BasisTech/BT_RLP_6.5.2** the command line utility is

/usr/local/BasisTech/BT_RLP_6.5.2/rlp/bin/ia32-glibc22-gcc32/rlp

2.4.1. Using the go Script

The RLP distribution includes a script that you can use to run the RLP command-line utility with sample data: **go.bat** in Windows and **go.sh** in Unix. This script is in:

BT_ROOT*/rlp/samples/scripts/*BT_BUILD

Windows example: If you installed **rlp-6.5.2-sdk-ia32-w32-msvc71.msi** to **C:\RLP-SDK-6.5.2**, the `go` script is

C:\RLP-SDK-6.5.2\rlp\samples\scripts\ia32-w32-msvc71\go.bat

Unix example: If you expanded **rlp-6.5.2-sdk-ia32-glibc22-gcc32.tar.gz** to **/usr/local/BasisTech/BT_RLP_6.5.2** the `go` script is

/usr/local/BasisTech/BT_RLP_6.5.2/rlp/samples/scripts/ia32-glibc22-gcc32/go.sh

1. Use the command-line prompt (Windows) or bash shell (Unix) to navigate to the sample scripts directory.

   ### Warning

   To find the required resources and work correctly, the `go` script must be run from the sample scripts directory: ***BT_ROOT*/rlp/samples/scripts/*BT_BUILD*** .

   The RLP license file (**rlp-licenses.xml**) must be in ***BT_ROOT*/rlp/rlp/licenses**. If it is not, RLP will not run.

2. Run the `go` script with one argument: the two-letter ISO639 code (more than two letters in special cases, such as Simplified Chinese) for a language for which you have an RLP license. Use one of the codes from the following table.

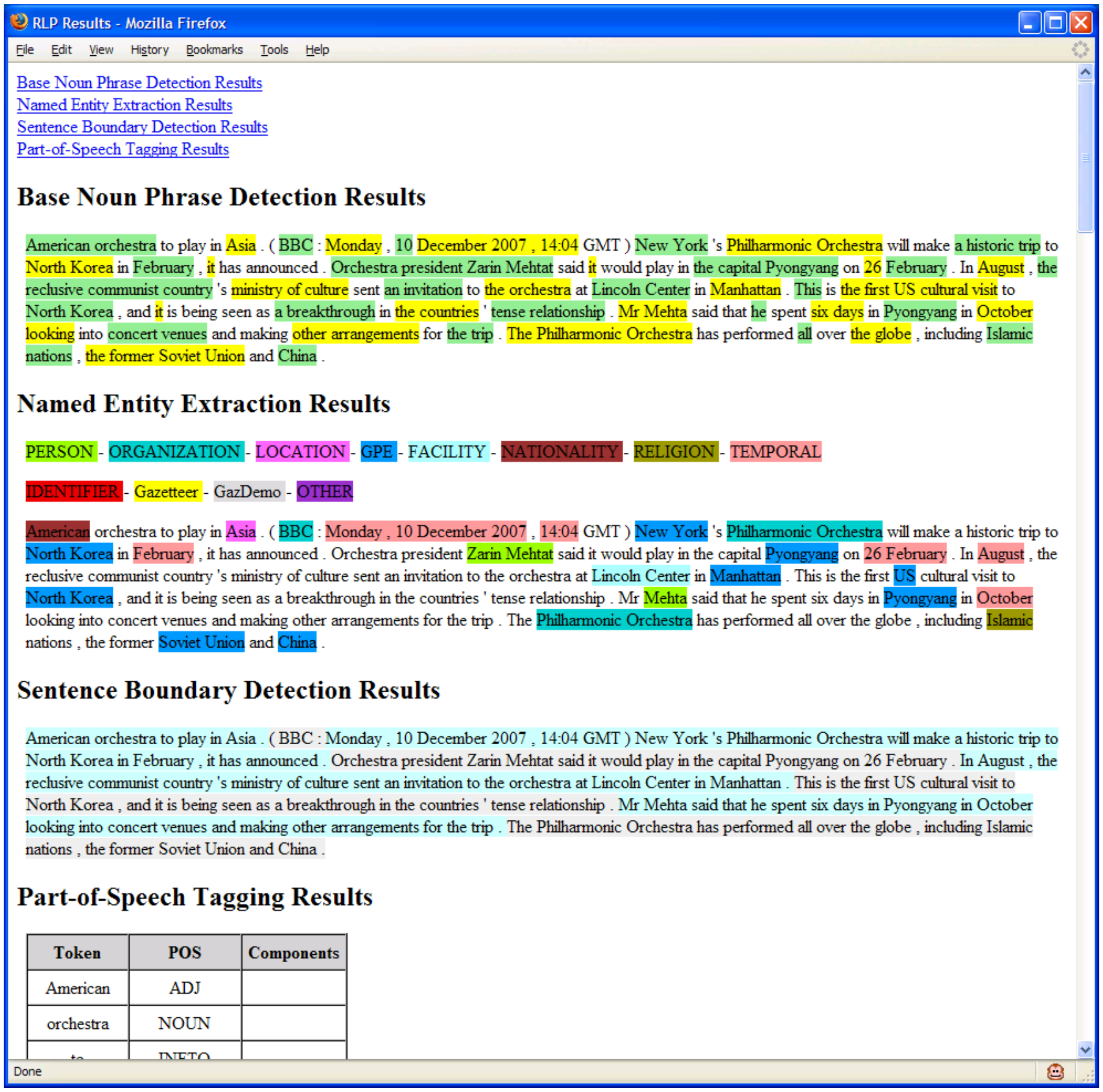
Language of text to analyze	Windows command	Unix command
Arabic	`go.bat ar`	`./go.sh ar`
Chinese - Simplified	`go.bat zh_sc`	`./go.sh zh_sc`
Chinese - Traditional	`go.bat zh_tc`	`./go.sh zh_tc`
Czech	`go.bat cs`	`./go.sh cs`
Dutch	`go.bat nl`	`./go.sh nl`
English	`go.bat en`	`./go.sh en`
Farsi (Persian)	`go.bat fa`	`./go.sh fa`
French	`go.bat fr`	`./go.sh fr`
German	`go.bat de`	`./go.sh de`
Greek	`go.bat el`	`./go.sh el`
Hungarian	`go.bat hu`	`./go.sh hu`
Italian	`go.bat it`	`./go.sh it`

Language of text to analyze	**Windows command**	**Unix command**
Japanese	`go.bat ja`	`./go.sh ja`
Korean	`go.bat ko`	`./go.sh ko`
Polish	`go.bat pl`	`./go.sh pl`
Portuguese	`go.bat pt`	`./go.sh pt`
Russian	`go.bat ru`	`./go.sh ru`
Spanish	`go.bat es`	`./go.sh es`
Urdu	`go.bat ur`	`./go.sh ur`

The RLP command-line utility analyzes the input text and generates an XML report on what it finds. The report includes an `xml-stylesheet` processing instruction.

Use your browser to open **rlp-output.xml**. The browser applies the `xml-stylesheet` to the file and displays the resulting HTML.

Figure 2.1. Report on English Sample Text


RLP Results - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Done

Base Noun Phrase Detection Results
Named Entity Extraction Results
Sentence Boundary Detection Results
Part-of-Speech Tagging Results

Base Noun Phrase Detection Results

American orchestra to play in Asia . (BBC : Monday , 10 December 2007 , 14:04 GMT) New York 's Philharmonic Orchestra will make a historic trip to North Korea in February , it has announced . Orchestra president Zarin Mehtat said it would play in the capital Pyongyang on 26 February . In August , the reclusive communist country 's ministry of culture sent an invitation to the orchestra at Lincoln Center in Manhattan . This is the first US cultural visit to North Korea , and it is being seen as a breakthrough in the countries ' tense relationship . Mr Mehta said that he spent six days in Pyongyang in October looking into concert venues and making other arrangements for the trip . The Philharmonic Orchestra has performed all over the globe , including Islamic nations , the former Soviet Union and China .

Named Entity Extraction Results

PERSON - ORGANIZATION - LOCATION - GPE - FACILITY - NATIONALITY - RELIGION - TEMPORAL

IDENTIFIER - Gazetteer - GazDemo - OTHER

American orchestra to play in Asia . (BBC : Monday , 10 December 2007 , 14:04 GMT) New York 's Philharmonic Orchestra will make a historic trip to North Korea in February , it has announced . Orchestra president Zarin Mehtat said it would play in the capital Pyongyang on 26 February . In August , the reclusive communist country 's ministry of culture sent an invitation to the orchestra at Lincoln Center in Manhattan . This is the first US cultural visit to North Korea , and it is being seen as a breakthrough in the countries ' tense relationship . Mr Mehta said that he spent six days in Pyongyang in October looking into concert venues and making other arrangements for the trip . The Philharmonic Orchestra has performed all over the globe , including Islamic nations , the former Soviet Union and China .

Sentence Boundary Detection Results

American orchestra to play in Asia . (BBC : Monday , 10 December 2007 , 14:04 GMT) New York 's Philharmonic Orchestra will make a historic trip to North Korea in February , it has announced . Orchestra president Zarin Mehtat said it would play in the capital Pyongyang on 26 February . In August , the reclusive communist country 's ministry of culture sent an invitation to the orchestra at Lincoln Center in Manhattan . This is the first US cultural visit to North Korea , and it is being seen as a breakthrough in the countries ' tense relationship . Mr Mehta said that he spent six days in Pyongyang in October looking into concert venues and making other arrangements for the trip . The Philharmonic Orchestra has performed all over the globe , including Islamic nations , the former Soviet Union and China .

Part-of-Speech Tagging Results

Token	POS	Components
American	ADJ	
orchestra	NOUN	
to	INFTO	

2.4.2. What Takes Place When You Run the go Script

The `go` script passes the following parameters to the RLP command-line utility:

1. The ISO639 language code you supply when you call `go`
2. `BT_ROOT`, the Basis root directory

   The RLP command-line utility begins by using this parameter to set the Basis root directory.
3. Environment

The environment defines the scope of operations available to RLP during this session and provides a pointer to your RLP license, which specifies which operations you are licensed to perform. The environment is specified with an XML configuration file: ***`BT_ROOT`*****/rlp/etc/rlp-global.xml**.

4. Context

   The context defines the sequence of processors that the RLP command-line utility applies to the sample input text. The context is defined with an XML configuration file. The `go` script specifies ***`BT_ROOT`*****/rlp/samples/etc/rlp-context.xml**.

5. Input text

   The input text is a UTF-8 text file in ***`BT_ROOT`*****/rlp/samples/data**. The file name takes the form **<*ln*>-text.txt** where <*ln*> is the language code you enter on the command line. For example, the sample file for English is **en-text.txt**.

The RLP command-line utility does the following:

1. Sets the Basis root directory.
2. Uses the environment configuration file to set up the runtime environment.
3. Uses the context configuration file to create a context object.
4. Uses the context object to process the input text. The context object defined with **rlp-context.xml** performs the following tasks:

   a. Converts the text to UTF-16.

   b. Tokenizes the text (each token is a word, multiword expression, possessive affix, or punctuation).

   c. Tags the part of speech for each token (not currently supported for Farsiand Urdu).

   d. Locates sentence boundaries.

   e. Identifies noun phrases (not supported for some languages).

   f. Finds named entities of various types (not supported for some languages).

   g. Generates an XML report with an `xml-stylesheet` processing instruction so browsers can display the report as HTML.

If the XML report you see is missing any of this information, your RLP license does not authorize one or more processors for the language you selected. To upgrade your license, contact Basis Technology Corp. at `ProductSupport@basistech.com` . *Note:* Named Entity extraction, Noun Phrase extraction, and Part-of-Speech detection are not supported for some languages.

2.4.3. Using the RLP Command-Line Utility to Process Your Own Text

Instead of using the `go` script, you can run the RLP command-line utility with the necessary arguments from the command line or with your own batch file or shell script. Call **rlp** with arguments for language, RLP root directory, environment configuration file, context configuration file, and input file: [1]

rlp[.exe] [-l *`ln`*] -root *`BT_ROOT`* *`envConfig contextConfig inputFile`* [-v]

[1]On Unix platforms, you must set `LD_LIBRARY_PATH` (or the equivalent environment variable for your platform) to include ***`BT_ROOT`*****.rlp/lib/*`BT_BUILD`*.**

`ln` is the two-letter ISO639 language code. You must include **-l** ***`ln`***, unless your context includes the RLP Language Identifier (RLI) and you RLP license authorizes the use of RLI.

Table 2.1. ISO639 Language Codes

Language	Code
Arabic	ar
Chinese - Simplified	zh_sc
Chinese - Traditional	zh_tc
Czech	cs
Dutch	nl
English	en
Upper-Case English[a]	en_uc
Farsi (Persian)	fa
French	fr
German	de
Greek	el
Hungarian	hu
Italian	it
Japanese	ja
Korean	ko
Polish	pl
Portuguese	pt
Russian	ru
Spanish	es
Urdu	ur

[a]For more accurate processing of English text that is entirely upper case, use the `en_uc` language code.

`BT_ROOT` is the Basis Technology root directory. You must include **-root** *`BT_ROOT`*.

`envConfig` is the pathname of the environment configuration file.

`contextConfig` is the pathname of the context configuration file.

`inputFile` is the pathname of the input file.

`-v` instructs the utility to write the version of RLP to the console.

Example 2.1. Processing a Unicode Input File

If your text is in a Unicode format (UTF-8, UTF-16, or UTF-32), you can use the same context configuration file the `go` script uses. This context configuration includes the Language Identifier (RLI), so you can also use it to process Unicode files for which you do not know the language.

1. Run the RLP command-line utility with the following parameters:

   [-l *`ln`*] -root *`BT_ROOT envConfig contextConfig inputFile`*

`ln` is the ISO639 language code. If you do not know the language code, you can omit this parameter: RLI determines the language.

`BT_ROOT` is the pathname of the Basis root directory.

`envConfig` is the pathname of *`BT_ROOT`* **/rlp/etc/rlp-global.xml**.

`contextConfig` is the pathname of *`BT_ROOT`* **/rlp/samples/etc/rlp-context.xml**.

`inputFile` is the pathname of the Unicode input file.

2. Examine the report the output processor generates: **rlp-output.xml**.

Example 2.2. Processing Non-Unicode Text

If the text file you want to process is not Unicode, and your license includes RLI and RCLU, use the Language Identifier processor (RLI) and the Core Library for Unicode (RCLU). RLI identifies the language and the encoding. RCLU can convert from many text encodings to UTF-16. For information about RLP requirements for text encoding, see Preparing your Data for Processing [77] .

To handle such text you need a context configuration file with RLI and RCLU, both listed before the processors that tokenize the text, tag parts of speech, and locate noun phrases, named entities, and sentence boundaries.

The sample context configuration file **rlp-context-rclu.xml** includes the required processors.

1. Run The RLP command-line utility with the following parameters:

   [-l *`ln`*] -root *`BT_ROOT envConfig contextConfig inputFile`*

   If you know the language, include **-l** and the two-letter ISO639 language code. If you do not know the language, do not include this parameter.

   `BT_ROOT` is the Basis root directory.

   `envConfig` is the pathname of the environment configuration file: ***`BT_ROOT`*/rlp/etc/rlp-global.xml**.

   `contextConfig` is the pathname to the context configuration file that starts with RLI and RCLU: ***`BT_ROOT`*/rlp/samples/etc/rlp-context-rclu.xml**.

   `inputFile` is the pathname to the input text file.

2. Examine the report the REXML processor generates: **rlp-output.xml**.

2.4.3.1. Other Uses

By using the RLP command-line utility to apply a different context configuration file to a different input file, you can generate a wide variety of result data. For more information about context configurations, see Defining an RLP context [18] .

You can include command-line parameters to specify encoding and property values to be passed to RLP processors.

For example, you can pass a `com.basistech.rexml.output_pathname` property setting to instruct REXML to direct output to a file. See REXML [165] .

For a complete list of the arguments that the RLP command-line utility accepts, run

```
rlp[.exe] -h
```

The following table lists the sample context configuration files that are shipped with RLP. They are in *BT_ROOT* **/rlp/samples/etc**.

Sample Context Configuration File	Purpose
rlp-context.xml	General purpose for Unicode input: generates UTF-16, detects language, handles text in all supported languages, identifies named entities, generates XML report.
rlp-context-no-op.xml	General purpose for Unicode input, with no XML report.
rlp-context-rclu.xml	General purpose for non-Unicode input.

2.5. Using the Windows Demo

For Windows users, we supply a separate package that includes a GUI demo. See The Rosette Demo [257] .

2.6. Supported Platforms and BT_BUILD Values

You must install an SDK package that is appropriate for your platform. The supported platforms and C++ compilers are listed in the table below.

If you are planning to use the C++ (or C) API, use the SDK package that incorporates the compiler you plan to use.

If you are planning to use the Java API, you can use any Java SDK 1.5 package (or later) built for your OS version and architecture. Java is supported except where noted otherwise.

If your platform and compiler do not appear in the following list, please contact Basis Technology Corp. at `ProductSupport@basistech.com` .

Table 2.2. Supported Platforms

OS	CPU	Compiler	BT_BUILD[a]
AIX 5.2	PowerPC	xlc 5.2	ppc-aix52-xlc[b]
FreeBSD 4.8	IA32	3.4	ia32-freebsd48-gcc34[b]
FreeBSD 6.0	AMD64	gcc 3.4.4	amd64-freebsd6-gcc344[b]
FreeBSD 6.0	IA32	gcc 3.4.4	ia32-freebsd6-gcc344[b]
HP-UX 11.00	IA64	HP aCC 5.41	ia64-hpux11-aCC541[b]
HP-UX 11.00	PA-RISC32	HP aCC A.03.33	parisc-hpux11-aCC333-aa[b]
Linux (glibc 2.2)	IA32	gcc 3.2	ia32-glibc22-gcc32
Linux (glibc 2.3)	AMD64	gcc 3.4	amd64-glibc23-gcc34
Linux (glibc 2.3)	AMD64	gcc 4.0	amd64-glibc23-gcc40
Linux (glibc 2.3)	IA32	gcc 3.2	ia32-glibc23-gcc32
Linux (glibc 2.3)	IA32	gcc 3.4.4	ia32-glibc23-gcc34
Linux (glibc 2.3)	IA32	gcc 4.0	ia32-glibc23-gcc40

OS	**CPU**	**Compiler**	**BT_BUILD**[a]
Linux (glibc 2.4)	AMD64	gcc 4.1	amd64-glibc24-gcc41
Linux (glibc 2.4)	IA32	gcc 4.1	ia32-glibc24-gcc41
Linux (glibc 2.5)	AMD64	gcc 4.1	amd64-glibc25-gcc41
Linux (glibc 2.5)	AMD64	gcc 4.2	amd64-glibc25-gcc42
Linux (glibc 2.5)	IA32	gcc 4.1	ia32-glibc25-gcc41
Linux (glibc 2.5)	IA32	gcc 4.2	ia32-glibc25-gcc42
MAC OS 10.5 (Darwin 9)	32-bit/64-bit Intel	gcc 4.0	universal-darwin9-gcc40
Solaris 10	AMD64	CC 5.8	amd64-solaris10-cc58
Solaris 10	AMD64	gcc 4.1.2	amd64-solaris10-gcc41
Solaris 10	IA32	CC 5.8	ia32-solaris10-cc58
Solaris 10	IA32	gcc 3.4	ia32-solaris10-gcc34
Solaris 10	SPARC32	CC 5.8	sparc-solaris10-cc58
Solaris 10	SPARC64	CC 5.8	sparc-solaris10-cc58-64
Solaris 10	SPARC64	gcc 4.1.2	sparc-solaris10-gcc412-64
Solaris 7-8	SPARC32	CC 5.2 (Forte Developer 6)	sparc-solaris28-cc52
Solaris 7-8	SPARC64	CC 5.2 (Forte Developer 6)	sparc-solaris28-cc52-64
Solaris 9	IA32	gcc 3.4.5	ia32-solaris9-gcc34
Solaris 9	SPARC32	CC 3.4	sparc-solaris9-gcc34
Solaris 9	SPARC32	CC 5.8 (Sun Studio 11)	sparc-solaris9-cc58
Solaris 9	SPARC64	CC 5.8 (Sun Studio 11)	sparc-solaris9-cc58-64
Solaris 9	SPARC64	gcc 4.1	sparc-solaris9-gcc41-64
Windows 32	IA32	Visual Studio 7.1	ia32-w32-msvc71
Windows 32	IA32	Visual Studio 7.1	ia32-w32-msvc71-static[c]
Windows 32	IA32	Visual Studio 8.0	ia32-w32-msvc80
Windows 32	IA32	Visual Studio 8.0	ia32-w32-msvc80-static[c]
Windows 64	AMD64	Visual Studio 8.0	amd64-w64-msvc80
Windows 64	AMD64	Visual Studio 8.0	amd64-w64-msvc80-static[c]

[a] `BT_BUILD` is embedded in the name of the download package. It is also the subdirectory name used in various locations for platform-specific files, such as binary library files.

[b]Java not supported.

[c]Built with a statically linked library. Does not include support for the Java API, the Core Library for Unicode (RCLU) [168] , iFilter [147] , or HTML Stripper [146] .

2.6.1. SDK Package File Name

The compressed SDK package file names take the form

rlp- *<ver>* **-sdk-** *BT_BUILD* `.` *<ext>*

where `<ver>` is RLP version (6.5.2 for the 6.5.2 release), *BT_BUILD* is in the table above, and `<ext>` is `tar.gz` for Unix platforms and `.exe` for Windows.

For the RLP 6.5.2 release, the package file names are:

- **rlp-6.5.2-sdk-ppc-aix52-xlc.tar.gz**
- **rlp-6.5.2-sdk-ia32-freebsd48-gcc34.tar.gz**
- **rlp-6.5.2-sdk-amd64-freebsd6-gcc344.tar.gz**
- **rlp-6.5.2-sdk-ia32-freebsd6-gcc344.tar.gz**
- **rlp-6.5.2-sdk-ia64-hpux11-aCC541.tar.gz**
- **rlp-6.5.2-sdk-parisc-hpux11-aCC333-aa.tar.gz**
- **rlp-6.5.2-sdk-ia32-glibc22-gcc32.tar.gz**
- **rlp-6.5.2-sdk-amd64-glibc23-gcc34.tar.gz**
- **rlp-6.5.2-sdk-amd64-glibc23-gcc40.tar.gz**
- **rlp-6.5.2-sdk-ia32-glibc23-gcc32.tar.gz**
- **rlp-6.5.2-sdk-ia32-glibc23-gcc34.tar.gz**
- **rlp-6.5.2-sdk-ia32-glibc23-gcc40.tar.gz**
- **rlp-6.5.2-sdk-amd64-glibc24-gcc41.tar.gz**
- **rlp-6.5.2-sdk-ia32-glibc24-gcc41.tar.gz**
- **rlp-6.5.2-sdk-amd64-glibc25-gcc41.tar.gz**
- **rlp-6.5.2-sdk-amd64-glibc25-gcc42.tar.gz**
- **rlp-6.5.2-sdk-ia32-glibc25-gcc41.tar.gz**
- **rlp-6.5.2-sdk-ia32-glibc25-gcc42.tar.gz**
- **rlp-6.5.2-sdk-universal-darwin9-gcc40.tar.gz**
- **rlp-6.5.2-sdk-amd64-solaris10-cc58.tar.gz**
- **rlp-6.5.2-sdk-amd64-solaris10-gcc41.tar.gz**
- **rlp-6.5.2-sdk-ia32-solaris10-cc58.tar.gz**
- **rlp-6.5.2-sdk-ia32-solaris10-gcc34.tar.gz**
- **rlp-6.5.2-sdk-sparc-solaris10-cc58.tar.gz**
- **rlp-6.5.2-sdk-sparc-solaris10-cc58-64.tar.gz**
- **rlp-6.5.2-sdk-sparc-solaris10-gcc412-64.tar.gz**
- **rlp-6.5.2-sdk-sparc-solaris28-cc52.tar.gz**
- **rlp-6.5.2-sdk-sparc-solaris28-cc52-64.tar.gz**
- **rlp-6.5.2-sdk-ia32-solaris9-gcc34.tar.gz**
- **rlp-6.5.2-sdk-sparc-solaris9-gcc34.tar.gz**
- **rlp-6.5.2-sdk-sparc-solaris9-cc58.tar.gz**
- **rlp-6.5.2-sdk-sparc-solaris9-cc58-64.tar.gz**
- **rlp-6.5.2-sdk-sparc-solaris9-gcc41-64.tar.gz**
- **rlp-6.5.2-sdk-ia32-w32-msvc71.zip**
- **rlp-6.5.2-sdk-ia32-w32-msvc71-static.zip**
- **rlp-6.5.2-sdk-ia32-w32-msvc80.zip**
- **rlp-6.5.2-sdk-ia32-w32-msvc80-static.zip**
- **rlp-6.5.2-sdk-amd64-w64-msvc80.zip**
- **rlp-6.5.2-sdk-amd64-w64-msvc80-static.zip**

2.6.2. Documentation Package File Name

The documentation is available in a **.zip** file for Windows and a **.tar.gz** for Unix. (The contents are identical.)

For the RLP 6.5.2 release, the English documentation package file names are:

- **rlp-6.5.2-doc-unix.tar.gz**
- **rlp-6.5.2-doc-win.zip**

The Japanese documentation package files names are:

- **rlp-6.5.2-doc-ja-unix.tar.gz**
- **rlp-6.5.2-doc-ja-win.zip**

Chapter 3. Creating an RLP Application

This document walks you through the process of creating an RLP application to extract information from text. Prior to running the application, you may or may not know the language of the text, but the text is assumed to all be in the same language. For information about how to process text input that may contain passages in different languages, see Processing Multilingual Text [73] .

3.1. Overview

- Define the objectives [17]
- Define an RLP environment [17]
- Define an RLP context [18]
- Prepare the input [18]
- Write the application [20]
- Build and run the application [40]

3.2. Defining the Objectives

You want to extract useful information from text, information that may be gleaned with a combination of linguistic analysis, dictionaries, and word lists that map words and phrases to entities of interest, such as geographical place names, the names of people, organizations, and so on. For an overview of the linguistic analysis that RLP can perform with various languages, see Key Features [1] .

Perhaps you want all the words that appear in the text, in which case you should decide whether you want the words as they appear in text (tokens) or in their dictionary form (stems). You may want nouns or noun phrases. If the input is Arabic, you may want vocalized transliterations of the names that appear in the text. You may want all sentences in which a particular verb or noun, or named entity appears. You may want RLP to identify the language or to process streams of text in multiple languages. Once you have determined what kind of data you need, you can define the RLP context that generates the relevant result data.

Based on your objectives and the language and encoding of your input text, you will define an environment and a context to process the text. Applying the context to the input text generates the result data that your application can use.

For information about the types of data that RLP can produce, see RLP Result Types [83] .

3.3. Defining an RLP Environment

An RLP runtime environment maintains ownership of language processors, and their associated data. It also defines the path to your RLP license file.

RLP is distributed with an environment configuration file that you can use as is: ***`BT_ROOT`*****/rlp/etc/rlp-global.xml**, where ***`BT_ROOT`*****/rlp** is the Basis root directory.

You may want to edit the `preload` setting for some processors. If a processor is used frequently, setting `preload` to `"yes"` may improve performance. When `preload` is set to `"no"`, RLP does not load the processor until it is actually used. You can also remove any processors that you do not plan to use in the applications that use this environment configuration.

If you instantiate more than one environment object in a process, each object is in fact a wrapper for a single underlying environment object. Accordingly all of the environment objects must be initialized with the same environment configuration (normally ***BT_ROOT*/rlp/etc/rlp-global.xml**).

3.4. Defining an RLP Context

The RLP context specifies an ordered list of language processors made available by the environment for processing the input text. [1] With the exception of the Unicode Converter, the processors that appear in the context are a subset of the processors listed in the environment configuration.

The RLP context object posts the input text to internal storage. All language processors get their input from and post results to internal storage. Some language processors process raw data and generate UTF-16 raw text. Other language processors scan the UTF-16 raw text and post tokens, named entities, etc. A given processor may depend on other processors, which means it requires as input the output generated by one or more of the processors that precede it in the context sequence.

The context determines which RLP results are created. In most cases, the body of your application works with these results.

3.4.1. Preparing the Input

The input must be in an encoding and a format that RLP can handle. For the details, see Preparing Your Data for Processing [77] .

To perform linguistic analysis, the RLP language processors work with text encoded as UTF-16. The byte order is big-endian or little-endian, depending on your platform. If your input text is UTF-16 in the correct byte order for your platform, no conversion of the input text is required. If the input includes a byte order mark (BOM), see Handling the BOM [78] .

For plain text in any standard encoding, you can use the `Unicode Converter`, followed by the `RLI` and `RCLU` language processors. If the input is Unicode, `Unicode Converter` converts it to UTF-16 with the correct byte order for the current platform. If the input is not Unicode, `RLI` detects the encoding (`RLI` also detects the language). If the input is not Unicode, `RCLU` converts the text to the required form of UTF-16. See Preparing Text in Any Encoding [77] .

If the input includes markup in addition to text (HTML, XML, PDF, RTF, and Microsoft Office documents), you can use the `iFilter` processor (Windows only) or `HTML Stripper`, to remove the markup before linguistic analysis takes place. For more information see, Preparing Marked Up Input [79] .

3.4.2. Language Processors

Some language processors apply to a specific language or set of languages. Other language processors apply to many or all supported languages. For example, the `BL1` processor applies to a number of European languages, and `CLA` is for processing Simplified or Traditional Chinese, whereas the `Gazetteer`, `NamedEntityExtractor`, and `NERedactLP` apply to many languages.

The order in which processors appear in the context configuration determines the order in which they will run. Order is important. Some processors take their input from the output of a previous processor rather than directly from the input text. In other words, some processors depend on the inclusion of another processor. For example, you must run the `Tokenizer` and `SentenceBoundaryDetector` (in that order) before you can run `ARBL` (Arabic Base Linguistics), `FABL` (Farsi Base Linguistics), or `URBL` (Urdu Base Linguistics).

[1]In releases prior to RLP 5.2.0, processors were divided into three categories in an RLP context: an input processor, language processors, and an optional output processors. This distinction no longer exists. All RLP processors are now called language processors.

For information about each of the language processors, their settings and dependencies, see Language Processors [123] .

3.4.3. Language Analyzer User Dictionaries

Some language processors, called *Language Analyzers*, use dictionaries to help process text in a given language. RLP supplies one or more dictionaries for each supported language. For some languages and language analyzers, you can create one or more user dictionaries.
A user dictionary can contain words specific to an industry or application. User dictionaries may also specify segmentation behavior for existing words. For example, you may want to prevent a compound word that is a product name from being segmented.

Currently, user dictionaries are supported for the following language analyzers: Base Linguistics Language Analyzer (European languages), Chinese Language Analyzer, Japanese Language Analyzer, and Korean Language Analyzer. For more information about creating and using user dictionaries with these language analyzers, please see their specific documentation in RLP Processors [123] .

3.4.4. Context Properties

The context maintains a property list of name/value pairs. Many processors look for properties to control their behavior. For example, the REXML Processor redirects output to a file if the `com.basistech.rexml.output_pathname` context property is set to a valid file pathname. To see the context properties for a given language processor, see "Context Properties" for that processor in RLP Processors [123] .

The name of a processor-specific context property is prefixed by `com.basistech.<processor name>` , such as `com.basistech.rexml.output_pathname`. A global context property [125] may apply to more than one processor, and includes a prefix to indicate its scope, such as `com.basistech.bl.query` (`bl` stands for base linguistics).

Properties can be set via entries in the context configuration XML document. The following entry, for example, tells REXML to send its output (an XML report) to a file:

```
<property name="com.basistech.rexml.output_pathname" value="c:\reports\rexml-out.xml"/>
```

Context properties can also be set through the API. See Setting Context Properties [26] .

3.4.5. Sample Context Configurations

3.4.5.1. General Purpose: Language and Encoding Not Known

The following sample context uses `Unicode Converter` to convert Unicode input to UTF-16. If the input is not Unicode, the context uses `RLI` and `RCLU` to identify encoding and language, and convert the input to UTF-16. It then includes language processors for all the languages RLP supports. At runtime, any processors that are not needed to process the input are inactive. If you know that some languages will never appear, you can remove the corresponding language processors.

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

<contextconfig>
  <languageprocessors>
    <languageprocessor>Unicode Converter</languageprocessor>
    <languageprocessor>BL1</languageprocessor>
    <languageprocessor>JLA</languageprocessor>
  <languageprocessor>CLA</languageprocessor>
  <languageprocessor>KLA</languageprocessor>
  <languageprocessor>Tokenizer</languageprocessor>
  <languageprocessor>SentenceBoundaryDetector</languageprocessor>
 <languageprocessor>ARBL</languageprocessor>
 <languageprocessor>FABL</languageprocessor>
 <languageprocessor>URBL</languageprocessor>
  <languageprocessor>Stopwords</languageprocessor>
  <languageprocessor>BaseNounPhrase</languageprocessor>
  <languageprocessor>NamedEntityExtractor</languageprocessor>
  <languageprocessor>Gazetteer</languageprocessor>
  <languageprocessor>RegExpLP</languageprocessor>
  <languageprocessor>NERedactLP</languageprocessor>
 </languageprocessors>
</contextconfig>
```

3.4.5.2. Unicode Input and Base Linguistic Analysis for One Language

The following sample is much more specific. It uses the `Unicode Converter` to convert Japanese text from UTF-8 (or some other Unicode encoding) to UTF-16. The `JLA` language processor (Japanese Language Analyzer) segments Japanese text into separate words and assigns part-of-speech tags to each word. If `com.basistech.jla.deep_compound_decomposition` is set to `"true"` (the default is `"false"`), `JLA` recursively decomposes into smaller components any tokens marked in the dictionary as decomposable. `ManyToOneNormalizer` uses the sample Japanese normalization dictionary [154] to return the normalized form of Japanese word variants. This context does not identify named entities.

```
<?xml version="1.0"?>
<!DOCTYPE contextconfig SYSTEM "contextconfig.dtd">
<contextconfig>
 <properties>
  <property name="com.basistech.jla.deep_compound_decomposition" value="true"/>
 </properties>
 <languageprocessors>
  <languageprocessor>Unicode Converter</languageprocessor>
  <languageprocessor>JLA</languageprocessor>
  <languageprocessor>ManyToOneNormalizer</languageprocessor>
 </languageprocessors>
</contextconfig>
```

3.4.5.3. Other

For an example that handles HTML, PDF, or Microsoft Office documents, see Using iFilter [79] .

For an example that processes multilingual text, see RLBL Context [73] and Single-Language Context [74] .

3.5. Coding the Application

Choose your programming language: C++, C, Java (1.5 or later), or .NET. [2] RLP has implemented an API for each of these languages. In either case, the essential steps an application must complete are the same.

[2]See Using the RLP C API [59] .

Steps

C++ only: Verify that RLP runtime library is compatible with the library used for compilation.

1. Process parameters required to run the application.

   Standard parameters include path to the Basis root directory, environment configuration XML, context configuration XML, context and property settings, language and encoding of the input text.

2. Set up the RLP environment [21] .

   Set the Basis root directory, set up diagnostic logging, and instantiate and configure the environment. You may also want to collect information about the RLP features your license authorizes you to use.

3. Set up the RLP context [25] .

   Use the environment object and an XML context configuration file (or string) to instantiate a context object. Set context properties as appropriate (see RLP Processors [123]).

4. Use the context object to process the input [26] .

5. Handle the RLP results generated during the previous step. This is the heart of your application. For detailed information, see Accessing RLP Results [83] .

6. Release the resources you have allocated.

3.6. Setting Up the RLP Environment

This section contains information about initializing the runtime environment and capturing diagnostic log output.

3.6.1. Setting the Basis Root Directory

The Basis root directory is the directory in which you installed **RLP**.

If you are using C++, use `BT_RLP_Environment::SetBTRootDirectory` to set the Basis root directory.

If you are using Java, there are two ways to set the Basis root directory:

- (Preferred) Use `com.basistech.util.Pathnames.setBTRoot` to set the Basis root directory
- Set the `bt.root` system property. You can do this from the command line when you launch the Java virtual machine:

  ```
  java -Dbt.root=BT_ROOT ...
  ```

  where *`BT_ROOT`* is the path to the Basis root directory.

In Java, you must also create an `EnvironmentParameters` object before you proceed to the next step. For example:

```
EnvironmentParameters envParams = new EnvironmentParameters();
```

3.6.2. Capturing Log Output

To instruct RLP to capture diagnostic information, do the following:

1. Create a log callback function.
2. Register the log callback function.
3. Set the log level (specify the kinds of messages you want to log).

3.6.2.1. Log Channels and Levels

Setting the log level specifies which channels are communicated to the callback function. RLP provides three channels, each with its corresponding level:

Channel	Level	Description
0	warning	Non-fatal errors
1	error	Serious runtime errors
2	info	Helpful information

The default logging level is "error".

You can set logging level to a single channel (e.g., "warning"), a comma-delimited list of channels (e.g., "warning,error"), or "all" (equivalent to "warning,error,info"). If you want to mute all channels, set logging level to "none".

For information about the codes returned if you include the "error" level, see Error codes [249] .

3.6.2.2. Capturing Log Output in C++

Create a callback function that conforms to the following prototype:

```
void log_callback(void* info_p, int channel, const char* message)
```

`info_p` is the data passed as the first parameter when you use `SetLogCallbackFunction` to register the callback: it can be any value you want (or NULL).

`channel` is the channel number the message is being written to: `BT_LOG::WARNING_CHANNEL`, `BT_LOG::ERROR_CHANNEL`, or `BT_LOG::INFO_CHANNEL`.

`message` is the text of the message.

Here is a callback function that writes log messages to the open file that is established when the function is registered:

```
static log_callback(void* info, int channel, char const* message){
  fprintf((FILE*) info, "%d\t%s\n", channel, message);
}
```

The environment class provides static functions for registering the callback function and setting log level. For example:

```
//Register log_callback to write log messages to stderr.
BT_RLP_Environment::SetLogCallbackFunction((void*) stderr, log_callback);
//Post all channels to the callback.
BT_RLP_Environment::SetLogLevel("all");
```

3.6.2.3. Capturing Log Output in Java

Create a class that implements `RLPEnvironment.LogCallback`, which contains a `message()` method. Then register the callback, and set the log level. For example:

```
public class MyCallback implements RLPEnvironment.LogCallback {
  public void message(int channel, String message){
    System.err.println(channel + "\t" + message);
  }
}
//Use the EnvironmentParameters object to set log callback and log level.
envParams.setsetLogCallback(new MyCallback());
envParams.setLogLevel("all");
```

3.6.2.4. Alternative

If you want RLP to post messages to standard error, you can set the `BT_RLP_LOG_LEVEL` environment value to indicate which channels you want sent to standard error (warnings, errors, and/or info). If you set this environment variable, you do not need to perform any of the steps listed above.

Valid values are (case insensitive): none, all, info, warning, error. Multiple values can be set in a comma separated list. For example:

```
export BT_RLP_LOG_LEVEL=all
export BT_RLP_LOG_LEVEL=none
export BT_RLP_LOG_LEVEL=warning,error
```

Note

Any setting of `BT_RLP_LOG_LEVEL` is overridden by a call to the C++ `SetLogLevel` function or the Java `setLogLevel` method, unless you pass "" as the parameter, in which case RLP uses the `BT_RLP_LOG_LEVEL` setting (if it is set), or "error" (if is not set).

3.6.3. Initializing the Environment

If you are using C++:

1. Create an empty RLP environment.
2. Use the global configuration file to initialize the environment.

```
//Instantiate the environment.
BT_RLP_Environment* rlp = BT_RLP_Environment::Create();
//envConfig, the pathname of rlp-global.xml, was set from a command-line
//argument. If you put the environment configuration xml in a buffer,
//use InitializeFormBuffer.
BT_Result rc; //For function results.
rc = rlp->InitializeFromFile(envConfig);
```

If you are using Java:

1. Define the path to the global configuration file as an environment parameter.
2. Instantiate an environment object with the environment parameters object.
3. Initialize the environment.

```
// envConfig, path to rlp-global.xml, set from command line.
envParams.setEnvironmentDefinition(new File(envConfig));
//Instantiate and initialize the environment.
RLPEnvironment env = new RLPEnvironment(envParams);
env.initialize();
```

3.7. Getting License Information

The RLP license you have obtained from Basis Technology determines the scope of operations you can perform with RLP. If you are unsure of exactly what you have, you can consult your RLP license file: **rlp-license.xml** (normally in the **rlp/rlp/licenses** subdirectory under the Basis root directory).

If you attempt to use a processor or process a language for which you do not hold a license, RLP issues a warning and continues operation.

Warning

Be sure you have purchased the licenses that are required for the processors that you want to use in your applications. Turn on the warning channel in your logging callback to receive such notification.

RLP provides an API for gathering information about your license. The environment object provides methods for determining whether you have a valid license, whether it authorizes base linguistics and named entity extraction for a given language, and whether it authorizes support for a given named feature. This release includes sample applications that demonstrate how to use this API: **samples/cplusplus/examine_license.cpp** and **samples/java/ExamineLicense.java**.

3.7.1. C++: `BT_RLP_Environment` License Methods

To determine whether you have a valid license, use

```
bool HasLicenses() const = 0;
```

To determine whether your license includes base linguistics or named entity extraction for a given language, use

```
bool HasLicenseForLanguage(BT_LanguageID lid, BT_UInt32 functionality)  const = 0;
```

To determine whether your license includes support for a named feature, use

```
virtual bool HasLicenseForNamedFeature(const char *feature,
                                          BT_UInt32 functionality) const = 0;
```

Language IDs are defined in **bt_language_names.h**. The arguments you can use for `functionality` and `feature` are defined in **bt_rlp_license_types.h**.

These methods are used in the `examineLicenses` method of the Sample C++ Application [28] .

3.7.2. Java: `RLPEnvironment` License Methods

To determine whether you have a valid license, use

```
boolean hasLicenses() throws RLPException
```

To determine whether you license includes base linguistics or named entity extraction for a given language, use

```
boolean hasLicenseForLanguage(int language_id, int functionality)
 throws RLPException
```

To determine whether your license includes support for a named feature, use

```
boolean hasLicenseForNamedFeature(String feature, int functionality)
 throws RLPException
```

Language IDs are enumerated in `com.basistech.util.LanguageCode`. The arguments you can use for `functionality` and `feature` are defined in `com.basistech.rlp.RLPConstants`.

These methods are used in the `examineLicenses` method of the Sample Java Application [35] .

3.8. Setting Up the Context

After initializing the environment, use the environment object to instantiate a context. The context can only contain a subset of the processors defined in the environment. A context configuration XML file or string defines the context; see Defining a Context [18] .

3.8.1. Instantiating a Context in C++

If the context configuration is an external XML file, use the `BT_RLP_Environment::GetContextFromFile` method.

If the context configuration is embedded in the application, use the `BT_RLP_Environment::GetContextFromBuffer` method.

The following fragment gets the context configuration from an external file:

```
//Get context configuration XML file pathname from the
//command line. For example:
const char* contextPath = argv[2];
BT_RLP_Context* context;
//rc is BT_Result return code; rlp is BT_RLP_Environment object.
rc = rlp->GetContextFromFile(contextPath, &context);
if (rc != BT_OK) {
  cerr << "Unable to create the context." << endl;
  delete rlp;
  return 1;
}
```

The sample C++ application that appears later in this chapter gets the context from a buffer. See item **3.** in Sample C++ Application [28] .

3.8.2. Instantiating a Context in Java

The `ContextParameters` class provides two versions of the `setContextDefinition` method. One version takes File (the context configuration file) as an argument. The other version takes a String (the context configuration XML).

Warning

If you use an XML String rather than an XML file, and the XML declaration contains an `encoding` attribute (optional), the encoding MUST be set to UTF-8. For example: `<?xml version='1.0' encoding='utf-8'?>`.

Once you have a `ContextParameters` object with a context, you can use the `RLPEnvironment getContext` method to instantiate the context.

The following fragment gets the context from a String within the application:

```
ContextParameters contextParam = new ContextParameters();
//contextSpec is a String with the context configuration XML.
contextParam.setContextDefinition(contextSpec);
RLPContext context = rlpEnv.getContext(contextParam);
```

The sample Java application that appears later in this chapter gets the context from a file. See item **3.** in Sample Java Application [34] .

3.8.3. Setting Context Properties

Many RLP processors have properties that you can set. A context property may be global rather than processor-specific [125] . You can set properties in the context configuration. You can also use the context object to set properties (overriding settings in the configuration). A context configuration may include property settings

In C++, use the `BT_RLP_Context::setPropertyValue` method to set context properties.

In Java, use the `RLPContext.setProperty` method to set context properties.

Both methods take two String parameters: property name and property value. For information about context properties, see RLP Processors [123] .

3.9. Processing Input

Once you have set up an environment and used it to instantiate a context, you can use the context object to process the input text. The context method you use depends on whether or not the input is a file. If the input is not a file, the context method you use depends on encoding.

For detailed information about the C++ API, see `ProcessFile`, and `ProcessBuffer` in BT_RLP_Context [api-reference/cpp-reference/classBT__RLP__Context.html].

For the details of the Java API, see the `process` methods in RLPContext [api-reference/java-reference/com/basistech/rlp/RLPContext.html].

3.9.1. Input Is a File

In C++, use the `BT_RLP_Context::ProcessFile` method. For an example, see item **4.** in the Sample C++ Application [28] .

In Java, use one of the `RLPContext.process` methods that takes a `String pathname` parameter. For an example, see item **4.** in the Sample Java Application [34] .

These methods post raw input data to internal storage. Unless you are using `iFilter` to process the file, the context must include `RLI`/`RCLU` or `Unicode Converter` to generate the UTF-16 raw text required by other language processors.

iFilter requires a File

If your context includes `iFilter`, the input must be a file.

3.9.2. Input Is Not a File

If the text input is not a file, determine whether the encoding matches what the RLP processors require on the current platform.

3.9.2.1. Encoding Does Not Match RLP Requirements

If the encoding is not UTF-16 with the correct byte order for the platform, you must call a process method that works with a buffer that can contain text in any encoding.

In C++, use the `BT_RLP_Context::ProcessBuffer` method.

In Java, use one of the `RLPContext process` methods that includes a `byte[]`, `ByteBuffer`, or `char[]` parameter for the input data.

These methods post raw input data in RLP internal storage. The context must include `RLI` and `RCLU`, or `Unicode Converter` to generate the UTF-16 raw text required by other language processors.

3.9.2.2. Input is UTF-16 with Platform Byte Order

If the text input is UTF-16 with the correct byte order for your platform, you can use one of the API methods described in the previous section, or you can call a more efficient method that generates the UTF-16 raw text required for processing. If the input includes a byte order mark (BOM), you may want to strip the BOM (the first character) before you process the input text. See Handling the BOM [78] .

In C++, use the `BT_RLP_Context::ProcessBufferUTF16` method. For an example, see RLBL C ++ Fragment [75]

In Java, use one of the `RLPContext.process` methods that takes a `String data` parameter.

For an example, see RLBL Java Fragment [76] .

The methods described in this section generate the UTF-16 raw text required for processing. Your context does not need `RCLU` or `Unicode Converter`; if it includes either of those processors, they are ignored.

3.10. Introduction to Our Sample Applications

The sample applications that follow are designed to provide you with a shell or template that you can use as a starting point for creating your own applications. [3] Look in the `samples` tree for the source files and the tools for building and running these samples.

The code comments indicate how each application implements the steps listed above. The fundamental changes you must make are the following:

- Adjust the input parameters.

  For runtime flexibility, you will probably use input parameters to define most of the following: Basis root directory, language, encoding, context configuration, input text. [4]

- Set up the RLP log callback to provide the information you need to understand any problems that may arise as you are running RLP.

- Define a context that generates the results you need.

  You can embed your context configuration in the application. For flexibility and wider use, you may want to maintain a separate context configuration file.

[3]For a sample C application, see Using the RLP C API [59] .
[4]Use the ISO639 language code [11] to designate the language.

- Use the context to process the input text.
- Handle the RLP results as your needs dictate.

  This is the heart of your application. The samples illustrate the basic procedures for accessing the different kinds of result objects that processing text input can generate.

For information about building and running these samples and the other samples included with the RLP distribution, see Building and Running the Sample C++ Applications [40] and Building and Running the Sample Java Applications [41] .

3.11. Sample C++ Application

C++ source file: **rlp_sample.cpp**. Input parameters are passed in from the command line, and the RLP context configuration document is embedded in the source code.

```
// Include the RLP interfaces
#include <bt_rlp.h>
#include <bt_language_names.h>
#include <bt_rlp_ne_iterator.h>

// Basis Tech string class, for UTF16 -> UTF-8 handling
#include <bt_xstring.h>

// C++ includes
#include <iostream>
#include <fstream>
#include <cstring>
#include <vector>

using namespace std;

//prototypes
static void handleResults(BT_RLP_Context* context, ofstream& out);
static void log_callback(void* info_p, int channel, const char* message);

//Use an XML string or file to define a context. The context is an
//ordered sequence of processors and related settings that specify
//the processing that RLP performs on the input. This app defines
//the context in the string below.

//This is a general-purpose context configuration.
// - The Unicode Converter converts the input file from UTF-8 (or any other
//   Unicode encoding) to UTF-16 for internal processing.
// - Depending on the language of the input, the appropriate language analyzer
//   (BL1, JLA, CLA, KLA, or ARBL) performs a variety of tasks appropriate for
//   that language. For example, JLA tokenizes Japanese text, assigns part of
//   speech tags, and determines alphabetic (Hiragana) readings for native
//   Japanese words in Kanji.
// - BaseNounPhrase detects noun phrases.
// - SentenceBoundaryDetector delimits the sentences in the input.
// - Stopwords uses the language-specific stopwords dictionary to tag tokens
//   as stopwords.
// - NamedEntityExtractor, Gazeteer, RegExpLP, and NamedEntityRedactor locate
//   named entities.
static const char* CONTEXT =
  "<?xml version='1.0'?>"
  "<contextconfig>"
```

```
 "<languageprocessors>"
 "<languageprocessor>Unicode Converter</languageprocessor>"
 "<languageprocessor>BL1</languageprocessor>"
 "<languageprocessor>JLA</languageprocessor>"
 "<languageprocessor>CLA</languageprocessor>"
 "<languageprocessor>KLA</languageprocessor>"
 "<languageprocessor>Tokenizer</languageprocessor>"
 "<languageprocessor>SentenceBoundaryDetector</languageprocessor>"
 "<languageprocessor>ARBL</languageprocessor>"
 "<languageprocessor>FABL</languageprocessor>"
 "<languageprocessor>URBL</languageprocessor>"
 "<languageprocessor>Stopwords</languageprocessor>"
 "<languageprocessor>BaseNounPhrase</languageprocessor>"
 "<languageprocessor>NamedEntityExtractor</languageprocessor>"
 "<languageprocessor>Gazetteer</languageprocessor>"
 "<languageprocessor>RegExpLP</languageprocessor>"
 "<languageprocessor>NERedactLP</languageprocessor>"
 "</languageprocessors>"
 "</contextconfig>";

/**
* 1. Process input parameters.
* 2. Set up RLP environment.
* 3. Set up RLP context for processing input.
* 4. Process input.
* 5. Work with the results.
* 6. Clean up.
*/
int main(int argc, const char* argv[])
{
 if (!BT_RLP_Library::VersionIsCompatible()) {
   fprintf(stderr, "RLP library mismatch: have %s expect %s\n",
     BT_RLP_Library::VersionString(),
     BT_RLP_LIBRARY6.5.2STRING);
   return 1;
 }

 //1. Process input parameters. This application gets the following
 //   from the command line:
 //      - Basis root directory
 //      - language ISO639 code
 //      - pathname of the environment config file
 //      - pathname of the input file
 //      - pathname of the output file (encoded in utf-8)
 if (argc != 6) {
   cerr << "Usage: " << argv[0]
   << " BT_ROOT LANGUAGE ENV_CONFIG_FILE INPUT_FILE OUTPUT_FILE" << endl;
   return 1;
 }
 const char* btRoot   = argv[1];
 // Get BT language ID (defined in bt_language_names.h)
 // from ISO639 code.
 const BT_LanguageID langID = BT_LanguageIDFromISO639(argv[2]) ;
 if (langID ==BT_LANGUAGE_UNKNOWN) {
   cerr << "Warning: Unknown ISO639 language code: " << argv[2] << endl;
   return 1;
 }
 const char* envConfig = argv[3];
 const char* inputFile = argv[4];
const char* outputFile = argv[5];
ofstream out(outputFile);
if (!out) {
  cerr << "Couldn't open output file: " << outputFile << endl;
  return 1;
}

//2. Set up the environment.

//2.1 Use BT_RLP_Environment static methods to set the Basis root
//    directory, to designate a log callback function, and to set
//    log level.
BT_RLP_Environment::SetBTRootDirectory(btRoot);
BT_RLP_Environment::SetLogCallbackFunction((void*) stderr,
  log_callback);
//Log level is some combination of "warning,error,info" or "all".
BT_RLP_Environment::SetLogLevel("error");

//2.2 Create a new (empty) RLP environment.
BT_RLP_Environment* rlp = BT_RLP_Environment::Create();
if (rlp == 0) {
  cerr << "Unable to create the RLP environment." << endl;
  return 1;
}
//2.3 Initialize the empty environment with the global environment
//    configuration file.
BT_Result rc = rlp->InitializeFromFile(envConfig);
if (rc != BT_OK) {
  cerr << "Unable to initialize the environment." << endl;
  delete rlp;
  return 1;
}

//3. Get a context from the environment. In this case the context
//   configuration is embedded in the app as a string. It could also
//   be read in from a file.
BT_RLP_Context* context;
rc = rlp->GetContextFromBuffer((const unsigned char*) CONTEXT,
  strlen(CONTEXT),
  &context);
if (rc != BT_OK) {
  cerr << "Unable to create the context." << endl;
  delete rlp;
  return 1;
}

//4. Use the context object to processes the input file. Must include
//   language id unless using RLI processor to determine language.
rc = context->ProcessFile(inputFile, langID);
if (rc != BT_OK) {
  cerr << "Unable to process the input file '"
    << inputFile << "'." << endl;
  rlp->DestroyContext(context);
  delete rlp;
  return 1;
}
//5. Gather results of interest produced by processing the input text.
handleResults(context, out);
fprintf(stdout, "\nSee output file: %s\n\n", outputFile);
```

```
  //6. Remove any objects still lying around.
  rlp->DestroyContext(context);
  delete rlp;
  return 0;
}

//5. Get results of interest and write to file.
static void handleResults(BT_RLP_Context* context, ofstream& out) {
  try {
    //5.1 Use the context object to get single-valued results: language,
    //    encoding, raw text, transcribed text (Arabic).
    BT_UInt32 lang = context->GetIntegerResult(BT_RLP_DETECTED_LANGUAGE);
    const char* langName = BT_ISO639FromLanguageID(lang);
    const BT_Char8* encoding =
      context->GetStringResult(BT_RLP_DETECTED_ENCODING);
    out << "Language: " << langName << "(" << lang << ")\t";
    if (encoding != 0)
      out << "Encoding: " << encoding << endl;
    BT_UInt32 len = 0;
    const BT_Char16* rawText =
      context->GetUTF16StringResult(BT_RLP_RAW_TEXT, len);
    bt_xstring rawText_as_utf8(rawText, len);
    out << "Raw text: " << rawText_as_utf8.c_str() << endl << endl;

    //5.2 Use token iterator to get information related to tokens: tokens,
    //    token offsets, part of speech tags, stems, normalized tokens,
    //    stopwords, compounds, and readings.
    BT_RLP_TokenIteratorFactory* factory =
      BT_RLP_TokenIteratorFactory::Create();

    //Provide access to readings and compounds.
    factory->SetReturnReadings(true);
    factory->SetReturnCompoundComponents(true);

    //Create the iterator and destroy the factory.
    BT_RLP_TokenIterator* token_iter = factory->CreateIterator(context);
    factory->Destroy();

    vector<bt_xwstring> tokens;
    while (token_iter->Next()) {
      //Get the data you want for each token.
      //Get the token (BT_RLP_TOKEN).
      const BT_Char16* token = token_iter->GetToken();
      tokens.push_back(bt_xwstring(token));

      //Get the token index (not currently used).
      //BT_UInt32 index = token_iter->GetIndex();

      //Get the part of speech (BT_RLP_PART_OF_SPEECH) for the token.
      const char* pos = token_iter->GetPartOfSpeech();
      //and so on ...

      //Convert the token from UTF-16 to UTF-8 and dump it.
      bt_xstring token_as_utf8(token);
      out << "Token: " << token_as_utf8.c_str() << endl;

      //The POS tag is stored as a null-terminated ASCII string,
      //so it is can be writen out directly.
  if(pos){
    out << "  POS: " << pos << endl;
  }

  //Get readings (for Japanese, render Kanji characters in
  // the Hiragana alphabet).
  int numReadings = token_iter->GetNumberOfReadings();
  if (numReadings > 0) {
    out << "  Readings: ";
    for (int i = 0; i < numReadings; ++i) {
      const BT_Char16* reading = token_iter->GetReading(i);
      //Dump as UTF-8.
      bt_xstring token_as_utf8(reading);
      out << token_as_utf8.c_str() << " ";
    }
    out << endl;
  }

  //Get compounds (see documentation for applicable languages)
  int numCompoundComponents = token_iter->GetNumberOfCompoundComponents();
  if (numCompoundComponents > 0) {
    out << "  Compound components: ";
    for (int i = 0; i < numCompoundComponents; ++i) {
      const BT_Char16* reading = token_iter->GetCompoundComponent(i);
      //Dump as UTF-8.
      bt_xstring token_as_utf8(reading);
      out << token_as_utf8.c_str() << " ";
    }
    out << endl;
  }
  out << endl;
}

//5.3 Use result iterator to get other results, such as base noun phrases,
//    and gazetteer names. Note: Can use result iterator to get any/all
//    results.

//Get base noun phrases.
BT_RLP_ResultIterator* result_iter =
  context->GetResultIterator(BT_RLP_BASE_NOUN_PHRASE);

const BT_RLP_Result* bnp;
while ((bnp = result_iter->Next()) != NULL){
  //The result is a pair of integers, indexes of the first and last + 1
  //tokens in the base noun phrase.
  BT_UInt32 first, last;
  bnp->AsIntegerPair(first, last);
  out << "Base Noun Phrase: ";
  // Use the tokens vector to construct the base noun phrases.
  // Convert tokens from UTF-16 to UTF-8.
  for (unsigned j = first; j < last; ++j) {
    bt_xstring u8token(tokens[j].c_str());
    out << u8token.c_str() << " ";
  }
  out << endl;
}

//5.4 Use named entity iterator to get information about named entities.
```

```
    BT_RLP_NE_Iterator_Factory *nif = BT_RLP_NE_Iterator_Factory::Create();
    BT_RLP_NE_Iterator *ni = nif->CreateIterator(context);
    //Ensure that multiple instances of a named entity are returned
    //with the same entity type.
    nif->SetConsistentType(true);
    nif->Destroy();
    while (ni->Next()) {
      const BT_Char16 *neWide = ni->GetRawNamedEntity();
      bt_xstring ne_as_utf8(neWide);
      BT_UInt32 neType = ni->GetType();
      //Get string representation of named entity type.
      const char* typeName = BT_RLP_NET_ID_TO_STRING(neType);
      out <<"Named Entity: " <<typeName <<" " <<ne_as_utf8.c_str() << endl;
    }
    ni->Destroy();

    //Cleanup
    context->DestroyResultIterator(result_iter);
    token_iter->Destroy();
  }
  catch (BT_RLP_InvalidResultRequest& e) {
    cerr << "Exception: " << e.what() << endl;
  }
  catch (...) {
    cerr << "Unhandled exception." << endl;
  }
}

/**
* The application registers this function to receive diagnostic log entries.
* RLP Environment LogLevel determines which message channels (error, warning.
* info) are posted to the callback.
*/
static void log_callback(void* info_p, int channel, const char* message)
{
  static const char* szINFO     = "INFO  : ";
  static const char* szERROR    = "ERROR : ";
  static const char* szWARN     = "WARN  : ";
  static const char* szUNKNOWN  = "UNKWN : ";
  const char* szLevel = szUNKNOWN;
  switch(channel) {
    case 0:
      szLevel = szWARN;
      break;
    case 1:
      szLevel = szERROR;
      break;
    case 2:
      szLevel = szINFO;
      break;
  }
  fprintf((FILE*) info_p, "%s%s\n", szLevel, message);
}


/*
Local Variables:
mode: c
tab-width: 2
```

```
c-basic-offset: 2
End:
*/
```

3.12. Sample Java Application

`RLPSample` reads in its input parameters from a properties file. The context configuration document is also a separate file.

Input parameters: **RLPSample.properties**.

```
# RLPSample.properties
# input values for RLPSample

# Sample code replaces $BT_ROOT with the command-line arg
# for the Basis Technology root directory.

env = $BT_ROOT/rlp/etc/rlp-global.xml
context = $BT_ROOT/rlp/samples/etc/rlp-context-no-op.xml
input = $BT_ROOT/rlp/samples/data/de-text.txt

#ISO639 language code
lang = de

mime_charset = UTF-8

# log_level can be a comma-delimited list from the following:
# 'warning,error,info' -- or it can be 'all' or 'none'
log_level = error

# output destination
out = RLPSample-out.txt
```

Context configuration: **rlp-context-no-op.xml**.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE contextconfig SYSTEM "http://www.basistech.com/dtds/2003/contextconfig.dtd">
<!-- This context is useful with applications that do not need REXML
      because they produce their own output. -->
<contextconfig>
  <languageprocessors>
    <languageprocessor>Unicode Converter</languageprocessor>
    <languageprocessor>BL1</languageprocessor>
    <languageprocessor>JLA</languageprocessor>
    <languageprocessor>CLA</languageprocessor>
    <languageprocessor>KLA</languageprocessor>
    <languageprocessor>Tokenizer</languageprocessor>
    <languageprocessor>SentenceBoundaryDetector</languageprocessor>
   <languageprocessor>ARBL</languageprocessor>
   <languageprocessor>FABL</languageprocessor>
   <languageprocessor>URBL</languageprocessor>
    <languageprocessor>Stopwords</languageprocessor>
    <languageprocessor>BaseNounPhrase</languageprocessor>
    <languageprocessor>NamedEntityExtractor</languageprocessor>
    <languageprocessor>Gazetteer</languageprocessor>
    <languageprocessor>RegExpLP</languageprocessor>
    <languageprocessor>NERedactLP</languageprocessor>
  </languageprocessors>
</contextconfig>
```

Java source: **RLPSample.java**.

```
 import com.basistech.rlp.RLPConstants;
import com.basistech.util.Pathnames;
import com.basistech.util.LanguageCode;
import com.basistech.rlp.EnvironmentParameters;
import com.basistech.rlp.RLPEnvironment;
import com.basistech.rlp.ContextParameters;
import com.basistech.rlp.RLPContext;
import com.basistech.rlp.RLPResultAccess;
import com.basistech.rlp.NamedEntityData;

import java.text.MessageFormat;
import java.text.ChoiceFormat;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.FileOutputStream;
import java.io.PrintStream;
import java.util.List;
import java.util.Iterator;
import java.util.Map;
import java.util.Properties;
import java.util.Enumeration;

/* RLPSample ilustrates the basic pattern for using RLP to process an
 * input file and to work with the results that are generated.
 *
 * 1. Load parameters required to run the application.
 * 2. Set up the RLP environment.
 * 3. Set up an RLP context.
 * 4. Use the context to process the input text.
 * 5. Capture results of interest generated during the previous step.
 * 6. Close the context and environment objects.
*/

public class RLPSample {
  /**
   * Instantiates and runs RLPSample.
   */
  public static void main(String args[]) {
    if (args.length < 2) {
      System.err.println("Usage: must call with 2 args: " +
        "the BT_ROOT directory and the properties file pathname");
      System.exit(1);
    }

    new RLPSample().run(args[0], args[1]);
  }

  /**
   * Performs the steps listed above.
   * @param btRoot the BT_ROOT directory (the install directory)
   * @param rlpProps the path to the properties file
   */
  void run(String btRoot, String rlpProps) {
    try {
      //1. Load input parameters.
```

```
//   App gets the Basis root directory and a
//   properties file from the command line.
//   Other parameters are loaded from the
//   properties file:
//    * env - environment configuration file
//    * context - context configuration file
//    * input - text input file
//    * lang - ISO639 language code
//    * mime_charset - charset encoding of input
//    * log_level - log level
//    * out - output file
Properties props = loadProperties(btRoot, rlpProps);

//2. Set up the RLP environment.

//2.1 Set the Basis root directory and create an
//    EnvironmentParameters object.
Pathnames.setBTRootDirectory(btRoot);
EnvironmentParameters envParams = new EnvironmentParameters();

//2.2 Set up logging.
//See LogCallback inner class below.
envParams.setLogCallback(new LogCallback());
//Log level is some combination of "warning,error,info"
//or "all" or "none".
envParams.setLogLevel(props.getProperty("log_level"));

//2.3 Instantiate and initialize the environment.
//Use the environment configuration file to define
//the environment.
String environmentPath = props.getProperty("env");
envParams.setEnvironmentDefinition(new File(environmentPath));
RLPEnvironment rlpEnv = new RLPEnvironment(envParams);
rlpEnv.initialize();

//3. Set up the context.
ContextParameters contextParam = new ContextParameters();
String contextPath = props.getProperty("context");
contextParam.setContextDefinition(new File(contextPath));

RLPContext rlpContext = rlpEnv.getContext(contextParam);

//If not using RLI to determine language, must specify
//the language of the input file.
LanguageCode language = LanguageCode.UNKNOWN;
if (props.getProperty("lang") != null) {
  language = LanguageCode.lookupByISO639(props.getProperty("lang"));
}

//4. Process the input file, including the charset of the file and
//   language id.
String input = props.getProperty("input");
String mime_charset = props.getProperty("mime_charset");
rlpContext.process(input, mime_charset, language);

//5. Handle the results generated by the processors.
String outFile = props.getProperty("out");
handleResults(rlpContext, outFile);
```

```
    // 6. Close context and environment.
    rlpContext.close();
    rlpEnv.close();
  }
  catch (Exception exc) {
    exc.printStackTrace();
    System.exit(1);
  }
}

/**
 * Loads properties from RLPSample.properties: environment XML
 * file, context XML file, input file, language, input file charset,
 * log callback, log file, output file.
 */
Properties loadProperties(String btRoot, String rlpProps)
  throws FileNotFoundException, IOException {

  Properties props = new Properties();
  InputStream in = new FileInputStream(rlpProps);
  props.load(in);
  in.close();
  // Need to change '/' in paths if non-UNIX platform.
  // Paths are relative to the BT_ROOT directory, so need to replace
  // "$BT_ROOT" in property values with btRoot.
  char sep = File.separatorChar;
  Enumeration propsEnum = props.propertyNames();
  while (propsEnum.hasMoreElements()) {
    String key = (String)propsEnum.nextElement();
    String value = props.getProperty(key);
    String newValue = value;
    if (value.startsWith("$BT_ROOT"))
      newValue = btRoot + value.substring("$BT_ROOT".length());
    if (sep != '/' && newValue.indexOf('/') > -1)
      newValue=newValue.replace('/', sep);
    if (!newValue.equals(value))
      props.setProperty(key, newValue);
  }
  return props;
}

/**
 * Instructs RLP to send message to designated output (standard out or a
 * file). Can log warnings, errors, info messages.  RLPEnvironment
 * LogLevel determines which messages are posted to the callback.
 */
class LogCallback implements RLPEnvironment.LogCallback {
  public void message(int channel, String message) {
    MessageFormat form = new MessageFormat("{0} {1}");
    ChoiceFormat numFormat =
      new ChoiceFormat("0#WARN:|1#ERROR:|2#INFO:");
    form.setFormatByArgumentIndex(0, numFormat);
    System.err.print(form.format(new Object[]
      {new Integer(channel), message}));
  }
}

/**
 * Assembles some of the result data and reports to the user:
```

```
 *   detected language
 *   detected encoding
 *   tokens
 *   noun phrases
 *   named entities
 *   stem and part of speech for each token
 *   compounds (if input is German or Dutch)
 *   sentence boundaries
 *
 * @param context RLP context responsible for processing the input
 * @param outFile report file name
 */
void handleResults(RLPContext rlpContext, String outFile) {
  try {
    // Target for result data.
    final PrintStream out;
    // Write file with UTF-8 encoding.
    FileOutputStream fos = new FileOutputStream(outFile);
    out = new PrintStream(fos, false, "UTF-8");
    // Inform the user.
    System.out.println("See results in " + outFile);

    //Instantiate a result access object. It can be used to get all
    //results.
    RLPResultAccess resultAccess = new RLPResultAccess(rlpContext);

    //Get language and encoding (singleton values).
    LanguageCode langCode = resultAccess.getDetectedLanguage();
    //Get the ISO639 language code and the language name from the LanguageCode.
    if (langCode != null) {
      String iso639Code = langCode.ISO639();
      String encoding =
        resultAccess.getStringResult(RLPConstants.DETECTED_ENCODING);
      out.println("ISO639 language code: " + iso639Code + "; Encoding: " + encoding);
    }
    //Get access to the tokens.
    List<Object> tokenList = resultAccess.getListResult(RLPConstants.TOKEN);

    //Get base noun phrases.
    List<Object> bnpList =
      resultAccess.getListResult(RLPConstants.BASE_NOUN_PHRASE);
    Iterator<Object> iter;
    if (bnpList != null && tokenList != null) {
      //Use tokens to assemble each noun phrase.
      iter = bnpList.iterator();
      while (iter.hasNext()) {
        //Each element is int[2]: start and end+1 token indexes for
        //the noun phrase.
        int[] startEndIndexes = (int[])iter.next();
        out.print("Noun phrase:");
        for (int i = startEndIndexes[0]; i < startEndIndexes[1]; i++) {
          out.print(" " + (String)tokenList.get(i));
        }
        out.println();
      }
    }
    //Get named entity data.
    //Ensure that multiple instances of a named entity are returned
    //with the same entity type.
```

```
resultAccess.setConsistentType(true);
NamedEntityData[] neData = resultAccess.getNamedEntityData(true);
for (int i = 0; i < neData.length; i++){
  String normalizedNE = neData[i].getNormalizedNamedEntity();
  String typeName   = neData[i].toString();
  out.println("Normalized Named entity (" + typeName + "): "
      + normalizedNE);
}
//Get sentence boundaries.
//Each Integer result is the index of the token after
//the end of the sentence, which is also the first token in
//the next sentence.
List<Object> sbList =
  resultAccess.getListResult(RLPConstants.SENTENCE_BOUNDARY);
if (sbList != null && tokenList != null) {
  iter = sbList.iterator();
  int start = 0; //for first sentence
  while (iter.hasNext()) {
    int end = ((Integer)iter.next()).intValue();
    out.print("SENTENCE:");
    // Assemble the sentence.
    for (int i = start; i < end; i++) {
      out.print(" " + (String)tokenList.get(i));
    }
    out.println();
    start = end; //for next sentence
  }
}
//Get stem and part of speech for each token.
List<Object> stemList = resultAccess.getListResult(RLPConstants.STEM);
List<Object> posList = resultAccess.getListResult(RLPConstants.PART_OF_SPEECH);
if (stemList != null && posList != null) {
  iter = posList.iterator();
  //Walk through the token list, stem list and pos list (then nth stem
  // and nth pos tag refer to the nth token).
  for (int i = 0; i < tokenList.size(); i++) {
    out.println("Token Stem POS: " + (String)tokenList.get(i) +
      " " + (String)stemList.get(i) + " " + (String)posList.get(i));
  }
}
//Get compound words if available (German, Dutch, or Japanese).
//Note: Also use getMapResult for readings and token variations.
Map<Integer, String[]> map = resultAccess.getMapResult(RLPConstants.COMPOUND);
if (map != null) {
  Iterator<Integer> iter2 = map.keySet().iterator();
  while (iter2.hasNext())
  {
    //Key is the token index.
    Integer key = iter2.next();
    out.print("Compound: " +
      (String)tokenList.get(key.intValue())+ ":");
    //Value is the compound elements that make up the token.
    String[] value = map.get(key);
    for (int i = 0 ; i < value.length; i++) {
      out.print(" " + value[i]);
    }
    out.println();
  }
}
```

```
      // Output complete.
      out.close();
    }
    catch (Exception e) {
      e.printStackTrace();
      System.exit(1);
    }
  }

}
```

3.13. Building and Running the Applications

The RLP distribution includes the source files and binary builds for three C++ sample applications and three Java sample applications, including the samples described in this chapter.
The distribution also includes scripts for building and running these sample applications. You do not need to build these samples to run them, but you may want to make your own modifications to the sources and rebuild, and you can use these scripts as the starting point for setting up a build environment for your own applications.

3.13.1. Building and Running the Sample C++ Applications

RLP ships with three sample C++ applications:

- `rlp_sample`

  Illustrates the basic pattern for processing text. See Sample C++ Application [28] .

- `rlbl_sample`

  Illustrates the pattern for processing multilingual text [73] .

- `examine_license`

  Illustrates the API for gathering information about the scope of your RLP license.

The source files for these samples are in ***BT_ROOT*/rlp/samples/cplusplus**.

3.13.1.1. Building the C++ Sample Programs

RLP supplies a script for building the C++ sample programs. On Windows platforms, the script is ***BT_ROOT*/rlp/samples/cplusplus/build.bat**. On Unix platforms, the script is ***BT_ROOT*/rlp/samples/cplusplus/build.sh**.

Important

The compiler and linker for your platform must be on your path.

If you are running Windows 32, you can run **vsvars32.bat** from the **Common7\Tools** directory in your Visual Studio installation. If you are running Windows 64, run **vcvarsamd.bat** from the **VC\bin\amd64\vcvarsamd64.bat** directory in your Visual Studio installation

If you are running Unix, be sure you have set the PATH environment variable to include the compiler and linker for your platform.

Before you call the script, you must set the *BT_BUILD* environment variable. You must also run the script from the ***BT_ROOT*/rlp/samples/cplusplus** directory.

Windows example:

```
set BT_BUILD=ia32-w32-msvc80
cd \btroot\rlp\samples\cplusplus
build.bat
```

Unix example:

```
export BT_BUILD=ia32-glibc22-gcc32
cd ~/btroot/rlp/samples/cplusplus
./build.sh
```

The build script calls another script in the same directory with *BT_BUILD* embedded in the script name: **build_*BT_BUILD*_cpp_samples.[sh|bat]**. You can examine this script for the command that is used to compile and link each of the C++ samples.

An executable for each sample program with the same name as the C++ source file is placed in ***BT_ROOT*/rlp/bin/*BT_BUILD*.**

3.13.1.2. Running the C++ Samples

The RLP distribution includes a script for running the C++ samples: **go-cpp-samples.bat** (Windows) or **go-cpp-samples.sh** (Unix). This script is in ***BT_ROOT*/rlp/samples/scripts/*BT_BUILD*** . The script runs all the samples. You can use it as a model for creating your own command-line scripts.

On Unix

On Unix platforms, you must set `LD_LIBRARY_PATH` (or its equivalent environment variable for your Unix operating system) to include the RLP library directory: ***BT_ROOT*/rlp/lib/*BT_BUILD*** .

3.13.2. Building and Running the Sample Java Applications

To compile a Java RLP application, you must place **btrlp.jar** and **btutil.jar** on the classpath. [5] These jars are located in ***BT_ROOT*/rlp/lib/*BT_BUILD*** , where *BT_ROOT* is the installation directory, and *BT_BUILD* is based on the operating system and C++ compiler for your platform. See Supported Platforms [13] in *Installing RLP*.

To run your application, you must place the application, **btrlp.jar**, and **btutil.jar** on the classpath. On Linux, you must set `LD_LIBRARY_PATH` (or its equivalent environment variable for your Unix operating system) to include the RLP library directory: ***BT_ROOT*/rlp/lib/*BT_BUILD*** .

RLP ships with three sample Java applications:

- `RLPSample`

  Illustrates the basic pattern for processing text. See Sample Java Application [34] .

- `MultiLangRLP`

[5] **btutil.jar** contains a property file (`com.basistech.util.build.properties`) that defines the relative path from *BT_ROOT* to `java.library.path`. This path includes *BT_BUILD*, so **btutil.jar** can only be used on the platform on which it was originally installed. If you want to port the JAR to a different platform, you must replace or override `com.basistech.util.build.properties`.

Illustrates the pattern for processing multilingual text [73] .

- `ExamineLicense`

  Illustrates the API for gathering information about the scope of your RLP license.

The source files for these samples are in ***BT_ROOT*/rlp/samples/java**. This directory also includes an Ant script that you can use to build and to run these samples. The script requires Ant (1.6.5 or later) with the `JAVA_HOME` environment variable set to the root of your Java SDK (1.5 or later). You should also set the `ANT_HOME` environment variable to point to the root of the Ant installation and put the `ANT_HOME/bin` directory on your PATH. For more information, see Ant. [http://ant.apache.org/]

3.13.2.1. Using the Ant Script

The Ant script requires one input property: `bt.arch` with the value of `BT_BUILD` (`ia32-w32-msvc71`, for example). If you set this property in the script (**build.xml**), you will not need to include it on the command line.

Go to ***BT_ROOT*/rlp/samples/java** and run Ant:

```
ant -Dbt.arch=BT_BUILD target
```

where *`target`* is one of the Ant build targets as described in the following table.

`target`	**Description**
`[NONE]`	(Default) Build all samples. Samples are compiled and placed in ***BT_ROOT*/rlp/samples/java/build/*BT_BUILD*/rlpsamples.jar**. For all builds, the class files are purged after the jar is created.
`clean`	Remove build files.
`build.RLPSample`	Build RLPSample and put the class files and properties file in **rlpsamples.jar**.
`build.MultiLangRLP`	Build MultiLangRLP and put the class files and properties file in **rlpsamples.jar**.
`build.ExamineLicense`	Build ExamineLicense and put the class file in **rlpsamples.jar**.
`RLPSample`	Run RLPSample: include *BT_ROOT* and the path to a properties file with other arguments (`RLPSample.properties`) as command-line arguments.
`MultiLangRLP`	Run MultiLangRLP: include *BT_ROOT* and the path to a properties file with other arguments (`MultiLangRLP.properties`) as command-line arguments.
`ExamineLicense`	Run ExamineLicense: include *BT_ROOT* and the path to the RLP environment configuration file as command-line arguments.

As you create your own applications, you can use this Ant script as the starting point for establishing your own build procedures.

3.13.2.2. Running the Java Samples from the Command Line

The RLP distribution includes a script for running the Java samples: **go-java-samples.bat** (Windows) or **go-java-samples.sh** (Unix). This script is in ***BT_ROOT*/rlp/samples/scripts/*BT_BUILD*** . To run the script, you must set the `JAVA_HOME` environment variable to the root of your Java SDK (1.5 or later). The script runs both samples. You can use it as a model for creating your own command-line scripts.

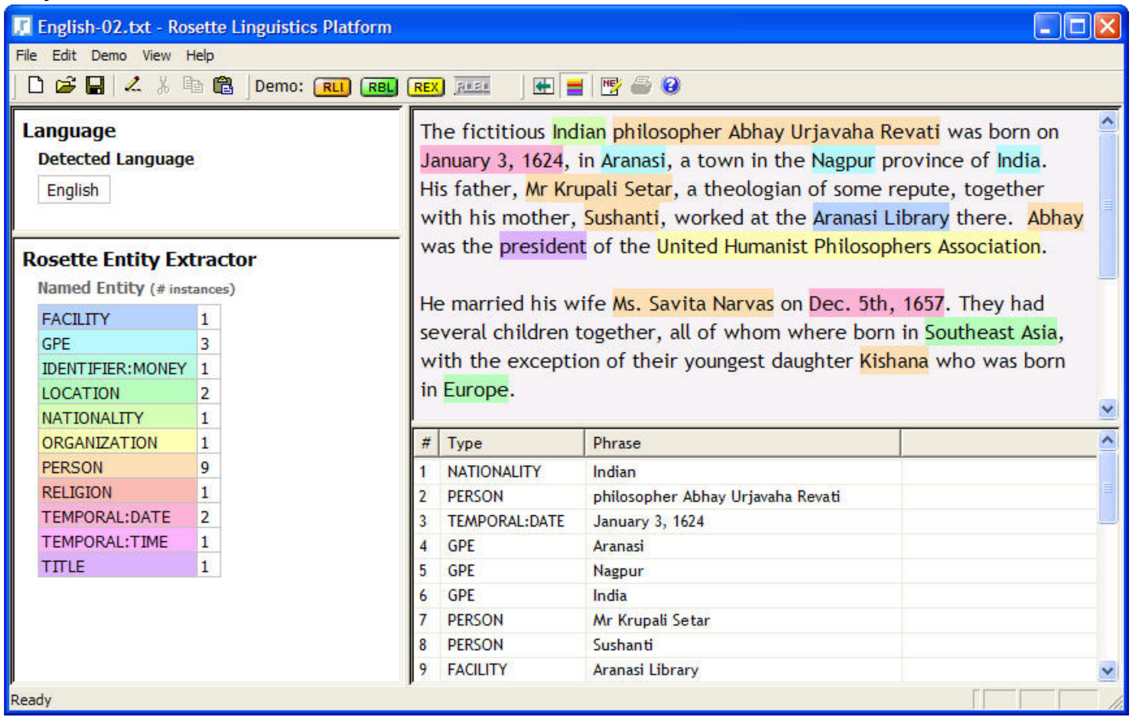
Chapter 4. Working with Named Entities

4.1. Introduction

A named entity refers to an object of potential interest, such as a person, organization, location, or date. When you process a document, locating named entities can help you classify the document and determine what kinds of data of interest it is likely to contain.

As shipped, RLP can identify a variety of entity types in a number of languages. You can also extend RLP coverage to include more entities, additional entity types, and additional languages.

For example, the Rosette Demo [257] finds the named entities highlighted and listed below in a news story :



Language

Detected Language

English

Rosette Entity Extractor

Named Entity (# instances)

FACILITY	1
GPE	3
IDENTIFIER:MONEY	1
LOCATION	2
NATIONALITY	1
ORGANIZATION	1
PERSON	9
RELIGION	1
TEMPORAL:DATE	2
TEMPORAL:TIME	1
TITLE	1

The fictitious Indian philosopher Abhay Urjavaha Revati was born on January 3, 1624, in Aranasi, a town in the Nagpur province of India. His father, Mr Krupali Setar, a theologian of some repute, together with his mother, Sushanti, worked at the Aranasi Library there. Abhay was the president of the United Humanist Philosophers Association.

He married his wife Ms. Savita Narvas on Dec. 5th, 1657. They had several children together, all of whom where born in Southeast Asia, with the exception of their youngest daughter Kishana who was born in Europe.

#	Type	Phrase
1	NATIONALITY	Indian
2	PERSON	philosopher Abhay Urjavaha Revati
3	TEMPORAL:DATE	January 3, 1624
4	GPE	Aranasi
5	GPE	Nagpur
6	GPE	India
7	PERSON	Mr Krupali Setar
8	PERSON	Sushanti
9	FACILITY	Aranasi Library

4.2. Identifying Named Entities

RLP provides three mechanisms for finding named entities and a redactor for resolving differences:

- **Statistical Analysis.** Using contextual features specified by a computational linguist and a substantial body of text in which named entities have been tagged by native speakers, Basis Technology has developed statistical models for a variety of named entity types in a number of languages. For more information, download Basis Technology's Entity Extraction Whitepaper: Entity Extraction Enables "Discovery" [http://www.basistech.com/entity-extraction/entity-extraction-whitepaper.html].

  Entity types include location, organization, person, geo-political entity, facility, religion, nationality, and title. Supported languages include Arabic, Simplified and Traditional Chinese, Dutch, English, Farsi (Persian), French, German, Italian, Japanese, Korean, Russian, Spanish, and Urdu. In conjunction with the base linguistics analysis performed by other RLP processors, the Named Entity Extractor [156] uses its statistical model to identify named entities of these types when processing a document in one of these languages.

- **Regular Expressions.** RLP comes with a collection of regular expressions in **regex-config.xml** that the Regular Expression [163] processor uses to find named entities. This collection includes language-specific expressions for distance, longitude and latitude, number, date, and time. It also includes generic expressions (all languages) for credit card number, email address, money, number, personal ID number, phone number, URL, and Universal Transverse Mercator coordinates.

  You can edit existing regular expressions and add your own regular expressions to this collection, potentially expanding the scope of languages and entity types.

- **Gazetteers.** There are two types of gazetteers: binary gazetteers and text gazetteers. A gazetteer of either type contains a list of names associated with an entity type. A gazetteer may be language specific or may apply to all languages. The Gazetteer [144] processor returns text that matches any entry from the gazetteers listed in **gazetteer-options.xml**. You can normalize whitespace, ignore case, and perform other normalizations to customize the matching algorithm the processor applies. Like regular expressions, you can use text gazetteers to expand the scope of languages and entity types

  Binary gazetteers are provided by Basis. Users cannot view or edit binary gazetteers.

  A text gazetteer is a file you create with a list of the names you associate with a particular entity type and, optionally, with a particular language.

- **Redactor.** The Named Entity Redactor [160] processor uses weighting factors that you can customize and, optionally, entity length to resolve any conflict between the three preceding processors. If, for example, the Named Entity Extractor and the Gazetteer each identify the same phrase (or overlapping text) as a named entity (perhaps one says it is a PERSON, and the other a LOCATION), the weights you have assigned tell the Named Entity Redactor which source and named entity type to report.

4.3. Setting Up an RLP Application to Return Named Entities

Creating an RLP Application [17] walks you through the process of establishing an RLP application to extract information from input documents. The only special concerns when it comes to getting named entitities is to be sure you include in your RLP context the language processors that are required to identify named entities in the input.

The following context includes the language processors required to get named entities from plain text in all the languages that RLP supports.

```
<contextconfig>
 <languageprocessors>
<languageprocessor>Unicode Converter </languageprocessor>
   <languageprocessor>RLI</languageprocessor>
   <languageprocessor>RCLU</languageprocessor>
   <languageprocessor>BL1</languageprocessor>
   <languageprocessor>JLA</languageprocessor>
   <languageprocessor>CLA</languageprocessor>
   <languageprocessor>KLA</languageprocessor>
   <languageprocessor>Tokenizer</languageprocessor>
   <languageprocessor>SentenceBoundaryDetector</languageprocessor>
   <languageprocessor>ARBL</languageprocessor>
  <languageprocessor>FABL</languageprocessor>
  <languageprocessor>URBL</languageprocessor>
   <languageprocessor>BaseNounPhrase</languageprocessor>
   <languageprocessor>NamedEntityExtractor</languageprocessor>
   <languageprocessor>Gazetteer</languageprocessor>
```

```
    <languageprocessor>RegExpLP</languageprocessor>
    <languageprocessor>NERedactLP</languageprocessor>
  </languageprocessors>
</contextconfig>
```

If your input text is not plain text, see Preparing Marked-Up or Binary data [79] , which provides information about handling HTML, XML, PDF, RTF, Microsoft Office documents, and XML.

4.4. Accessing the Named Entities that RLP has Found

For the data that RLP returns for each named entity it finds, see NAMED_ENTITY [86] .

To use the C++ API to access named entities, see Using the Named Entity Iterator [97] .

To use the Java API, see RLPResultRandomAccess [103] .

4.5. The Standard Set of Named Entities

With no user modifications, Named Entity Extractor, Gazetteer, and the Regular Expression processor return the named entity types identified in the table below.

As noted in the table, some of the regular expressions for locating named entities are language-specific; others are generic.

Table 4.1. Named Entitity Types , Languages, and Processors

Named Entity	Languages	Language Processor
FACILITY	Arabic, English, Japanese, Korean, Farsi (Persian), Russian, Simplified Chinese, Traditional Chinese, Urdu, Upper-Case English	Named Entity Extractor
GPE	Arabic, English, Japanese, Korean, Farsi (Persian), Russian, Simplified Chinese, Traditional Chinese, Urdu, Upper-Case English	
LOCATION	Arabic, Dutch, English, French, German, Italian, Japanese, Korean, Farsi (Persian), Russian, Simplified Chinese, Spanish, Traditional Chinese, Urdu, Upper-Case English	
NATIONALITY	Arabic, English, Japanese, Farsi (Persian), Simplified Chinese, Traditional Chinese, Urdu, Upper-Case English	
ORGANIZATION	Arabic, Dutch, English, French, German, Italian, Japanese, Korean, Farsi (Persian), Russian, Simplified Chinese, Spanish, Traditional Chinese, Urdu, Upper-Case English	

Named Entity	**Languages**	**Language Processor**
PERSON	Arabic, Dutch, English, French, German, Italian, Japanese, Korean, Farsi (Persian), Russian, Simplified Chinese, Spanish, Traditional Chinese, Urdu, Upper-Case English	Named Entity Extractor
RELIGION	Arabic, English, Japanese, Farsi (Persian), Simplified Chinese, Traditional Chinese, Urdu, Upper-Case English	
TITLE	Arabic, English, Japanese, Korean, Russian, Simplified Chinese, Traditional Chinese, Upper-Case English	
NATIONALITY	Russian	Gazetteer
RELIGION	Russian	
TITLE	French, German, Italian, Spanish, Dutch, Portuguese, Farsi (Persian), Urdu	
IDENTIFIER:CREDIT_CARD_NUM	generic	Regular Expression
IDENTIFIER:DISTANCE	Arabic, German, Dutch, English, Upper-Case English, Spanish, Farsi (Persian), French, Portuguese, Italian, Russian, Japanese, Simplified Chinese, Traditional Chinese, Korean	
IDENTIFIER:EMAIL	generic	
IDENTIFIER:LATITUDE_LONGITUDE	Arabic, German, Dutch, English, Upper-Case English, Spanish, Farsi (Persian), French, Portuguese, Italian, Hungarian, Czech, Greek, Russian, Polish, Japanese, Simplified Chinese, Traditional Chinese, Korean	
IDENTIFIER:MONEY	generic	
IDENTIFIER:NUMBER	Arabic, German, Dutch, English, Upper-Case English, Spanish, Farsi (Persian), French, Portuguese, Italian, Russian, Japanese, Simplified Chinese, Traditional Chinese	
IDENTIFIER:PERSONAL_ID_NUM	generic	
IDENTIFIER:PHONE_NUMBER	generic	
IDENTIFIER:URL	generic	
IDENTIFIER:UTM	generic	

Named Entity	Languages	Language Processor
TEMPORAL:DATE	Arabic, German, Dutch, English, Upper-Case English, Spanish, Farsi (Persian), French, Portuguese, Italian, Hungarian, Czech, Greek, Russian, Polish, Japanese, Simplified Chinese, Traditional Chinese, Korean	Regular Expression
TEMPORAL:TIME	Arabic, German, Dutch, English, Upper-Case English, Spanish, Farsi (Persian), French, Portuguese, Italian, Russian, Japanese, Simplified Chinese, Traditional Chinese, Korean	

Definitions

- FACILITY: A man-made structure or architectural entity such as a building, stadium, monument, airport, bridge, factory, or museum.
- GPE: An geo-political entity comprised of three elements: a population, a geographic location and a government. A country, state, city, or other location that contains both a population and a centralized government.
- IDENTIFIER:UTM: Universal Transverse Mercator coordinates for a geographical location.
- LOCATION: Name of a geographically defined place such as a continent, body of water, mountain, park, or full address. It also refers to a region that either spans GPE boundaries (such as *Middle East*, *Northeast*, *West Coast*) or is contained within a larger GPE (such as *Sunni Triangle*, *Chinatown*).

  Important: For those languages that do not include GPE, LOCATION is expanded to include GPE.
- NATIONALITY: Reference to a country or region of origin, such as *American* or *Swiss*.
- ORGANIZATION: A corporation, institution, government agency, or other group of people defined by an established organizational structure.
- PERSON: A human identified by name, nickname or alias.
- RELIGION: Reference to an organized religion or theology as well as its followers.
- TITLE: Appellation associated with a person by virtue of occupation, office, birth, or as an honorific.

4.6. Joining Adjacent Named Entities

If two named entities are adjacent, you may wish to join them into a single named entity. By default, adjacent TITLE elements are joined into a single TITLE element, and TITLE and PERSON elements (in either order) are joined into a single PERSON element. You can modify these specifications and add your own join specifications. [162] .

4.7. Consistency Returning Named Entities

The linguistic context in which an entity appears helps the Named Entity Extractor [156] identify and assign a type to the entity. Hence it is possible that different instances of an entity, appearing multiple times in a document, may be returned with different types.

If you want to guarantee that the Named Entity Redactor [160] always returns multiple occurrences of a given entity with the same type, no matter the linguistic context, RLP provides APIs for returning all instances of a named entity with the type assigned to the first instance.

For each language, the set method takes a boolean (true to enforce consistency), and the get method returns the current boolean setting. The default setting is false. For reference documentation, see the API Reference [api-reference/index.html].

C++. The `BT_RLP_NE_Iterator_Factory` class includes `SetConsistentType` and `GetConsistentType` functions

Java. `com.basistech.rlp.RLPResultAccess` and `com.basistech.rlp.ResultRandomAccess` include `setConsistentType` and `getConsistentType` methods.

C. The C API includes `BT_RLP_NE_Iterator_Factory_SetConsistentType` and `BT_RLP_NE_Iterator_Factory_GetConsistentType` functions.

.NET. The `Context` class includes `SetConsistentType` and `GetConsistentType` functions

4.8. Blacklisting Named Entities

If you want to exclude certain entities that are sometimes returned by the Named Entity Extractor [156] , you can create one or more blacklist dictionaries. Each dictionary you create applies to a specific entity type, and you may not use more than one dictionary for an entity type. The Named Entity Redactor [160] does not return entities for this type that are in this dictionary. You can instruct the Named Entity Redactor to log occurrences of blacklisted entities to a file.

For efficiency, a blacklist dictionary must be compiled into a binary form with big-endian or little-endian byte order to match the platform.

Procedure for Blacklisting Named Entities

1. Create the dictionary source file [50] .
2. Compile the user dictionary [51] .
3. Put the dictionary in an appropriate location [51] .
4. Edit the Nanmed Entity Redactor configuration file to include the blackist dictionary and (optionally) to log any blacklist entries that are encountered when processing documents [51] .

4.8.1. Creating a Blacklist Dictionary Source File

The source file for a blacklist dictionary is UTF-8 encoded. The file may begin with a byte order mark (BOM). Each entry is a single line. For example:

```
General Mechanics
Test User
Big Mac
```

Use Normalized Tokens in the Blacklist Entries

For those languages for which RLP returns normalized tokens (currently Arabic, Urdu, and Farsi), named entities are returned as normalized tokens. Accordingly, each token in the blacklist entries should be a NORMALIZED_TOKEN [86] .

4.8.2. Compiling the Blacklist Dictionary

For efficient processing, the dictionary must 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 SPARC), generate a binary dictionary on each of these platforms.

The script for generating a binary dictionary is ***BT_ROOT*/rlp/rlp/tools/build_blacklist.bat** for Windows and ***BT_ROOT*/rlp/rlp/tools/build_blacklist.sh** for Unix.

The `BT_ROOT` and `BT_BUILD` environment variables must be set. `BT_BUILD` specifies the platform identifier embedded in your SDK package file name (see Supported Platforms [13]).

Windows example:

```
set BT_ROOT="c:\Program Files\Basis Technology\btroot"
set BT_BUILD=ia32-w32-msvc80
```

You run this script from the command line with two arguments: the source file and the binary file to be created. For example:

```
build_blacklist.bat blacklist-person.txt blacklist-person-LE.bin
```

LE or BE. `LE` indicates that the byte order in this file is little endian. You can compile the same source file on a big-endian platform and replace `LE` with `BE` in the binary file name. As illustrated below in the Named Entity Redactor configuration file, you can use the same configuration file for both platforms: RLP replaces `<env name="endian"/>` with `LE` on a little-endian platform and with `BE` on a big-endian platform. If you are not using a configuration file that applies to both little-endian and big-endian platforms, you do not need to include `LE` or `BE` in the binary file name, or `<env name="endian"/>` in the configuration file.

4.8.3. Where to Put the Blacklist Dictionary

You can put the blacklist dictionary where you want. For example, you might want to create a blacklist directory: ***BT_ROOT*/rlp/blacklist**.

Place the binary file you have generated in the location of your choice.

4.8.4. Updating the Named Entity Redactor Configuration File

For each blacklist dictionary you generate, you must place a `blacklist` element in the Named Entity Redactor configuration file [162] : ***BT_ROOT*/rlp/etc/neredact-config.xml**. The blacklist elements must be contained in a `blacklists` element as illustrated below.

For example, suppose you create **blacklist-person.bin** (for PERSON) and **gpe-org** (for GPE), and place these files in ***BT_ROOT*/rlp/blacklist**.

```
<neredactconfig>
  <joiners>
    <joiner left='TITLE' right='TITLE' joined='TITLE'/>
    <joiner left='TITLE' right='PERSON' joined='PERSON'/>
    <joiner left='PERSON' right='TITLE' joined='PERSON'/>
  </joiners>
  <blacklists>
    <!-- No more than one dictionary per entity type-->
    <blacklist type='PERSON'><env name="root"/>/blacklist/bl-person-<env name="endian"/>.bin</blacklist>
    <blacklist type='GPE'><env name="root"/>/blacklist/bl-gpe-<env name="endian"/>.bin</blacklist>
  </blacklists>
  <blacklistlog><env name="root"/>/blacklist/blacklist.log</blacklistlog>
</neredactconfig>
```

Each `blacklist` element must have a `type` attribute identifying the entity type. The content of the element is the path to the binary file. At runtime, RLP replaces <env name="root"/> with the absolute path to ***BT_ROOT*/rlp**.

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. If you are not working on both big-endian and little-endian platforms, you can omit "BE" or "LE" from the binary filename and <env name="endian"> from the entry in the configuration file.

If you want to log blacklist entries when they are encountered, include a `blaclistlog`element as illustrated above with the path to the log file.

Each entry in the blacklist logfile is a line with 4 tab-separated fields:

1. Date and time. Format is yyyy-mm-dd:HH:MM:SS in 24-hours format
2. Source file or "buffer" (if the input data is processed from a buffer rather than a file)
3. Named entity type
4. The entity

For example:

```
2009-03-03:22:34:54    /data/file1.txt    PERSON    Test User
```

Note: user can specify date and time format as supported in C library function strftime. The default format is specified with `%Y-%m-%d:%H:%M:%S`. Set the `BT_BLACKLIST_DATETIME_FORMAT` environment variable to override the default date and time format.

For example (Unix):

```
export BT_BLACKLIST_DATETIME_FORMAT=%m/%d/%Y-%H:%M:%S
```

4.9. Extending the Coverage of Named Entities

You can extend the coverage for the entity types listed above as well as create new entitiy types.

The Named Entity Editor

The Rosette Demo (Windows only) includes a Named Entity Editor, which you can use with some restrictions to modify named entity definitions, create new named entity types, and extend the set of languages that are analyzed for named entities. See Using the Named Entities Editor [263] . The Named Entity Editor does not provide access to named regular expressions. You can edit the **XML** configuration files directly, which provides access to all features.

The Named Entity Extractor uses binary data files to locate entities for eight entity types: PERSON, LOCATION, ORGANIZATION, GPE (geo-political entity), FACILITY, RELIGION, NATIONALITY, and TITLE.

The Gazetteer and Regular Expressions processors are customizable in terms of the data (entities) they return, and the entity types and subtypes (predefined or user-defined) with which they tag individual entities.

Types and Subtypes. As explained in Result Types: NAMED_ENTITY [86] , an integer triple is generated for each named entity. The first two integers define the range of tokens that make up the entity. The third integer identifies the entity type and optional subtype, as well as the processor (Named Entity Extractor, Gazetteer, or Regular Expression). RLP maps these integers to strings (TYPE[:SUBTYPE]), such as "PERSON" and "ORGANIZATION:GOVERNMENT". By convention, the names for predefined types and subtypes are upper case. The integers and strings for predefined types and subtypes are specified in **bt_ne_types.h** (C++) and `com.basistech.rlp.RLPNENamedConstants` (Java), which also define the API for getting from integer to string and vice versa.

Defining your own Types and Subtypes. You can define your own types and subtypes. You define the name; RLP takes care of defining the unique integer to be used to identify this type/subtype and origin. Use the same API as for user-defined and predefined types to get from integer to string and vice versa.

For `Gazetteer`, you define types and lists of the names you want to find for each type. The types may be predefined or user defined. See Customizing Gazetteer [53] .

For the `Regular Expression` processor, you define the types and rules for finding named entities. The types may be predefined or user defined. See Creating Regular Expressions [56] .

You can also provide "weights" to predefined and user-defined types to determine how the Named Entity Redactor [160] resolves conflicts when more than one processor returns the same or an overlapping set of tokens.

4.9.1. Customizing Gazetteer

By its nature, Gazetteer [144] is designed to be customized such that it guarantees the return of words or phrases as named entities. This language processor uses user-created gazetteer files to identify specific words or phrases within the input text. A gazetteer may contain entries for a specific language or may be generic. If it is generic, it is applied to text in any language. See Gazetteer Dictionary Paths [145] .

Multiple files may be defined for multiple purposes, such as tracking famous personalities in news media, infectious diseases in journal articles, or specific product names and trademarks in market reports.

Important

The Gazetteer only uses the gazetteers specified in the Gazetteer options file, ***`BT_ROOT`*****/rlp/etc/gazetteer-options.xml**.

4.9.1.1. The Gazetteer Source File

Note: A user-defined gazetteer source file may either be a flat text file, as described below, or an XML file [55] .

Gazetteer processes text and isolates specific terms defined by the user in a Gazetteer Source File (GSF), which has the following properties:

- The file is UTF-8 encoded.

- Each comment line is prefixed with *#*.
- The first non-comment line is the TYPE[:SUBTYPE], which applies to the entire GSF, and will be used as the entity type name for output. Type and subtype may be predefined or user-defined [53] .
- The file contains user-defined entity strings, one per line.

For example, suppose you wantnumber to track common infectious diseases. You might create a GSF that looks like this:

```
# File: infectious-diseases-gazetteer.txt
#
DISEASE:INFECTIOUS
tuberculosis
e. coli
malaria
influenza
```

In many cases, a single GSF may not be enough. You may create as many GSFs as you like. For example, perhaps you also want to search for the scientific names of the infectious disease in the example above. You might create a file like this:

```
# File: latin-infectious-gazetteer.txt
#
DISEASE:INFECTIOUS
Mycobacterium tuberculosis
Escherichia coli
Plasmodium malariae
Orthomyxoviridae
```

Perhaps you wish to be able to track certain diseases by their causes:

```
# File: infectious-bacterial-gazetteer.txt
#
DISEASE:BACTERIAL
Escherichia coli
E. coli
Staphylococcus aureus
Streptococcus pneuminiae
Salmonella
```

Or perhaps you wish to track specific diseases showing resistance to antibiotics:

```
# File: resistant-diseases-gazetteer.txt
#
DISEASE:RESISTANT
Staphylococcus aureus
Streptococcus pneumoniae
Salmonella spp.
Campylobacter jejuni
Escherichia coli
Enterococcus faecium
```

Or the drugs used to treat them:

```
# File: antimicrobial-drugs-gazetteer.txt
#
DRUG:ANTIMICROBIAL
methicillin
vancomycin
```

```
macrolide
fluoroquinolone
```

Any or all of these gazetteer source files may be used by Gazetteer.

4.9.1.2. Using Multiple Gazetteer Source Files

The Gazetteer options file, ***BT_ROOT*/rlp/etc/gazetteer-options.xml**, specifies the gazetteers that are used when the Gazetteer runs. For example:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE gazetteerconfig SYSTEM "gazetteerconfig.dtd">
<gazetteerconfig>
  <DictionaryPaths>
    <DictionaryPath><env name="root"/>/rlp/source/samples/gazetteer1.txt</DictionaryPath>
    <DictionaryPath><env name="root"/>/rlp/source/samples/gazetteer2.txt</DictionaryPath>
     <!-- BinDictionaryPath elements for the binary gazetteers that Basis ships are not included
          in this view -->
  </DictionaryPaths>
</gazetteerconfig>
```

Suppose you wish to track infectious diseases, particularly those diseases that have developed resistance to antimicrobial drugs. Then you might configure **gazetteer-options.xml** as follows, using example GSFs from the previous section:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE gazetteerconfig SYSTEM "gazetteerconfig.dtd">
<gazetteerconfig>
  <DictionaryPaths>
    <DictionaryPath><env name="root"/>/rlp/source/samples/infectious-diseases-gazetteer.txt</DictionaryPath>
    <DictionaryPath><env name="root"/>/rlp/source/samples/resistant-diseases-gazetteer.txt</DictionaryPath>
    <DictionaryPath><env name="root"/>/rlp/source/samples/antimicrobial-drugs-gazetteer.txt</DictionaryPath>
     <!-- BinDictionaryPath elements for the binary gazetteers that Basis ships are not included
          in this view -->
  </DictionaryPaths>
</gazetteerconfig>
```

4.9.1.3. Gazetteer Source Files in XML

In addition to the text format described above, Basis Technology has defined an XML format for gazetteers. Use the following subset of **gazetteer.dtd**: [1]

```
 <!ELEMENT gazetteer (header,entities) >
 <!ELEMENT header (type) >
 <!-- Named Entity TYPE[:SUBTYPE] -->
 <!ELEMENT type (#PCDATA)>
 <!ELEMENT entities (entity+) >
 <!ELEMENT entity (names) >
 <!ELEMENT names (name+)>
 <!ELEMENT name (data)>
 <!--Text of the name -->
 <!ELEMENT data (#PCDATA)>
```

The following example shows the Infectious Diseases Gazetteer defined above in plain text [54] in the XML format:

[1]Gazetteers may be used for other purposes, such as providing input for a Rosette Name Index (see the *RLP Name Components Application Developer's Guide*), in which case the XML may use additional elements specified in **gazetteer.dtd**. The Gazetteer [144] processor ignores elements that do not appear in the subset defined here.

```
<?xml version='1.0' encoding='utf-8' standalone='no'?>
<!DOCTYPE gazetteer SYSTEM "gazetteer.dtd">
<gazetteer>
 <header>
  <type>DISEASE:INFECTIOUS</type>
 </header>
 <entities>
  <entity>
   <names>
    <name>
     <data>tuberculosis</data>
    </name>
    <name>
     <data>tb</data>
    </name>
   </names>
  </entity>
  <entity>
   <names>
    <name>
     <data>e. coli</data>
    </name>
   </names>
  </entity>
  <entity>
   <names>
    <name>
     <data>malaria</data>
    </name>
   </names>
  </entity>
  <entity>
   <names>
    <name>
     <data>influenza</data>
    </name>
   </names>
  </entity>
 </entities>
</gazetteer>
```

For more information, see Gazetteer [144] .

4.9.2. Creating Regular Expressions

The Regular Expression language processor [163] uses a configuration file (**regex-config.xml**) to map the strings matching each regular expression defined in that file to a language and a named entity `TYPE[:SUBTYPE]`. Add to or modify this file to enable the processor to find the desired named entities.

The `type` attribute for each `<regexp>` element specifies the type and optional subtype of the entities to be returned by the regular expression in the `<regexp>` element . Type and subtype may be predefined or user-defined [53] . For example, if the type is "COMPOUND" and subtype is "ORGANIC", `type="COMPOUND:ORGANIC"`.

The Regular Expression language processor uses the Tcl regular expression engine with some character class extensions. It is compatible with the Perl 5 regular expression syntax, with the exceptions noted below.

Differences with Perl 5

- Syntax:
  - \y is used (not \b) to match word boundaries.
  - \m and \M match only the beginnings and ends of words respectively.
- Lookbehind assertions are not supported.
- Recursive patterns are not supported.
- Conditional matches (?(*condition*)*yes-pattern*|*no-pattern*) are not supported.
- Atomic groups (?>group) are not supported.
- Using character properties (\p{} and \P{}) to match characters is not supported.

  The Basis Technology implementation has extended character classes to cover a number of character properties (see below).

4.9.2.1. Character Classes

Tcl supports the full Unicode locale. Character classes are extended to cover all Unicode characters.

Standard Character Classes. [:alpha:] A letter. [:upper:] An upper-case letter. [:lower:] A lower-case letter. [:digit:] A decimal digit. [:xdigit:] A hexadecimal digit. [:alnum:] An alphanumeric (letter or digit). [:print:] An alphanumeric (same as [:alnum:]). [:blank:] A space or tab character. [:space:] A character producing whitespace in displayed text. [:punct:] A punctuation character. [:graph:] A character with a visible representation. [:cntrl:] A control character.

Character Classes for Unicode Properties. [:Cc:] Control. [:Cf:] Format. [:Co:] Private Use. [:Cs:] Surrogate. [:Ll:]Lower-case letter. [:Lm:] Modifier letter. [:Lo:] Other letter. [:Lt:]Title-case letter. [:Lu:] Upper-case letter. [:Mc:] Spacing mark. [:Me:] Enclosing mark. [:Mn:] Non-spacing mark. [:Nd:] Decimal Number. [:Nl:] Letter number. [:No:] Other number. [:Pc:] Connector punctuation. [:Pd:] Dash punctuation. [:Pe:] Close punctuation. [:Pf:] Final punctuation. [:Pi:] Initial punctuation. [:Po:] Other punctuation. [:Ps:] Open punctuation. [:Sc:] Currency symbol. [:Sk:] Modifier symbol. [:Sm:] Mathematical symbol. [:So:] Other symbol. [:Zl:] Line separator. [:Zp:] Paragraph separator. [:Zs:] Space separator.

Character Classes for Writing Scripts. [:Arabic:] [:Armenian:] [:Bengali:] [:Bopomofo:] [:Braille:] [:Buginese:] [:Buhid:] [:Canadian_Aboriginal:] [:Cherokee:] [:Common:] [:Coptic:] [:Cyrillic:] [:Devanagari:] [:Ethiopic:] [:Georgian:] [:Glagolitic:] [:Greek:] [:Gujarati:] [:Gurmukhi:] [:Han:] [:Hangul:] [:Hanunoo:] [:Hebrew:] [:Hiragana:] [:Inherited:] [:Kannada:] [:Katakana:] [:Khmer:] [:Lao:] [:Latin:] [:Limbu:] [:Malayalam:] [:Mongolian:] [:Myanmar:] [:New_Tai_Lue:] [:Ogham:] [:Oriya:] [:Runic:] [:Sinhala:] [:Syloti_Nagri:] [:Syriac:] [:Tagalog:] [:Tagbanwa:] [:Tai_Le:] [:Tamil:] [:Telugu:] [:Thaana:] [:Thai:] [:Tibetan:] [:Tifinagh:] [:Yi:]

You can use these character class extensions just like the standard character classes.

For example, [[:Hiragana:]] matches a single Hiragana character, and [[:Zs:]] matches a whitespace character.

Reference Documentation. For a description of Tcl syntax for regular expressions, see Tcl Regular Expression Syntax [239] . Unless you specify otherwise (see "Metasyntax"), a regular expression is understood to be an Advanced Regular Expression (ARE) as described in that documentation.

Chapter 5. Using the RLP C API

5.1. Introduction

In response to customer requests, RLP now includes a C API. For reference documentation, see API Reference [api-reference/index.html].

To review the basic structure of an RLP application, consult Creating an RLP Application [17] , which also includes C++ and Java samples. The C samples that follow are designed to help you incorporate RLP functionality in your C applications.

5.2. Sample C Application

C source file: **rlp_sample_c.c**. Input parameters are passed in from the command line, and the RLP context configuration document is embedded in the source code.

```
// C includes
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <locale.h>

// Include the RLP interfaces
#include <bt_types.h> // bool, true, false, BT_Char16 etc.
#include <bt_rlp_c.h> // The RLP C API header.
#include <bt_rlp_ne_types.h> // BT_RLP_NET_ID_TO_STRING
#include <bt_xwchar.h> // bt_xutf16toutf8

// Forward declaration of private functions
static void handleResults(BT_RLP_ContextC* contextp, FILE* out);
static void log_callback(void* info_p, int channel, const char* message);
static void putus(const BT_Char16* str, FILE* out);

//Use an XML string or file to define a context. The context is an
//ordered sequence of processors and related settings that specify
//the processing that RLP performs on the input. This app defines
//the context in the string below.

//This is a general-purpose context configuration.
// - The Unicode Converter converts the input file from UTF-8 (or any other
//   Unicode encoding) to UTF-16 for internal processing.
// - Depending on the language of the input, the appropriate language analyzer
//   (BL1, JLA, CLA, KLA, or ARBL) performs a variety of tasks appropriate for
//   that language. For example, JLA tokenizes Japanese text, assigns part of
//   speech tags, and determines alphabetic (Hiragana) readings for native
//   Japanese words in Kanji.
// - BaseNounPhrase detects noun phrases.
// - SentenceBoundaryDetector delimits the sentences in the input.
// - Stopwords uses the language-specific stopwords dictionary to tag tokens
//   as stopwords.
// - NamedEntityExtractor, Gazeteer, RegExpLP, and NamedEntityRedactor locate
//   named entities.
static const char* CONTEXT =
  "<?xml version='1.0'?>"
  "<contextconfig>"
  "<languageprocessors>"
  "<languageprocessor>Unicode Converter</languageprocessor>"
  "<languageprocessor>BL1</languageprocessor>"
  "<languageprocessor>JLA</languageprocessor>"
  "<languageprocessor>CLA</languageprocessor>"
  "<languageprocessor>KLA</languageprocessor>"
  "<languageprocessor>Tokenizer</languageprocessor>"
  "<languageprocessor>SentenceBoundaryDetector</languageprocessor>"
  "<languageprocessor>ARBL</languageprocessor>"
  "<languageprocessor>FABL</languageprocessor>"
  "<languageprocessor>URBL</languageprocessor>"
  "<languageprocessor>Stopwords</languageprocessor>"
  "<languageprocessor>BaseNounPhrase</languageprocessor>"
  "<languageprocessor>NamedEntityExtractor</languageprocessor>"
  "<languageprocessor>Gazetteer</languageprocessor>"
  "<languageprocessor>RegExpLP</languageprocessor>"
  "<languageprocessor>NERedactLP</languageprocessor>"
  "</languageprocessors>"
  "</contextconfig>";

/**
 * 1. Process input parameters.
 * 2. Set up RLP environment.
 * 3. Set up RLP context for processing input.
 * 4. Process input.
 * 5. Work with the results.
 * 6. Clean up.
*/
int main(int argc, char* argv[])
{
  BT_Result rc;
  BT_RLP_EnvironmentC *envp;
  BT_RLP_ContextC *contextp;
  const char* btRoot;
  BT_LanguageID langID;
  const char* envConfig;
  const char* inputFile;
  const char* outputFile;
  FILE* out;

  if (!BT_RLP_CLibrary_VersionIsCompatible(BT_RLP_CLIBRARY_INTERFACE_VERSION)){
    fprintf(stderr, "RLP library mismatch: have C lib %ld expect %ld\nOr incompatibility "
        "between the core (%s) and C binding libraries.\n",
        BT_RLP_CLibrary_VersionNumber(),
        BT_RLP_CLIBRARY_INTERFACE_VERSION,
        BT_RLP_Library_VersionString());
    return 1;
  }

  //1. Process input parameters. This application gets the following
  //   from the command line:
  //      - Basis root (BT_ROOT) directory
  //      - language ISO639 code
  //      - pathname of the environment config file
  //      - pathname of the input file
  //      - pathname of the output file (encoded in utf-8)
  if ((argc != 6)
      || ((argc == 2) && (0 == strncmp(argv[1], "-h", 2)))) {
    fprintf(stderr,
        "Usage: %s BT_ROOT LANGUAGE ENV_CONFIG_FILE INPUT_FILE OUTPUT_FILE\n",
    argv[0]);
  return 1;
}

btRoot = argv[1];
// Get BT language ID (defined in bt_language_names.h)
// from ISO639 code.
langID = BT_LanguageIDFromISO639(argv[2]) ;
if (langID == BT_LANGUAGE_UNKNOWN) {
  fprintf(stderr, "Warning: Unknown ISO639 language code: %s\n", argv[2]);
  return 1;
}
envConfig = argv[3];
inputFile = argv[4];
outputFile = argv[5];

out = fopen(outputFile, "w");
if (out == 0) {
  fprintf(stderr, "Couldn't open output file: %s\n", outputFile);
  return 1;
}

//2. Set up the environment.

//2.1 Use BT_RLP_ENVIRONMENT static methods to set the root directory,
//    to designate a log callback function, and to set
//    log level.
BT_RLP_Environment_SetBTRootDirectory(btRoot);
fprintf(out, "The rlp root directory now is: %s\n", BT_RLP_Environment_RootDirectory());
BT_RLP_Environment_SetLogCallbackFunction((void*) stderr, log_callback);
//Log level is some combination of "warning,error,info" or "all".
BT_RLP_Environment_SetLogLevel("error");

//2.2 Create a new (empty) RLP environment.
envp = BT_RLP_Environment_Create();
if (envp == 0) {
  fprintf(stderr, "Env create failed.\n");
  return 1;
}

//2.3 Initialize the empty environment with the global environment
//    configuration file.
rc = BT_RLP_Environment_InitializeFromFile(envp, envConfig);
if (rc != BT_OK){
  fprintf(stderr, "Env initialize failed. %d returned.\n", rc);
  return 1;
}

//3. Get a context from the environment. In this case the context
//   configuration is embedded in the app as a string. It could also
//   be read in from a file.
rc = BT_RLP_Environment_GetContextFromBuffer(envp,
    (const unsigned char*)CONTEXT,
    strlen(CONTEXT),
    &contextp);
if (rc != BT_OK){
  fprintf(stderr, "GetContextFromBuffer failed. %d returned.\n", rc);
  BT_RLP_Environment_Destroy(envp);
  return 1;
```

```
 }

 //4. Use the context object to process the input file. Must include
 //   language id unless using RLI processor to determine language.
 rc = BT_RLP_Context_ProcessFile(contextp,
     inputFile, langID, "UTF-8", 0);
 if (rc != BT_OK){
   fprintf(stderr, "Unable to process the input file %s. %d returned.\n", inputFile, rc);
   BT_RLP_Environment_DestroyContext(envp, contextp);
   BT_RLP_Environment_Destroy(envp);
   return 1;
 }

 //5. Gather results of interest produced by processing the input text.
 handleResults(contextp, out);
 fprintf(stdout, "\nSee output file: %s\n\n", outputFile);
 //6. Remove any objects still lying around.
 BT_RLP_Environment_DestroyContext(envp, contextp);
 BT_RLP_Environment_Destroy(envp);
 return 0;
}

//5. Get results of interest and publish to file.
static void handleResults(BT_RLP_ContextC* contextp, FILE* out)
{
 BT_UInt32 lang;
 const char* langName;
 const char* encoding;
 BT_UInt32 len = 0;
 const BT_Char16* rawText;
 BT_RLP_TokenIteratorFactoryC* factoryp;
 BT_RLP_TokenIteratorC* tkitp;
 BT_RLP_ResultIteratorC* resitp;
 const BT_RLP_ResultC* bnpp; // Holder for a Base Nouse Phrase as well as a NE
 BT_UInt32 nn, i, j;
 const BT_Char16** tokens;
 int tkix = 0; // Index for the next token
 BT_UInt32 numReadings, numCompoundComponents;

 //5.1 Use the context object to get single-valued results: language,
 //    encoding, raw text, transcribed text (Arabic).
 lang = BT_RLP_Context_GetIntegerResult(contextp, BT_RLP_DETECTED_LANGUAGE);
 langName = BT_ISO639FromLanguageID(lang);
 encoding =
   (const char*)BT_RLP_Context_GetStringResult(contextp, BT_RLP_DETECTED_ENCODING);
 fprintf(out, "Language: %s (%d)\n", langName, lang);
 if (encoding != 0)
   fprintf(out, "Encoding: %s\n", encoding);

 rawText =
   BT_RLP_Context_GetUTF16StringResult(contextp, BT_RLP_RAW_TEXT, &len);

 fprintf(out, "Raw text: ");
 putus(rawText, out);

 //5.2 Use token iterator to get information related to tokens: tokens,
 //    token offsets, part of speech tags, stems, normalized tokens,
 //    stopwords, compounds, and readings.
```

```
factoryp = BT_RLP_TokenIteratorFactory_Create();
if (factoryp == 0){
  fprintf(stderr, "TokenIteratorFactory_Create failed. ");
  exit(1);
}

//Provide access to readings and compounds.
BT_RLP_TokenIteratorFactory_SetReturnReadings(factoryp, true);
BT_RLP_TokenIteratorFactory_SetReturnCompoundComponents(factoryp, true);

//Create the iterator and destroy the factory.
tkitp = BT_RLP_TokenIteratorFactory_CreateIterator(factoryp, contextp);
if (tkitp == 0) {
  fprintf(stderr, "CreateIterator failed. ");
  exit(1);
}

BT_RLP_TokenIteratorFactory_Destroy(factoryp);

nn = BT_RLP_TokenIterator_Size(tkitp);
tokens = (const BT_Char16**) malloc(sizeof(BT_Char16*) * nn); // Must free()
fprintf(out, "\n");

while(BT_RLP_TokenIterator_Next(tkitp)){
  const char *pos;
  const BT_Char16* token;
  BT_UInt32 ix, start, end;

  token = BT_RLP_TokenIterator_GetToken(tkitp);
  tokens[tkix++] = token;
  pos = BT_RLP_TokenIterator_GetPartOfSpeech(tkitp);
  ix = BT_RLP_TokenIterator_GetIndex(tkitp);
  start = BT_RLP_TokenIterator_GetStartOffset(tkitp);
  end = BT_RLP_TokenIterator_GetEndOffset(tkitp);

  fprintf(out, "#%d: s:%d, e:%d, ", ix, start, end);
  if (pos) {
    fprintf(out, "pos:%s, ", pos);
  }
  fprintf(out, "text:");
  putus(token, out);
  fprintf(out, "\n");

  //Get readings (for Japanese, render Kanji characters in
  // the Hiragana alphabet).
  numReadings = BT_RLP_TokenIterator_GetNumberOfReadings(tkitp);
  if (numReadings > 0) {
    fprintf(out, "  Readings: ");
    for (i = 0; i < numReadings;i++) {
      const BT_Char16* reading =
        BT_RLP_TokenIterator_GetReading(tkitp, i);
      putus(reading, out);
    }
    fprintf(out, "\n");
  }

  //Get compounds (see documentation for applicable languages).
  numCompoundComponents =
    BT_RLP_TokenIterator_GetNumberOfCompoundComponents(tkitp);
  if (numCompoundComponents > 0) {
    fprintf(out, "  Compound components: ");
    for (i = 0; i < numCompoundComponents; i++) {
      const BT_Char16* reading =
        BT_RLP_TokenIterator_GetCompoundComponent(tkitp, i);
      putus(reading, out);
      fprintf(out, " ");
    }
    fprintf(out, "\n");
  }
}
fprintf(out, "\n");

//5.3 Use result iterator to get other results, such as base noun phrases,
//    gazetteer names, and named entities. Note: Can use result iterator
//    to get any/all results.

//Get base noun phrases.
resitp = BT_RLP_Context_GetResultIterator(contextp, BT_RLP_BASE_NOUN_PHRASE);
while ((bnpp = BT_RLP_ResultIterator_Next(resitp)) != NULL){
  //The result is a pair of integers, indexes of the first and last + 1
  //tokens in the base noun phrase.
  BT_UInt32 first, last;

  BT_RLP_Result_AsIntegerPair(bnpp, &first, &last);
  fprintf(out, "Base Noun Phrase: ");
  // Use the tokens vector to construct the base noun phrases.
  for(j = first; j < last; j++) {
    putus(tokens[j], out);
    fprintf(out, " ");
  }
  fprintf(out, "\n");
}

BT_RLP_Context_DestroyResultIterator(contextp, resitp);

//Get named entities.
resitp = BT_RLP_Context_GetResultIterator(contextp, BT_RLP_NAMED_ENTITY);
while ((bnpp = BT_RLP_ResultIterator_Next(resitp)) != NULL){
  //Each result is three integers, indexes of the first and last + 1
  //tokens, and type/subtype.
  BT_UInt32 first, last, type;
  const char* typeName;

  BT_RLP_Result_AsIntegerTriple(bnpp, &first, &last, &type);
  typeName = BT_RLP_NET_ID_TO_STRING(type);
  fprintf(out, "%s ", typeName);
  //Uses token indexes to assemble the named entity.
  //Convert tokens from UTF-16 to UTF-8.
  for (i = first; i < last; i++) {
    putus(tokens[i], out);
    fprintf(out, " ");
  }
  fprintf(out, "\n");
}

//Cleanup
free((void*)tokens);
BT_RLP_Context_DestroyResultIterator(contextp, resitp);
```

```
  BT_RLP_TokenIterator_Destroy(tkitp);
  fprintf(out, "End of sample program.\n");
  fclose(out);
}

/**
* The application registers this function to receive diagnostic log entries.
* RLP Environment LogLevel determines which message channels (error, warning.
* info) are posted to the callback.
*/
static void log_callback(void* info_p, int channel, char const* message)
{
  static char* szINFO     = "INFO  : ";
  static char* szERROR    = "ERROR : ";
  static char* szWARN     = "WARN  : ";
  static char* szUNKNOWN  = "UNKWN : ";
  char* szLevel = szUNKNOWN;
  switch(channel) {
    case 0:
      szLevel = szWARN;
      break;
    case 1:
      szLevel = szERROR;
      break;
    case 2:
      szLevel = szINFO;
      break;
  }
  fprintf((FILE*) info_p, "%s%s\n", szLevel, message);
}

/*
 * Output a UTF-16 string to a file in UTF-8.
 */
static void putus(const BT_Char16* t, FILE* out)
{
#define BYTEBUF_MAX_SIZE 1024
  static char bytebuf[BYTEBUF_MAX_SIZE];
  bt_xutf16toutf8(bytebuf, t, BYTEBUF_MAX_SIZE);
  fputs(bytebuf, out);
}

/*
Local Variables:
mode: c
tab-width: 2
c-basic-offset: 2
End:
*
```

5.3. Sample C Application For Handling Arabic Alternative Analyses

C source file: **ar-rlp_sample_alternatives_c.c**. Input parameters are passed in from the command line, and the RLP context configuration document is embedded in the source code.

This application is similar to the sample documented above. The important distinction is that it is designed to handle Arabic text and to extract alternative analyses (lemmas, roots, stems, normalized tokens, and parts of speech). The following fragments highlight the distinctive features of this sample.

```
//Set the lemma, root and alternative results properties.
BT_RLP_Context_SetPropertyValue(contextp, "com.basistech.arbl.roots","true");
BT_RLP_Context_SetPropertyValue(contextp, "com.basistech.arbl.lemmas","true");
BT_RLP_Context_SetPropertyValue(contextp, "com.basistech.arbl.alternatives","true");

//Use a token iterator (*tkitp) to get each token and to get alternative analyses.
while(BT_RLP_TokenIterator_Next(tkitp)){
    const BT_Char16* token;
    BT_UInt32 ix, start, end;

    token = BT_RLP_TokenIterator_GetToken(tkitp);
    assert(token!=0);
    tokens[tkix++] = token;

    ix = BT_RLP_TokenIterator_GetIndex(tkitp);
    start = BT_RLP_TokenIterator_GetStartOffset(tkitp);
    end = BT_RLP_TokenIterator_GetEndOffset(tkitp);


    printf("#%d: s:%d, e:%d, token:", ix, start, end);
    putus(token);
    printf("\n");

//Get alternative analyses (for Arabic only).
    numAnalyses = BT_RLP_TokenIterator_GetNumberOfAnalyses(tkitp);
    if (numAnalyses > 0) {

        const BT_Char16 * altNorm;
        const BT_Char16 * altLemma;
        const BT_Char16 * altStem;
        const BT_Char16 * altRoot;
        const char * altPOS;
        int count=0;

        printf("   Alternative Analyses: \n");

        while(BT_RLP_TokenIterator_NextAnalysis(tkitp)) {
            ++count;

            altNorm = BT_RLP_TokenIterator_GetNormalForm(tkitp);
            printf("\tNormal\t%d :", count);
            putus(altNorm);
            printf("\n");

            altPOS = BT_RLP_TokenIterator_GetPartOfSpeech(tkitp);
            printf("\tPOS\t%d :%s\n", count, altPOS);

            altStem = BT_RLP_TokenIterator_GetStemForm(tkitp);
            printf("\tStem\t%d :", count);
            putus(altStem);
            printf("\n");

            altLemma = BT_RLP_TokenIterator_GetLemmaForm(tkitp);
            //com.basistech.arbl.lemmas could be "false".
            if (altLemma!=0){
                printf("\tLemma\t%d :", count);
                putus(altLemma);
                printf("\n");
            }

            altRoot = BT_RLP_TokenIterator_GetRootForm(tkitp);
            //com.basistech.arbl.roots could be "false".
            if (altRoot!=0){
                printf("\tRoot\t%d :", count);
                putus(altRoot);
                printf("\n");
            }
            printf("\n");
        }
        printf("\n");
    }
}
```

5.4. Sample C Application for Examining the RLP License

C source file: **examine_license_c.c**. Input parameters (the Basis Root directory and the path to the RLP environment configuration file) are passed in from the command line.

This application generates the list of features that your RLP license authorizes.

5.5. Building and Running the Sample C Applications

The source files for the C samples are in ***BT_ROOT*/rlp/samples/capi**:

- **rlp_sample_c.c**
- **ar-rlp_sample_alternatives_c.c**
- **examine_license_c.c**

5.5.1. Building the C Sample Applications

RLP supplies a script for building the C++ sample programs. On Windows platforms, the script is ***BT_ROOT*/rlp/samples/capi/build.bat**. On Unix platforms, the script is ***BT_ROOT*/rlp/samples/capi/build.sh**.

Important

The compiler and linker for your platform must be on your path.

If you are running Windows 32, you can run **vsvars32.bat** from the **Common7\Tools** directory in your Visual Studio installation. If you are running Windows 64, run **vcvarsamd.bat** from the **VC\bin\amd64\vcvarsamd64.bat** directory in your Visual Studio installation

If you are running Unix, be sure you have set the PATH environment variable to include the compiler and linker for your platform.

Before you call the script, you must set the *BT_BUILD* environment variable. You must also run the script from the ***BT_ROOT*/rlp/samples/capi** directory.

Windows example:

```
set BT_BUILD=ia32-w32-msvc80
cd \btroot\rlp\samples\capi
build.bat
```

Unix example:

```
export BT_BUILD=ia32-glibc22-gcc32
cd ~/btroot/rlp/samples/capi
./build.sh
```

The build script calls another script in the same directory with *`BT_BUILD`* embedded in the script name: **build_*`BT_BUILD`*_c_samples.[sh|bat]**. You can examine this script for the command that is used to compile and link each of the C samples.

An executable for each sample program with the same name as the C source file is placed in ***`BT_ROOT`*/rlp/bin/*`BT_BUILD`*.**

5.5.2. Running the C Samples

The RLP distribution includes a script for running the C samples: **go-c-samples.bat** (Windows) or **go-c-samples.sh** (Unix). This script is in ***`BT_ROOT`*/rlp/samples/scripts/*`BT_BUILD`*** . The script runs both samples. You can use it as a model for creating your own command-line scripts.

On Unix

On Unix platforms, you must set `LD_LIBRARY_PATH` (or its equivalent environment variable for your Unix operating system) to include the RLP library directory: ***`BT_ROOT`*/rlp/lib/*`BT_BUILD`*** .

Chapter 6. Using the .NET API

6.1. Introduction

RLP includes a .NET wrapper that provides a .NET API for customers on the `ia32-w32-msvc80` and `amd64-w64-msvc80` platforms: Windows 32-bit and 64-bit with the Visual Studio 8.0 compiler. The .NET wrapper uses the .NET 2.0 Framework to provide access to RLP functionality. For reference documentation, see API Reference [api-reference/index.html].

To review the basic structure of an RLP application, consult Creating an RLP Application [17] , which also includes C++ and Java samples. The C# sample that follows is designed to help you incorporate RLP functionality in your .NET applications.

6.2. Sample C# Application

C# source file: **RLPSample.cs**.

Input parameters for *`BT_ROOT`* (the Basis root directory), language ID, and input file are passed in from the command line. The sample uses the standard environment configuration file and one of the sample context files used for processing Unicode input.

```
using System;
using System.IO;
using System.Text;
using System.Collections;
using BasisTechnology.RLP;

namespace rlp_sample_csharp
{
    /// <summary>
    /// Basis Technology RLP C# sample program.
    /// </summary>
    class RLPSample
    {
        /// <summary>
        /// The main entry point for the Basis Technology RLP C# sample program.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            Console.WriteLine("");
            Console.WriteLine("Basis Technology RLP C# sample program.");
            Console.WriteLine("");

            if(args.Length != 3)
            {
                Console.WriteLine("Usage: rlp_sample_csharp [rlp-root] [language] [input-file]");
                return;
            }

            //
            // Extract command line options
            //
            string rlp_root      = args[0];
            string language_code = args[1];
            string inputFile     = args[2];
```

```
//
// Define the RLP configuration files we will use.
//
string env_def      = rlp_root + "/etc/rlp-global.xml";
string context_def  = rlp_root + "/samples/etc/rlp-context-no-op.xml";

//
// Echo our runtime parameters
//
Console.WriteLine("   RLP root dir:              " + rlp_root);
Console.WriteLine("   Environment definition file: " + env_def);
Console.WriteLine("   Context definition file:     " + context_def);
Console.WriteLine("   Language:                  " + language_code);
Console.WriteLine("   Input file:                " + inputFile);
Console.WriteLine("");

// Set the RLP root directory so that RLP executables
// can be located at run time.
BasisTechnology.RLP.Environment.SetRootDirectory(rlp_root);
// Get the numeric ID of the language desired
LanguageID lid = LanguageNameUtils.LanguageIDFromISO639(language_code);
// Read the input file
string text;
using(StreamReader fs = File.OpenText(inputFile))
{
   text = fs.ReadToEnd();
}

// Create the RLP environment object.
using (BasisTechnology.RLP.Environment env = BasisTechnology.RLP.Environment.Create())
{
   try
   {
      env.InitializeFromFile(env_def);
   }
   catch (BasisTechnology.RLP.Error e)
   {
      Console.WriteLine(e.GetMessage());
      System.Environment.Exit(1);
   }
   // Create the RLP processing context.  (We can create as many as needed.)
   using (Context ctx = env.CreateContextFromFile(context_def))
   {
      // Process the text
      ctx.ProcessBuffer(text, lid);
      // Access results
      bool safeOnly = true;
      object r = ctx.GetResultData(ResultType.TOKEN, safeOnly);
      // The TOKEN result is an array of strings.
      string[] tokens = (string[])r;
      int ntoks = tokens.Length;
      // The COMPOUND result is a hash table.
      // The key is the token index
      r = ctx.GetResultData(ResultType.COMPOUND, safeOnly);
      Hashtable comps = (Hashtable)r;
      // The NAMED_ENTITY result is an array of NEData objects.
      NamedEntityData[] NE = ctx.GetNamedEntityResultData(true);
      // Print results to stdout as UTF-8.
```

```
                using (StreamWriter sw =
                      new StreamWriter(Console.OpenStandardOutput(), System.Text.Encoding.UTF8))
                {
                    sw.WriteLine("TOKENS:");
                    for( uint ix = 0; ix < ntoks; ix++ )
                    {
                        sw.Write(tokens[ix]);
                        if( comps != null )
                        {
                            string[] strs = (string[])comps[ix];
                            if( strs != null )
                            {
                                sw.Write("\t(Compound components:");
                                for( uint i = 0; i < strs.Length; i++ )
                                {
                                    if( i > 0)
                                    {
                                        sw.Write(" - ");
                                    }
                                    sw.Write(strs[i]);
                                }
                                sw.Write(")");
                            }
                        }
                        sw.WriteLine();
                    }
                    sw.WriteLine("Named Entities:");
                    if(NE!= null)
                        for(uint i=0; i < NE.Length; i++)
                        {
                            sw.WriteLine(NE[i].NormalizedNamedEntity + " - " + NE[i].GetNETypeName());
                        }
                }
            }
        }
    }
}
```

6.3. Building and Running the Sample C# Application

The executable for the sample C# application is shipped with the release, so you do not need to build the application to run it. You may, however, be interested in modifying and rebuilding the application.

The source file for the C# sample is ***BT_ROOT*/rlp/samples/csharp/rlp_sample_csharp/RLPSample.cs**.

6.3.1. Building the Sample C# Application

RLP supplies a script for building the C# sample program: ***BT_ROOT*/rlp/samples/csharp/rlp_sample_csharp/build.bat**.

Important

The compiler and linker for your platform must be on your path.

On the Windows 32 platform, you can run **vsvars32.bat** from the **Common7/Tools** directory in your Visual Studio installation.

Before you call the script, you must set the *BT_BUILD* environment variable. You must also run the script from the ***BT_ROOT*/rlp/samples/csharp/rlp_sample_csharp** directory.

Example:

```
set BT_BUILD=ia32-w32-msvc80
cd \btroot\rlp\samples\csharp\rlp_sample_csharp
build.bat
```

The build script calls another script with in the same directory with *BT_BUILD* embedded in the script name: **build_*BT_BUILD*_csharp_samples.bat**. You can examine this script for the command that is used to compile and link the C# sample.

An executable with the same name as the C# source file is placed in ***BT_ROOT*/rlp/bin/*BT_BUILD*.**

6.3.2. Running the Sample C# Application

The RLP `ia32-w32-msvc80` and `amd64-w64-msvc80`distributions include a script for running the C# sample: **go-csharp-samples.bat**. Depending on your platform, this script is in ***BT_ROOT*/rlp/samples/scripts/ia32-w32-msvc80** or ***BT_ROOT*/rlp/samples/scripts/amd64-w64-msvc80**.

Use the command-line prompt to navigate to the directory that contains the script.

The C# sample displays the parameters it is using, parses the input file, and outputs information about the input to standard output. The output includes each token. If the input contains compounds (such as German, Dutch, and Japanese text), the output also identifies the individual components that make up each compound.

The output is encoded as UTF-8, so you may want to direct it to a file, which you can then read with an application such as **Notepad** that can correctly display UTF-8.

For example, to run the sample and view the output:

```
go-csharp-samples.bat >out.txt
Notepad out.txt
```

Chapter 7. Processing Multilingual Text

Text files containing material written in more than one language are not uncommon. This chapter provides information about using RLP to process multilingual text.

Keep in mind that the procedure documented in Creating an RLP Application [17] is based on the assumption that you are using RLP to process input text in a single language. If you include the required language processors (and you have the necessary RLP license), you can use the same context to process files in a variety of languages, but each file contains text in a single language. Either you identify the language when you use the context to process the input, or you include the Language Identifier (RLI) in the context, in which case RLI identifies the language for the language processors that follow.

7.1. Strategy for Handling Multilingual Text

If you want to extract information from a text file with text in more than one language, do the following:

1. Use an RLP context with the Rosette Language Boundary Locator [73] (RLBL) to identify language regions within the input text. Each language region contains contiguous text in a single language.
2. Use another RLP context to process each region [74] individually.

7.2. RLBL

Formerly known as the Multilingual Language Identifier (MLI), the Rosette Language Boundary Locator (RLBL) is a collection of three language processors that you can use to detect boundaries between the language regions within multilingual input. You include these processors in an RLP context in the order in which they appear below. [1]

1. Text Boundary Detector [186] detects sentence-level text boundaries in a language-independent fashion.
2. Script Boundary Detector [184] detects boundaries between language scripts, such as Latin, Cyrillic, Arabic, and Hangul (Korean).
3. Language Boundary Detector [152] uses text boundaries, script boundaries, and RLI functionality, to identify language regions.

7.2.1. RLBL Context

The context you define to process multilingual text should contain the processors to convert the input to plain text in UTF-16 encoding (see Preparing the Input [18]) and the three RLBL processors in the correct order, but no other language processors (the other language processors handle single-language text). Such a context produces raw text (a UTF-16 image of the input) and the information you need to identify each language region and extract it from the raw text.

A standard RLBL context configuration is as follows. Note the shortened processor names ("`Detector`" has been removed): `Text Boundary`, `Script Boundary`, and `Language Boundary`. For information about properties you can set, see Language Boundary Detector Context Properties [152] .

```
<?xml version='1.0'?>
<contextconfig>
```

[1]For information about the Unicode standards regarding text boundaries and script names, see Unicode Standard Annex #29 [http://www.unicode.org/reports/tr29] and Unicode Standard Annex #24 [http://www.unicode.org/reports/tr24].

```
<languageprocessors>
  <languageprocessor>Unicode Converter</languageprocessor>
  <languageprocessor>Text Boundary</languageprocessor>
  <languageprocessor>Script Boundary</languageprocessor>
  <languageprocessor>Language Boundary</languageprocessor>
</languageprocessors>
  </contextconfig>
```

7.3. Processing Language Regions

Once you have identified language regions, you can process each region with a context designed to handle text in a single language. The Language Boundary Detector provides the information you need to extract each language region from the raw text and to process the region with the appropriate language identifier.

7.3.1. Single-Language Context

The input is UTF-16 raw text, so the context configuration for a language region does not require a processor to generate raw text (see Processing UTF-16 input [27]). The following context configuration provides general coverage. If the text for a region is Japanese, for example, JLA processes the text, and the other language-specific processors (BL1, CLA, KLA, and ARBL) do nothing.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE contextconfig
        SYSTEM "http://www.basistech.com/dtds/2003/contextconfig.dtd">
<contextconfig>
  <languageprocessors>
    <languageprocessor>BL1</languageprocessor>
    <languageprocessor>JLA</languageprocessor>
    <languageprocessor>CLA</languageprocessor>
    <languageprocessor>KLA</languageprocessor>
    <languageprocessor>Tokenizer</languageprocessor>
    <languageprocessor>SentenceBoundaryDetector</languageprocessor>
    <languageprocessor>ARBL</languageprocessor>
   <languageprocessor>FABL</languageprocessor>
   <languageprocessor>URBL</languageprocessor>
    <languageprocessor>Stopwords</languageprocessor>
    <languageprocessor>BaseNounPhrase</languageprocessor>
    <languageprocessor>NamedEntityExtractor</languageprocessor>
    <languageprocessor>Gazetteer</languageprocessor>
    <languageprocessor>RegExpLP</languageprocessor>
    <languageprocessor>NERedactLP</languageprocessor>
  </languageprocessors>
</contextconfig>
```

7.4. From the Coding Perspective

After you process input parameters and set up the environment, do the following:

1. Instantiate two context objects: an RLBL context for processing the input text and a standard context for processing each language region.

2. Use the RLBL context to process the input text.

3. Get the following result data from the RLBL context: raw text and language regions.

   The raw text is the input text in UTF-16.

For each language region, the result data (LANGUAGE_REGION [85]) is 6 integers. Of primary interest are the raw text offsets that delimit each region and the language identifier for the region.

Note

You can also collect result data for script regions and sentence-level text boundaries. If you want sentence boundaries, you can get more accurate data for each language region during the next step.

4. Process each region with the second context, and use that context to get results of interest.

   For each region, the input is a portion of the UTF-16 raw text with the appropriate language identifier.

5. Perform the required cleanup.

7.4.1. Code examples

The following code fragments illustrate the process for looping through the regions returned by an RLBL context, using a standard context to process each region, and calling a method to handle the results for each region.

Both fragments use two RLP context objects: `rlblContext` and `context`. The configuration for these contexts is along the lines illustrated above. See RLBL Context [73] and Single-Language context [74] .

For details about setting up the RLP environment, instantiating a context, processing an input file, and handling the result data for each language region, see Creating an RLP Application [17] .

7.4.2. C++ Fragment

```
//Multilingual text has been processed with rlblContext.


//Get raw text.
BT_UInt32 len = 0;
const BT_Char16* rlblRawText =
        rlblContext->GetUTF16StringResult(BT_RLP_RAW_TEXT, len);

//Get language regions.
BT_RLP_ResultIterator* rlblResult_iter =
        rlblContext->GetResultIterator(BT_RLP_LANGUAGE_REGION);


//Get each region, process it, and handle the results.
const BT_RLP_Result* rlblResult;
while ((rlblResult = rlblResult_iter->Next()) != NULL){
  BT_UInt32 numbers[6];
  rlblResult->AsUnsignedIntegerVector(numbers, 6);
  BT_UInt32 start, end, level, type, script;
  BT_LanguageID language;
  start = numbers[0];
  end = numbers[1];
  level = numbers[2];
  type = numbers[3];
  script = numbers[4];
  language = BT_LanguageID(numbers[5]);
//Get pointer to start of region in the raw text.
BT_Char16* rawRegion = rlblRawText + start;
//Use pointer to start of region, length of region, and language
//identifier to process the raw text for the language region.
   BT_Result rc =
       context->ProcessUTF16Buffer(rawRegion, end - start, language);
if (rc !=BT_OK) {
  cerr << "Unable to process region." << endl;
  delete rlblContext;
  delete context;
  delete rlp;
  return 1;
 }
//Handle results for the region.
handleRegion(context);
}
```

For examples of how to extract result data for each region, see See `handleResults()` in rlp_sample [28] .

The complete application from which the preceding fragment has been extracted is in **samples/ cplusplus**: **rlbl_sample.cpp**. See Building and Running the Sample C++ Applications [40] .

7.4.3. Java Fragment

```
////Multilingual text has been processed with rlblContext.

//Get raw text.
RLPResultAccess resultAccess = new RLPResultAccess(rlblContext);
String rawText = resultAccess.getStringResult(RLPConstants.RAW_TEXT);

//Get Language regions.
List regions = resultAccess.getListResult(RLPConstants.LANGUAGE_REGION);

//Work with each region.
Iterator iter = langRegionList.iterator();
while (iter.hasNext()){
 int[] langRegion = (int[])iter.next();
 int start = langRegion[0];
 int end = langRegion[1];
 int level = langRegion[2];
 int type = langRegion[3];
 int script = langRegion[4];
 int langId = langRegion[5];

 //Process raw text for the region.
 context.process(rawText.substring(start, end), langId);
 //Handle the results for the region.
 handleResults(context);
}
```

For examples of how to extract result data for each region, see See `handleResults()` in RLPSample [34] .

The complete application from which the preceding fragment has been extracted is in **samples/java**: **MultiLang.java** and **MultiLang.properties**. See Building and Running the Sample Java Applications [41] .

Chapter 8. Preparing Your Data for Processing

Most of the RLP language processors process text in Unicode UTF-16LE or UTF-16BE (little-endian or big-endian byte order, depending on the platform byte order). For accurate linguistic analysis, the text should be plain text; it should not contain markup or a binary format, such as is found in HTML, XML, PDF, or Microsoft Office documents. The MIME type (document type) should be text/plain; if the MIME type is otherwise (such as text/html or application/pdf), plain text should be extracted from the input.

The text you want to process is likely not to be UTF-16 and it may contain markup that degrades the accuracy of linguistic analysis.

RLP provides language processors for detecting the encoding and MIME type of your input, for extracting plain text if the MIME type is not text/plain, and for converting the input to the required UTF-16 encoding. This chapter explains how to define contexts that perform the necessary conversions before performing the operations that require UTF-16 and plain text. See Preparing Marked Up Input [79] .

8.1. Preparing Plain Text

For input text that does not contain markup, you need to convert the text to UTF-16 in the correct byte order for your platform (unless it is already in that encoding).

RLP provides procedures for handling two basic categories of input text:

Category	**Description**
Any Encoding [77]	Plain text in any standard encoding.
UTF-16LE/BE (platform byte order) [78]	Plain Text in Unicode UTF-16LE or UTF-16BE encoding that conforms to the platform byte order.

8.1.1. Plain Text in Any Encoding

Any encoding means text with no markup in a Unicode or commonly used non-Unicode encoding. For the encodings that RLP can handle, see Unicode Converter [187] and RCLU Encodings [171] .

In the context configuration, use the Unicode Converter, RLI (Rosette Language Identifier), and RCLU (Rosette Code Library for Unicode) language processors (in that order) as the first three language processors. *Note: if you are using mime_detector, you should put it first.*

If the input is Unicode, Unicode Converter converts it to UTF-16 with the correct byte order for the current platform.

If the input is not Unicode, RLI [178] detects encoding (it also detects language), and the RCLU processor [168] converts the text to UTF-16 for use by subsequent language processors.

Here is a general purpose context that begins by identifying the encoding and converting the input to UTF-16:

```
<contextconfig>
 <languageprocessors>
  <languageprocessor>Unicode Converter </languageprocessor>
  <languageprocessor>RLI</languageprocessor>
  <languageprocessor>RCLU</languageprocessor>
  <!-- Other language processors, such as the following -->
  <languageprocessor>BL1</languageprocessor>
  <languageprocessor>JLA</languageprocessor>
```

```
   <languageprocessor>CLA</languageprocessor>
   <languageprocessor>KLA</languageprocessor>
   <languageprocessor>Tokenizer</languageprocessor>
   <languageprocessor>SentenceBoundaryDetector</languageprocessor>
   <languageprocessor>ARBL</languageprocessor>
  <languageprocessor>FABL</languageprocessor>
  <languageprocessor>URBL</languageprocessor>
   <languageprocessor>Stopwords</languageprocessor>
   <languageprocessor>BaseNounPhrase</languageprocessor>
   <languageprocessor>NamedEntityExtractor</languageprocessor>
   <languageprocessor>Gazetteer</languageprocessor>
   <languageprocessor>RegExpLP</languageprocessor>
   <languageprocessor>NERedactLP</languageprocessor>
 </languageprocessors>
</contextconfig>
```

Notes

If you know the encoding and the language, you can include these values as parameters when you process the text, in which case RLI is not required in the context.

If you know the input is Unicode, you can omit RLI and RCLU (keep RLI if you need to identify the language).

If you know the input is not Unicode, you can omit Unicode Converter.

8.1.2. Plain Text in UTF-16LE/BE

If your text is in UTF-16 in the platform byte order (e.g., UTF-16LE on an Intel x86 CPU or UTF-16BE on a Sun Sparc), you do not need to convert the text, unless you want to rely on the Unicode Converter to strip the BOM.

1. (Optional) Strip the byte order mark (BOM).

   ### Handling the BOM

   Unicode data may start with a byte order mark (BOM), the character code U+FEFF. UTF-16 and UTF-32 data require the BOM to indicate byte order. For UTF-8, byte order is not an issue, but UTF-8 data may include a BOM as an encoding signature.

   RLP does not use a BOM in its internal UTF-16 encoding. Byte order for the platform is known, and the BOM is unnecessary. If, however, your input is UTF-16 with a BOM and you do not pass it through Unicode Converter or RCLU, the BOM remains. If you have UTF-16 data with the correct byte order for your platform, but it includes a BOM, you probably want to strip the BOM (the first character) before you process the data.

2. In the context configuration, do not include a processor for converting the input to UTF-16:

```
<contextconfig>
 <languageprocessors>
   <!--Note: If you want to detect language, start with RLI. -->
   <languageprocessor>RLI</languageprocessor>
   <languageprocessor>BL1</languageprocessor>
   <languageprocessor>JLA</languageprocessor>
   <languageprocessor>CLA</languageprocessor>
   <languageprocessor>KLA</languageprocessor>
   <languageprocessor>Tokenizer</languageprocessor>
```

```
  <languageprocessor>SentenceBoundaryDetector</languageprocessor>
  <languageprocessor>ARBL</languageprocessor>
  <languageprocessor>FABL</languageprocessor>
  <languageprocessor>URBL</languageprocessor>
  <languageprocessor>Stopwords</languageprocessor>
  <languageprocessor>BaseNounPhrase</languageprocessor>
  <languageprocessor>NamedEntityExtractor</languageprocessor>
  <languageprocessor>Gazetteer</languageprocessor>
  <languageprocessor>RegExpLP</languageprocessor>
  <languageprocessor>NERedactLP</languageprocessor>
 </languageprocessors>
</contextconfig>
```

8.2. Preparing Marked-Up or Binary Input

Even if the input is encoded as Unicode, it may contain data other than pure text, such as HTML or XML markup tags. For linguistic processing, markup tags are extraneous text. The RLP language processors have been statistically trained with "normal" text from a variety of sources. For example, the Language Identifier (RLI) uses n-gram statistics to determine language. Markup tags degrade language detection. Part-of-speech tag disambiguation and Named Entity recognition also suffer from extraneous tag tokens. If you leave the tags in, RLP generates less accurate results.

PDF files and Microsoft Office documents use proprietary binary formats. The text content must be extracted from such a file before RLP can process it.

8.2.1. Using iFilter

Only on Windows

iFilter [147] is only available on Windows and requires an input file.

iFilter extracts plain text from text with markup and proprietary formats. The result is UTF-16 raw text from which all the extraneous data has been stripped. You can use iFilter to process files of the following MIME types:

MIME Type	File Type (standard file extensions)
text/plain	Text (txt, TXT)
text/html	HTML (html, htm, HTM, HTML)
text/xml	XML (xml, XML)
text/rtf	Rich Text Format (rtf, RTF)
application/pdf	Acrobat PDF (pdf, PDF)
application/msword	Microsoft Word (doc, DOC)
application/vnd.ms-excel	Microsoft Excel (xls, XLS)
application/vnd.ms-powerpoint	Microsoft Powerpoint (ppt, PPT)
application/vnd.ms-access	Microsoft Access (mdb, MDB)

You must provide iFilter a file pathname and a MIME type (from the table above). You can specify the MIME type when you call the API method for processing the input. You can also use the mime_detector [155] language processor to detect MIME type.

The following context is a general-purpose context for handling the MIME types listed above.

```
<contextconfig>
  <languageprocessors>
    <languageprocessor>mime_detector</languageprocessor>
    <languageprocessor>iFilter</languageprocessor>
    <languageprocessor>RLI</languageprocessor>
    <languageprocessor>BL1</languageprocessor>
     <languageprocessor>JLA</languageprocessor>
    <languageprocessor>CLA</languageprocessor>
    <languageprocessor>KLA</languageprocessor>
    <languageprocessor>Tokenizer</languageprocessor>
   <languageprocessor>SentenceBoundaryDetector</languageprocessor>
    <languageprocessor>ARBL</languageprocessor>
    <languageprocessor>FABL</languageprocessor>
    <languageprocessor>URBL</languageprocessor>
     <languageprocessor>Stopwords</languageprocessor>
    <languageprocessor>BaseNounPhrase</languageprocessor>
    <languageprocessor>NamedEntityExtractor</languageprocessor>
    <languageprocessor>Gazetteer</languageprocessor>
    <languageprocessor>RegExpLP</languageprocessor>
    <languageprocessor>NERedactLP</languageprocessor>
    </languageprocessors>
</contextconfig>
```

8.2.2. HTML Stripper

RLP includes another strategy for handling HTML data. This strategy is not limited to Windows, it can be used to process buffers as well as files, and it internally uses RLI and RCLU to identify the encoding and convert the input to UTF-16.

Here is a general purpose context that incorporates the HTML Stripper:

```
<contextconfig>
  <languageprocessors>
    <languageprocessor>HTML Stripper</languageprocessor>
    <languageprocessor>RLI</languageprocessor>
    <languageprocessor>BL1</languageprocessor>
    <languageprocessor>JLA</languageprocessor>
    <languageprocessor>CLA</languageprocessor>
    <languageprocessor>KLA</languageprocessor>
    <languageprocessor>Tokenizer</languageprocessor>
    <languageprocessor>SentenceBoundaryDetector</languageprocessor>
    <languageprocessor>ARBL</languageprocessor>
    <languageprocessor>FABL</languageprocessor>
    <languageprocessor>URBL</languageprocessor>
    <languageprocessor>Stopwords</languageprocessor>
    <languageprocessor>BaseNounPhrase</languageprocessor>
    <languageprocessor>NamedEntityExtractor</languageprocessor>
    <languageprocessor>Gazetteer</languageprocessor>
    <languageprocessor>RegExpLP</languageprocessor>
    <languageprocessor>NERedactLP</languageprocessor>
    </languageprocessors>
</contextconfig>
```

8.2.3. Handling XML Without iFilter

One simple way to remove the XML tags from an XML document (including XHTML) is to use an XSLT processor (such as Xalan or Saxon) to apply an XSL stylesheet to the document with output method set to text. Here is such a stylesheet that also strips out extraneous whitespace and generates UTF-8 plain text:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!--strip xml tags -->
  <xsl:output method="text" encoding="utf-8" indent="no"/>
  <!-- strip whitespace -->
  <xsl:strip-space elements="*"/>
</xsl:stylesheet>
```

8.3. Other Considerations

8.3.1. Normalizing Text

In many cases, Unicode allows a given string to be represented in multiple forms. This issue arises, for example, when representing a word that contains a character with an accent or diacritic mark. The issue also arises in Asian character sets, where some characters can be rendered in either a "half-width" or a "full-width" format, each with its own codepoint. In Japanese, for instance, both Katakana and Latin letters ("Romaji") can be written in both formats. In Chinese and Korean encodings, only Latin letters appear in both forms.

Normalization eliminates non-meaningful differences in the representation of Unicode strings, in the interest of simplifying linguistic processing, indexing, and other operations.

The Rosette Core Library for Unicode [168] (RCLU) provides a number of properties that you can set to normalize text. The Japanese Language Analyzer [147] (JLA) also includes a property that you can set to normalize Arabic numerals in Japanese text (see com.basistech.jla.normalize.result.token [150]).

8.3.2. File Size

While there is no set limit to the size of file that you can input into RLP, we do recommend that you process files less than 10 MB. Processing larger files may adversely affect performance. How RLP responds to large input files is difficult to predict as the response depends on the processors you specify, the operating system you use, and the hardware (particularly RAM) of the machine you run it on.

We recommend that you split very large files into smaller files that can be more easily processed.

Chapter 9. Accessing RLP Result Data

While processing a text stream, the RLP language processors generate a number of types of result objects. Processors post these results to internal storage making them available for subsequent processors, and for handling by the RLP application, which is the subject of this chapter.

When you use a context object to process input, RLP begins by posting the raw data, with any parameters you supply (such as language, encoding, filename, and MIME type) to internal storage. The first job of the processors is to generate UTF-16 raw text (stripped of extraneous data if necessary, such as HTML tags). Other language processors scan the raw text and generate other results types: tokens, part-of-speech tags, sentence boundaries, base noun phrases, named entities, and so on. Each processor in the context has access to the results that have been posted by earlier processors. The RLP context configuration, including property settings, and the settings in the processor options files influence the output that the processors generate. For more information, see RLP Processors [123] and Defining an RLP Context [18] .

9.1. Result Types

RLP defines integer constants for the result types.

The *Type* column in the following table provides the core type names that you can map to the appropriate C++ or Java constant. For C++ (`BT_RLP_` *TYPE*), see **bt_rlp_result_types.h**. For Java (`RLPConstants.` *TYPE*), see `com.basistech.rlp.RLPConstants`.

The *Data* column indicates the data type or structure for each result.

In addition to describing the type, the *Description* column indicates the processors that generate this result type.

Type	Data	Description
`ALTERNATIVE_LEMMAS`	Integer and UTF-16 string vector	Alternative LEMMA [85] results. For each token, provides the token index and the alternative lemmas. Generated by Arabic Base Linguistics [126] if `com.basistech.arbl.alternatives` and `com.basistech.arbl.lemmas` are set to `true`.
`ALTERNATIVE_NORM`	Integer and UTF-16 string vector	Alternative NORMALIZED_TOKEN [86] results. For each token, provides the token index and the alternative normalized tokens. Generated by Arabic Base Linguistics [126] if `com.basistech.arbl.alternatives` is set to `true`.
`ALTERNATIVE_PARTS_OF_SPEECH`	Integer and ASCII string vector	Alternative PART_OF_SPEECH [86] results. For each token, provides the token index and the alternative parts of speech. Generated by Arabic Base Linguistics [126] if `com.basistech.arbl.alternatives` is set to `true`.

Type	**Data**	**Description**
`ALTERNATIVE_ROOTS`	Integer and UTF-16 string vector	Alternative ROOTS [87] results. For each token, provides the token index and the alternative roots. Generated by Arabic Base Linguistics [126] if `com.basistech.arbl.alternatives` and `com.basistech.arbl.roots` are set to `true`.
`ALTERNATIVE_STEMS`	Integer and UTF-16 string vector	Alternative STEM [87] results. Provides the token index and the alternative stems. Generated by Arabic Base Linguistics [126] if `com.basistech.arbl.alternatives` is set to `true`.
`BASE_NOUN_PHRASE`	Integer pair	Index of first token and index + 1 of last token in the noun phrase. Generated by the Base Noun Phrase Detector [132] for Arabic, Chinese (Simplified and Traditional), Dutch, English, French, German, Italian, Japanese, and Spanish.
`COMPOUND`	Integer and UTF-16 string vector	Index to a token that represents a compound word and vector of components that make up the compound. Generated by Base Linguistics Language Analyzer [129] (for German, Dutch, and Hungarian); Chinese Language Analyzer [133] ; Japanese Language Analyzer [147] ; and Korean Language Analyzer [151] .
`DETECTED_ENCODING`	ASCII string	Name of the character encoding. Generated by Unicode Converter [187] and Rosette Language Identifier [178] . You may also pass this value to RLP when you process text, or use the RLP context object to set this value.
`DETECTED_LANGUAGE`	Integer	RLP-defined language code for the text. For C++, see **bt_language_names.h**. For Java, see `com.basistech.util.LanguageCode`. Generated by the Rosette Language Identifier [178] , which produces a single value (the first language found if the text contains multiple languages). You may also use the RLP context object to set this value programmatically. Reposted by the Chinese Script Converter [139] . If the text does contain multiple languages, use the Language Boundary Processor [152] to return LANGUAGE_REGION results (see below).

Type	**Data**	**Description**
`DETECTED_SCRIPT`	Integer	ISO15924 code for the writing script. For C++, see **bt_iso_15924_codes.h**. For Java, see `com.basistech.util.ISO15924`. Generated by the Rosette Language Identifier [178] . Reposted by the Chinese Script Converter [139] .
`LANGUAGE_REGION`	Six integers	Defines a language region: 1. Raw-text offset for start of region 2. Raw-text offset + 1 for end of region 3. Level: (currently not used) 4. Type: (currently not used) 5. Script (currently not used) 6. Language identifier (see DETECTED_LANGUAGE above) Generated by the Language Boundary Detector [152] .
`LEMMA`	UTF-16 string	Canonical form of a lexeme. In Arabic, a vocalized form useful for searches (a singular and plural noun, for example, share the same lemma, as does a verb in its perfect and imperfect tenses; the lemma for all domonstrative and relative pronouns is the masculine singular). Generated by Arabic Base Linguistics [126] if `com.basistech.arbl.lemmas` is set to `true`.
`MANY_TO_ONE_NORMALIZED_TOKEN`	UTF-16 string	Normalized form of a token as determined by the ManyToOneNormalizer [153] and a language-specific normalization dictionary. If the token does not appear in an appropriate normalization dictionary, the `MANY_TO_ONE_NORMALIZED_TOKEN` is identical to the TOKEN [88] . See also NORMALIZED_TOKEN [86] .
`MAP_OFFSETS`	Integer	Maps a character in normalized text to its location prior to normalization. Generated by Rosette Core Library for Unicode [168] when `com.basistech.rclu.mapoffsets` is true. (The default is false.) Useful when you need to manipulate the original input text while processing text that the Rosette Core Library for Unicode has normalized.

Type	**Data**	**Description**
`MIME_TYPE`	ASCII string	MIME type of input. Supplied by user or generated by mime_detector [155] . RLP recognizes the following MIME types: • text/plain • text/html • text/xml • text/rtf • application/pdf • application/msword • application/vnd.ms-excel • application/vnd.ms-powerpoint • application/ms.access
`NAMED_ENTITY`	Integer triple	Defines a named entity: 1. Index of first token in the named entity 2. Index + 1 of last token in the named entity 3. RLP-defined integer designating the source (1st byte), type (2nd and 3rd bytes), and optional subtype (4th byte) of the entity. The source designates one of three processors: Named Entity Extractor, Regular Expressions Processor, or Gazetteer. RLP maps types and subtypes to strings. For C++, see **bt_rlp_ne_types.h**. For Java, see `com.basistech.rlp.RLPNENameMap` Strings for types and subtypes are defined in ***`BT_ROOT`*/rlp/ etc/ne-types.xml**, and may also be defined in gazetteers and in the configuration file for regular expressions. See Customizing Named Entities [52] . Generated by the Named Entity Extractor [156] , the Regular Expression processor [163] , the the Gazetteer [144] processor, and the Named Entity Redactor [160] , which should be included with any of the preceding processors.
`NORMALIZED_TOKEN`	UTF-16 string	The normalized form for the token. Generated byArabic Base Linguistics [126] , Farsi Base Linguistics [141] , and Urdu Base Linguistics [188] processors. See also MANY_TO_ONE_NORMALIZED_TOKEN [85] .
`PART_OF_SPEECH`	ASCII string	Part of speech for the token. See Part of Speech Tags [209] . Generated by Base Linguistics Language Analyzer [129] , Arabic Base Linguistics [126] , Chinese Language Analyzer [133] , Japanese Language Analyzer [147] , and Korean Language Analyzer [151] .

Type	**Data**	**Description**
`RAW_TEXT`	UTF-16 string	The input text. Generated by the Unicode Converter [187] or Rosette Core Library for Unicode [168] , or instantiated from the input by a context process method that takes UTF-16 as input (see Processing UTF-16 Input [27]). Reposted by the Chinese Script Converter [139] .
`READING`	Integer and UTF-16 string vector	Alternate readings (transcriptions) for a token. Provides the index to the token and the alternate readings. Generated by the Chinese Language Analyzer [133] and Japanese Language Analyzer [147] .
`ROOTS`	UTF-16 string	For Semitic languages, the root for the stem or normalized token. Generated by Arabic Base Linguistics [126] , provided (1) **arbl-options.xml** includes a valid `rootpath` entry, and (2) `com.basistech.arbl.roots` is set to `true`.
`SCRIPT_REGION`	Integer triple	Defines a script region: 1. Raw-text offset for start of region 2. Raw-text offset+ 1 for end of region 3. ISO15924 script identifier RLP provides utilities for mapping ISO15924 integer values to 4-character codes and English names. For C++, see **bt_iso15924.h**. For Java, see `com.basistech.util.ISO15924`. Generated by the Script Boundary Detector [184] .
`SENTENCE_BOUNDARY`	Integer	Index of last token + 1 for the sentence. The value for the previous sentence (0 for the first sentence) is the index of the first token in the sentence. Generated by Base Linguistics Language Analyzer [129] and Sentence Boundary Detector [183] (in conjunction with Tokenizer or the appropriate language processor).
`STEM`	UTF-16 string	The dictionary form for the token. Generated by the Base Linguistics Language Analyzer [129] , Arabic Base Linguistics [126] , Chinese Language Analyzer [133] , Korean Language Analyzer [151] , Japanese Language Analyzer [147] , Farsi Base Linguistics [141] , and Urdu Base Linguistics [188] .

Type	**Data**	**Description**
`STOPWORD`	Integer	Index for a token that is a stopword. Generated by the Chinese Language Analyzer [133] , the Japanese Language Analyzer [147] , the Korean Language Analyzer [151] , and the Stopwords processor [185] using the language-specific stopword dictionary specified in the processor configuration file: **cla-options.xml**, **jla-options.xml**, **kla-options.xml** (specifies directory containing dictionaries), or **stop-options.xml**.
`TEXT_BOUNDARIES`	Integer	Raw-text offset + 1 of a sentence-level text boundary. Generated by the Text Boundary Detector [186] , which uses Unicode rules (UAX 29 [http://www.unicode.org/reports/tr29/]) for determining sentence boundaries. If you have identified the language, you can use the Sentence Boundary Detector [183] to generate SENTENCE_BOUNDARY results (see above).
`TOKEN`	UTF-16 string	An atomic element from the input text, such as word, number, multiword expression, possessive affix, or punctuation. Generated by Chinese Language Analyzer [133] , Chinese Script Converter (CSC) [139] , Base Linguistics Language Analyzer [129] , Japanese Language Analyzer [147] , Korean Language Analyzer [151] , Tokenizer [187] (for use primarily by Arabic Base Linguistics [126] , Farsi Base Linguistics [141] , and Urdu Base Linguistics [188]). SentenceBoundary Detector [183] and Chinese Script Converter [139] may update and repost the TOKEN results.
`TOKEN_OFFSET`	Integer pair	Raw-text start and end + 1 offsets for the token. Generated by Base Linguistics Language Analyzer [129] , Chinese Language Analyzer [133] , Japanese Language Analyzer [147] , Korean Language Analyzer [151] , Sentence Boundary Detector [183] , Tokenizer [187] , and Chinese Script Converter [139] (which may update and repost the TOKEN results).
`TOKEN_SOURCE_ID`	Integer	Identifies the dictionary in which the token was found. Generated by Japanese Language Analyzer [147] .

Type	Data	Description
`TOKEN_SOURCE_NAME`	UTF-16 string	Name of the dictionary identified by `TOKEN_SOURCE_ID`. For the system dictionary and any user dictionary that you have not named or compiled, the name is "" (empty string). For information about assigning a name to a user dictionary, see Creating the source file for a Japanese Language Analyzer User Dictionary [200] . Generated by Japanese Language Analyzer [147] .
`TOKEN_VARIATIONS`	Integer and UTF-16 string vector	Variant orthographic representations of a token (word). Provides the token index and a vector of variant representations. Useful for producing search strings. Generated by Arabic Base Linguistics [126] if `com.basistech.arbl.variations` is set to true, and Farsi Base Linguistics [141] if `com.basistech.fabl.variations` is set to true.
`TRANSCRIBED_TEXT`	UTF-16 string	Currently unused.

9.2. Handling RLP Results in C++

After using a context object to process a text file or buffer, use the following to access the results that have been generated.

- A token iterator [89]

  Provides access to tokens and several related result types. For access to multiple result types returned by a token iterator, use a token iterator instead of a set of result iterators.

- One or more result iterators [92]

  Provides access to all result types, but you need a separate iterator for each type. Use a result iterator to get a single result type; use multiple result iterators to get multiple result types not available with the token iterator.

- A named entity iterator [97]

  Provides access to named entities.

- The context object [98]

  Provides access to single-valued result types and avoids the overhead of creating a result iterator to perform a single iteration.

9.2.1. Using a Token Iterator

`BT_RLP_TokenIterator` coordinates access to a number of result objects associated with each token. For each iteration, you can get token, stem, normalized token, part of speech, and offset. For compound tokens (in German, Dutch, Hungarian, Korean, Japanese, or Chinese, for example) you can also get the individual components of the compound. You can also set the iterator to give alternative readings (transcriptions) for each token, provided the language processor supports readings. For Arabic tokens, you can get lemmas, alternative lemmas, alternative normalized tokens, alternatve roots, alternative stems, and alternative parts of speech.

For descriptions of the result types, see Result types [83] .

Result type	Data you can get with a token iterator
`BT_RLP_TOKEN`	The token and its index. Use `GetToken()`.
`BT_RLP_TOKEN_OFFSET`	The raw-text offsets (start and end + 1) for the token. Use `GetStartOffset()` and `GetEndOffset()`.
`BT_RLP_PART_OF_SPEECH`	The part of speech for the token. Use `GetPartOfSpeech()`.
`BT_RLP_STEM`	The stem [87] (dictionary form) for the token. Use `GetStemForm()`.
`BT_RLP_MANY_TO_ONE_NORMALIZED_TOKEN`	The many-to-one normalized form of a token [85] . Use `GetManyToOneNormalForm()`
`BT_RLP_NORMALIZED_TOKEN`	The normalized form for the token. Use `GetNormalForm()`.
`BT_RLP_STOPWORD`	Whether or not the token is a stopword. Use `IsStopword()`.
`BT_RLP_COMPOUND`	The number of components that make up the compound and the value of each. To get this information, set the token iterator factory accordingly (`SetReturnCompoundComponents(true)`). Use `GetNumberOfCompoundComponents()` and `GetCompoundComponent(BT_UInt32 index)`.
`BT_RLP_READING`	The number of readings and the value for each. To get this data, set the token iterator factory accordingly (`SetReturnReadings(true)`) and use `GetNumberOfReadings()` and `GetReading(BT_UInt32 index)`.
`BT_RLP_TOKEN_SOURCE_ID`	Identifier for the dictionary in which the token was found. To get this data (for Japanese input only), set `com.basistech.jla.generate_token_sources` to true and use `GetSourceId()`
`BT_RLP_TOKEN_SOURCE_NAME`	Name of the dictionary identified by `BT_RLP_TOKEN_SOURCE_ID`. To get this data (for Japanese input only), set `com.basistech.jla.generate_token_sources` to true and use `GetSourceName(BT_UInt32 id)`
`BT_RLP_LEMMA`	Lemma for the token. To get this data (for Arabic input only), set `com.basistech.arbl.lemmas` to true and use `GetLemmaForm()`

Result type	Data you can get with a token iterator
`BT_RLP_ALTERNATIVE_LEMMAS`	Alternative lemmas for the token. To get this data (for Arabic input only), set `com.basistech.arbl.lemmas` and `com.basistech.arbl.alternatives` to true, use `GetNextAnalysis()` to step through the analyses, and use `GetLemmaForm()` to get the lemma for each analysis
`BT_RLP_ALTERNATIVE_NORM`	Alternative normalized tokens for the token. To get this data (for Arabic input only), set `com.basistech.arbl.alternatives` to true, use `GetNextAnalysis()` to step through the analyses, and use `GetNormalForm()` to get the normalized token for each analysis
`BT_RLP_ROOTS`	Root for the token. To get this data (Arabic input only), set `com.basistech.arbl.roots` to true, and use `GetRootForm()`.
`BT_RLP_ALTERNATIVE_ROOTS`	Alternative roots for the token. To get this data (for Arabic input only), set `com.basistech.arbl.roots` and `com.basistech.arbl.alternatives` to true, use `GetNextAnalysis()` to step through the analyses, and use `GetRootForm()` to get the root for each analysis
`BT_RLP_ALTERNATIVE_STEMS`	Alternative stems for the token. To get this data (for Arabic input only), set `com.basistech.arbl.alternatives` to true and use `GetNextAnalysis()` to step through the analyses, and `GetStemForm()` to get the stems for each analysis
`BT_RLP_ALTERNATIVE_PARTS_OF_SPEECH`	Alternative parts of speech for the token. To get this data (for Arabic input only), set `com.basistech.arbl.alternatives` to true and use `GetNextAnalysis()` to step through the analyses, and `GetPartOfSpeech()` to get the part of speech for each analysis

Procedure 9.1. How to Use a Token Iterator

1. Create a token iterator factory.

```
BT_RLP_TokenIteratorFactory* factory =
                        BT_RLP_TokenIteratorFactory::Create();
```

2. To decompose compound tokens or access readings (and the language processor supports the operation), set the factory object accordingly.

```
//Provides access to readings.
factory->SetReturnReadings(true);
```

```
//Provides access to the components of compound tokens.
factory->SetReturnCompoundComponents(true);
```

3. Use the factory object to create a token iterator for the context which processed the text.

```
// context points to the BT_RLP_Context object used to process the text
BT_RLP_TokenIterator* iter = factory->CreateIterator(context);
```

4. Destroy the token iterator factory.

```
factory->Destroy();
```

5. Use the token iterator to access each token and obtain the data of interest.

```
while (iter->Next()) {
 //Get the data you want for each token.

 //Gets the token (BT_RLP_TOKEN).
 const BT_Char16* token = iter->GetToken();

  //Gets the token index.
 BT_UInt32 index = iter->GetIndex();

 //Gets the part of speech (BT_RLP_PART_OF_SPEECH) for the token.
 const char* pos = iter->GetPartOfSpeech();

 //Get alternative analyses (Arabic text only).
 if (iter->GetNumberOfAnalyses > 0 ) {
   const BT_Char16* altLemma;
   const BT_Char16* altRoot;
   while (iter->GetNextAnalysis) {
     altLemma = iter->GetLemmaForm();
     altRoot = iter->GetRootForm();
     //Handle altLemma and altRoot...
   }
 }
   //And so on.
}
```

6. Destroy the token iterator.

```
iter->Destroy();
```

9.2.2. Using a Result Iterator

You can use a `BT_RLP_ResultIterator` object to iterate through the results of a specified type. If you are only interested in a single type, or if you want results of a type not returned by the token iterator, use a result iterator. To get the results of more than one type, you must use a separate result iterator for each type.

As you iterate through the results of a given type, the result iterator returns a pointer to each result object. The structure of the data associated with the result object is determined by the result type. See Result data structures [93] for the data structure associated with each result type and the `BT_RLP_Result` method for accessing the data.

Procedure 9.2. How to Use a Result Iterator

1. Use the context object to create a result iterator for the desired result type.

```
//For example, create a result iterator to get base noun phrases.
BT_RLP_ResultIterator* iter =
    context->GetResultIterator(BT_RLP_BASE_NOUN_PHRASE);
```

2. Use the result iterator to access each result object, and the result object to access the data.

```
const BT_RLP_Result* bnp;
while ((bnp = iter->Next()) != NULL){
  //The result is a pair of integers, indexes of the first token and last
  //token + 1 in the base noun phrase.
  BT_UInt32 first, last;
  bnp->AsIntegerPair(first, last);
  //Use the result...
}
```

3. Destroy the result iterator.

```
context->DestroyResultIterator(iter);
```

9.2.2.1. Result Data Structures

The following table maps each result type to a result data structure. As indicated below, the `BT_RLP_Result` class provides a method for accessing each data structure.

For descriptions of the result types, see Result types [83] .

Result type	Data structure
`BT_RLP_TOKEN`	Null-terminated UTF-16 string [94]
`BT_RLP_TOKEN_SOURCE_NAME`	
`BT_RLP_STEM`	
`BT_RLP_MANY_TO_ONE_NORMALIZED_TOKEN`	
`BT_RLP_NORMALIZED_TOKEN`	
`BT_RLP_ROOTS`	
`BT_RLP_GAZETEER_NAMES`	
`BT_RLP_LEMMA`	
`BT_RLP_RAW_TEXT`	Non-null-terminated UTF-16 string and string length [94]
`BT_RLP_TRANSCRIBED_TEXT`	
`BT_RLP_PART_OF_SPEECH`	String of 8-bit chars [94]
`BT_RLP_DETECTED_ENCODING`	
`BT_RLP_MIME_TYPE`	
`BT_RLP_DETECTED_LANGUAGE`	Integer [95]
`BT_RLP_DETECTED_SCRIPT`	
`BT_RLP_STOPWORD`	
`BT_RLP_SENTENCE_BOUNDARY`	
`BT_RLP_TEXT_BOUNDARIES`	
`BT_RLP_MAP_OFFSETS`	
`BT_RLP_SOURCE_ID`	

Result type	Data structure
BT_RLP_COMPOUND	Integer and vector of UTF-16 strings [95]
BT_RLP_ALTERNATIVE_LEMMAS	
BT_RLP_ALTERNATIVE_NORM	
BT_RLP_ALTERNATIVE_ROOTS	
BT_RLP_ALTERNATIVE_STEMS	
BT_RLP_READING	
BT_RLP_TOKEN_VARIATIONS	
BT_RLP_ALTERNATIVE_PARTS_OF_SPEECH	Integer and vector of ASCII strings [95]
BT_RLP_TOKEN_OFFSET	Integer pair [96]
BT_RLP_BASE_NOUN_PHRASE	
BT_RLP_NAMED_ENTITYY	Integer triple [96]
BT_RLP_SCRIPT_REGION	
BT_RLP_LANGUAGE_REGION	Integer vector [96]

9.2.2.2. Null-terminated UTF-16 String

The `BT_RLP_Result` method

```
const BT_Char16* AsUTF16String()
```

returns a null-terminated UTF16 string for the following result types:

- BT_RLP_TOKEN
- BT_RLP_TOKEN_SOURCE_NAME
- BT_RLP_STEM
- BT_RLP_LEMMA
- BT_RLP_NORMALIZED_TOKEN
- BT_RLP_MANY_TO_ONE_NORMALIZED_TOKEN
- BT_RLP_ROOTS

9.2.2.3. Non-Null-Terminated UTF-16 String and String Length

The `BT_RLP_Result` method

```
const BT_Char16* AsCountedUTF16String(BT_UInt32 &length)
```

returns a UTF-16 string that is not null terminated for each of the following result types. The `length` parameter returns the length of the string. A single iteration returns all the data. Alternatively, use the context object [98] to return these single-valued types:

- BT_RLP_RAW_TEXT
- BT_RLP_TRANSCRIBED_TEXT

9.2.2.4. Null-Terminated String of 8-Bit Characters

The `BT_RLP_Result` method

```
const BT_Char8* AsString()
```

returns a null-terminated string of 8-bit chars for the following result types:

- `BT_RLP_PART_OF_SPEECH`
- `BT_RLP_DETECTED_ENCODING`
- `BT_RLP_MIME_TYPE`

9.2.2.5. Integer

The `BT_RLP_Result` method

```
BTUInt32 AsUnsignedInteger()
```

returns a 32-bit unsigned integer for the following result types:

- `BT_RLP_DETECTED_LANGUAGE`
- `BT_RLP_DETECTED_SCRIPT`
- `BT_RLP_STOPWORD`
- `BT_RLP_SENTENCE_BOUNDARY`
- `BT_RLP_TEXT_BOUNDARIES`
- `BT_RLP_MAP_OFFSETS`
- `BT_RLP_TOKEN_SOURCE_ID`

9.2.2.6. Integer and Vector of UTF-16 Strings

The `BT_RLP_Result` method

```
void AsIntegerUTF16StringVectorPair(BT_Uint32 &index,
                                    BT_RLP_Result_UTF16StringVector const  *&strings)
```

returns the index to the token to which the result applies and a vector of UTF-16 strings for the following types:

- `BT_RLP_COMPOUND`
- `BT_RLP_READING`
- `BT_RLP_TOKEN_VARIATIONS`
- `BT_RLP_ALTERNATIVE_LEMMAS`
- `BT_RLP_ALTERNATIVE_NORM`
- `BT_RLP_ALTERNATIVE_ROOTS`
- `BT_RLP_ALTERNATIVE_STEMS`

`BT_RLP_Result_UTF16StringVector` provides access to the size of the vector

```
BT_UInt32 BT_RLP_Result_UTF16StringVector::Size()
```

and a pointer to the specified string in the vector

```
BT_Char16 const* BT_RLP_Result_UTF16StringVector::Get(BT_UInt32  index)
```

9.2.2.7. Integer and Vector of ASCII Strings

The `BT_RLP_Result` method

```
void AsIntegerStringVectorPair(BT_Uint32 &index,
                               BT_RLP_Result_StringVector const *&strings)
```

returns the index to the token to which the result applies and a vector of ASCII strings for the following type:

- `BT_RLP_ALTERNATIVE_PARTS_OF_SPEECH`

9.2.2.8. Integer Pair

The `BT_RLP_Result` method

```
void AsIntegerPair(BTUInt32 &a, BTUInt32 &b)
```

Returns a pair of 32-bit unsigned integers for the following result types:

- `BT_RLP_TOKEN_OFFSET`

  `a` is the raw-text offset of the start of the token.

  `b` is the raw-text offset + 1 of the end of the token.

- `BT_RLP_BASE_NOUN_PHRASE`

  `a` is the index of the first token in the noun phrase.

  `b` is the index + 1 of the last token in the noun phrase.

9.2.2.9. Integer Triple

The `BT_RLP_Result` method

```
void AsIntegerTriple(BTUInt32 &a, BTUInt32 &b, BTUInt32 &c)
```

returns three 32-bit unsigned integers for the following result types:

- `BT_RLP_NAMED_ENTITY` [1]

  `a` is the index of the first token in the named entity.

  `b` is the index + 1 of the last token in the named entity.

  `c` identifies the named entity type.

- `BT_RLP_SCRIPT_REGION`

  `a` is the raw-text offset for the start of the script region.

  `b` is the raw-text offset + 1 for the end of the script region.

  `c` identifies the script.

9.2.2.10. Integer Vector

The `BT_RLP_Result` method

```
void AsUnsignedIntegerVector(BTUInt32* vector, BTUInt32 size)
```

returns a vector of unsigned 32-bit integers for the following result type:

- `BT_RLP_LANGUAGE_REGION`

  The vector contains six integers: start, end, level, type, script (not used), and language.

[1] See also Named Entity Iterator [97] .

9.2.3. Using the Named Entity Iterator

You can use the `BT_RLP_NE_Iterator` to iterate through NAMED_ENTITY [86] results generated by Named Entity Extractor [156] , the Regular Expression processor [163] , the Gazetteer [144] processor, and the Named Entity Redactor [160] .

Named Entity Iterator or Result Iterator? As described in the previous section, you can use the result iterator [92] to iterate through named entities. The named entity iterator simplifies access to named entities and provides some additional control over the data you can collect. If normalized tokens [86] are available, the named entity iterator provides direct access to the normalized tokens in the named entities. You can also instruct the iterator to strip affixes (prefixes and suffixes) from these tokens. These features are useful in applications designed to generate query strings.

Note: Currently, RLP generates normalized tokens and supports affix stripping for Arabic only. You can set Arabic Base Linguistics [126] to generate normalized tokens.

Procedure 9.3. How to Use the Named Entity Iterator

1. Create a Named Entity Iterator Factory.

```
BT_RLP_NE_Iterator_Factory*  factory = BT_RLP_NE_Iterator_Factory::Create();
```

2. If you want to strip prefixes and suffixes from the tokens in the named entities, set the factory `StripAffixes` flag to `true`. Currently, the stripping of affixes only applies to Arabic.

```
//Strip prefixes and suffixes.
factory->setStripAffixes(true);
```

3. Use the factory object to create the iterator for the context with which you have processed the text.

```
BT_RLP_NE_Iterator ne_iter = factory->CreateIterator(context);
```

4. Destroy the factory

```
factory->Destroy();
```

5. Use the Named Entity Iterator to iterate over the named entities in the context and get data of interest.

```
while (ne_iter->Next()) {
 //Get the data you want for each named entity.

 //Get the number of tokens in the named entity.
 BT_UInt32 size = ne_iter->Size();

 //Get the named entity as it appeared in the text.
 const BT_Char16* ne_raw = ne_iter->GetRawNamedEntity();

 //Get the normalized named entity.
 //  * Whitespace between tokens is normalized to a single space.
 //  * Uses normalized tokens if available; otherwise tokens.
 const BT_Char16* ne_normal = ne_iter->GetNamedEntity();

 //Get the named entity type (an integer), and its string representation
 //("PERSON", "LOCATION", "ORGANIZATION", ...).
 BT_UInt32 type = ne_iter->GetType();
 const char* type_name = BT_RLP_NET_ID_TO_STRING(type);

 //Get the token offsets for the start and end of the named entity.
BT_UInt32 start_token_offset = ne_iter->GetStartOffset();
BT_UInt32 end_token_offset = ne_iter->GetEndOffset();

//and so on ...
}
```

6. Destroy the Named Entity Iterator

```
ne_iter->Destroy();
```

9.2.4. Getting Results from the Context Object

You can use the `BT_RLP_Context` object to get single-valued results, that is results that do not require an iterator. `BT_RLP_Context` also provides access to integer arrays, where each integer is a single result.

Single-valued Results

- Use

  ```
  BT_Char8 const *GetStringResult(BT_RLP_EntityType type)
  ```

  where `type` is `BT_RLP_DETECTED_ENCODING`. The result is an ASCII string.

- Use

  ```
  BTUInt32 GetIntegerResult(BT_RLP_EntityType type)
  ```

  where `type` is `BT_RLP_DETECTED_LANGUAGE` or `BT_RLP_DETECTED_SCRIPT`. The result is an unsigned integer.

- Use

  ```
  BT_Char16 const *GetUTF16StringResult(BT_RLP_EntityType type,
                                         BT_UInt32& resultLength)
  ```

  where `type` is `BT_RLP_RAW_TEXT` or `BT_RLP_TRANSCRIBED_TEXT`, and `resultLength` returns the length of the string. The result is a UTF-16 non-null-terminated string.

Multi-Valued Integer Results

- Provides support for getting an array of unsigned integers for sentence or text boundaries.

  Use

  ```
  BT_UInt32 const *GetUnsignedIntegerArrayResults(BT_RLP_EntityType type,
                                                   BT_UInt32& count)
  ```

  where `type` is `BT_RLP_SENTENCE_BOUNDARY` or `BT_RLP_TEXT_BOUNDARIES` and `count` returns the size of the array of integers. The result is an array of integers.

9.3. Handling RLP Results in Java

`RLPResultAccess` [99] provides access to all the RLP result types.

`RLPResultRandomAccess` [102] is deprecated.

9.3.1. Using RLPResultAccess

Depending on the result type, call one of four methods:

- `getListResult()` returns a list of Strings, Integers, or int arrays.
- `getMapResult()` returns a sorted set of Map entries. For each entry, the key is an Integer token index and the value is an array of Strings.
- `getStringResult()` returns a single String.
- `getIntegerResult()` returns a single Integer.

Procedure 9.4. How to Use RLPResultAccess

After you have used a context object to process text:

1. Instantiate an `RLPResultAccess` object.

```
//context is the RLPContext object used to process the text.
RLPResultAccess resultAccess = new RLPResultAccess(context);
```

2. Use the appropriate methods to retrieve the result data you want. For lists and maps, use the standard `java.util` API to access individual results.

```
//For example, get a list of tokens, and a list of noun phrases.
//Each noun phrase object is int[2] with indexes that indicate
//the range of tokens that make up the noun phrase.
List tokenList = resultAccess.getListResult(RLPConstants.TOKEN);
List bnpList = resultAccess.getListResult(RLPConstants.BASE_NOUN_PHRASE)
Iterator iter = bnpList.iterator();
while (iter.hasNext()){
  // Start and end+1 token indexes for noun phrase.
  int[] pair = (int[])iter.next();
  int start = pair[0];
  int end = pair[1];
  StringBuffer nounPhrase = new StringBuffer();
  //Assemble the noun phrase.
  for (int i = start; i < end; i++){
    if (i < start)
      nounPhrase.append(" ");
    nounPhrase.append((String)tokenList.get(i));
  }
  //Handle the noun phrase.
  System.out.println("Noun Phrase: " + nounPhrase.toString());
}

//If the language contains (and you are using a processor that supports)
//compound words, you can get a Map of compounds. Each Integer key
//is the index of the corresponding token, and the value is an array of
//the Strings that make up the compound.
Map compoundMap = resultAccess.getMapResult(RLPConstants.COMPOUND);
iter = map.keySet().iterator();
while (iter.hasNext()){
  Integer key = (Integer)iter.next();
  //Can use the key to get the associated token.
  String token = tokenList.get(key.intValue());
  String[] value = (String[])map.get(key);
  //Handle the compound...
}

//Singleton Integer and String results.
Integer langId =
        resultAccess.getIntegerResult(RLPConstants.DETECTED_LANGUAGE);
// Use langID int value to get the ISO639 2-letter language code.
String language = BTLanguageCodes.ISO639FromLanguageID(langId.intValue());

String mimeCharset =
        resultAccess.getStringResult(RLPConstants.DETECTED_ENCODING);
```

9.3.1.1. RLP Result Types and RLPResultAccess Methods

Result type	RLPResultAccess method
RLPConstants.TOKEN	getListResult() [101]
RLPConstants.PART_OF_SPEECH	
RLPConstants.STEM	
RLPConstants.MANY_TO_ONE_NORMALIZED_TOKEN	
RLPConstants.NORMALIZED_TOKEN	
RLPConstants.ROOTS	
RLPConstants.LEMMA	
RLPConstants.GAZETEER_NAMES	
RLPConstants.STOPWORD	
RLPConstants.SENTENCE_BOUNDARY	
RLPConstants.TEXT_BOUNDARIES	
RLPConstants.TOKEN_OFFSET	
RLPConstants.BASE_NOUN_PHRASE	
RLPConstants.NAMED_ENTITY	
RLPConstants.SCRIPT_REGION	
RLPConstants.LANGUAGE_REGION	
RLPConstants.MAP_OFFSETS	
RLPConstants.TOKEN_SOURCE_ID	
RLPConstants.TOKEN_SOURCE_NAME	
RLPConstants.COMPOUND	getMapResult() [102]
RLPConstants.READING	
RLPConstants.TOKEN_VARIATIONS	
RLPConstnats.ALTERNATIVE_LEMMA	
RLPConstants.ALTERNATIVE_NORM	
RLPConstants.ALTERNATIVE_PARTS_OF_SPEECH	
RLPConstants.ALTERNATIVE_ROOTS	
RLPConstants.ALTERNATIVE_STEMS	

Result type	RLPResultAccess method
RLPConstants.DETECTED_LANGUAGE	getIntegerResult() [102]
RLPConstants.DETECTED_SCRIPT	
RLPConstants.DETECTED_ENCODING	getStringResult() [102]
RLPConstants.RAW_TEXT	
RLPConstants.TRANSCRIBED_TEXT	
RLPConstants.MIME_TYPE	

9.3.1.2. RLPResultAccess getListResult() Method

```
public List getListResult(int type)
```

returns a list of objects as follows:

type	Object	Comment
RLPConstants.TOKEN	String	A token [88]
RLPConstants.PART_OF_SPEECH	String	A POS tag [86]
RLPConstants.STEM	String	Dictionary form of token [87]
RLPConstants.LEMMA	String	Lemma of token [85]
RLPConstants.MANY_TO_ONE_NORMALIZED_TOKEN	String	A many-to-one normalized token [85]
RLPConstants.NORMALIZED_TOKEN	String	A normalized token [86]
RLPConstants.LEMMA	String	Lemma of token [85]
RLPConstants.ROOTS	String	Semitic root of word [87]
RLPConstants.TOKEN_SOURCE_NAME	String	(Japanese only) Name of the dictionary associated with the TOKEN_SOURCE_ID [89]
RLPConstants.STOPWORD	Integer	A stopword [88]
RLPConstants.SENTENCE_BOUNDARY	Integer	Token index of sentence boundary (last token + 1 for the sentence). [87]
RLPConstants.TEXT_BOUNDARIES	Integer	Offset for sentence-level text boundary (offset + 1 of end of sentence) [88]
RLPConstants.MAP_OFFSETS	Integer	Maps character in normalized text to its location in the pre-normalized text. [85]
RLPConstants.TOKEN_SOURCE_ID	Integer	(Japanese only) Identifies the dictionary in which the token was found [88]
RLPConstants.TOKEN_OFFSET	int[2]	Pair of token offsets [88]
RLPConstants.BASE_NOUN_PHRASE	int[2]	Token indexes delimiting a noun phrase [84]
RLPConstants.NAMED_ENTITY	int[3]	Token indexes delimiting a named entity and entity type [86] (see also getNamedEntityData [103])

type	Object	Comment
RLPConstants.SCRIPT_REGION	int[3]	Offsets and ID for a script region [87]
RLPConstants.LANGUAGE_REGION	int[6]	Offsets (2), level, type, script (not used), and language ID for a language region [85]

9.3.1.3. RLPResultAccess getMapResult() Method

```
public Map getMapResult(int type)
```

returns a sorted set of map entries where the key for each entry is an Integer index to the associated token and the value is `String[]` (compound components, readings, or token alternatives). Use this method to retrieve results of the following `type`:

- `RLPConstants.COMPOUND`
- `RLPConstants.READING`
- `RLPConstants.TOKEN_VARIATIONS`
- `RLPConstants.ALTERNATIVE_LEMMAS`
- `RLPConstants.ALTERNATIVE_NORM`
- `RLPConstants.ALTERNATIVE_PARTS_OF_SPEECH`
- `RLPConstants.ALTERNATIVE_ROOTS`
- `RLPConstants.ALTERNATIVE_STEMS`

9.3.1.4. RLPResultAccess getIntegerResult() Method

```
public Integer getIntegerResult(int type)
```

returns a single Integer for the following result types:

- `RLPConstants.DETECTED_LANGUAGE`
- `RLPConstants.DETECTED_SCRIPT`

9.3.1.5. RLPResultAccess getStringResult() Method

```
public String getStringResult(int type)
```

returns a single String for the following result types:

- `RLPConstants.DETECTED_ENCODING`
- `RLPConstants.RAW_TEXT`
- `RLPConstants.TRANSCRIBED_TEXT`
- `RLPConstants.MIME_TYPE`

9.3.2. RLPResultRandomAccess

The result random access object provides a single call that returns the entire result set for the specified result type. Depending on type, you must cast the return value accordingly.

For most usage patterns, we recommend you use `RLPResultAccess`. For more information about this lower-level API, see the *Javadoc*.

9.3.2.1. getNamedEntityData

You can use the `getNamedEntityData()` method to collect data about NAMED_ENTITY [86] results generated by Named Entity Extractor [156] , the Regular Expression processor [163] , the Gazetteer [144] processor, and the Named Entity Redactor [160] .

getNamedEntityData() or getListResult(). As described in the previous section, you can use the getListResult() [101] to obtain the delimiting token indexes and entity type for each named entity. `getNamedEntityData()` simplifies access to named entities and provides some additional control over the data you can collect. If normalized tokens [86] are available, the named entity iterator provides direct access to the normalized tokens in the named entities. You can also instruct the iterator to strip affixes (prefixes and suffixes) from these tokens. These features are useful in applications designed to generate query strings.

Note: Currently, RLP generates normalized tokens and supports affix stripping for Arabic only. You can set Arabic Base Linguistics [126] to generate normalized tokens.

Procedure 9.5. How to Get Named Entity Data

1. Instantiate a `RLPResultRandomAccess` object

```
//context is the RLPContext object used to process the data.
RLPResultRandomAccess resultRA = new RLPResultRandomAccess(context);
```

2. Get an array of `NamedEntityData` objects (one for each named entity).

```
//If the text is Arabic and you have configured the processor to return normalized tokens,
//you can set stripAffixes to true to remove particle prefixes and suffixes from the
//named entity tokens.
boolean stripAffixes = true;
NamedEntityData neData = resultRA.getNamedEntityData(stripAffixes);
```

3. Used `NamedEntityData` methods to obtain the data of interest.

```
//For example, get normalized named entities and the String representation
//of their entity types.
for (int i = 0; i < neData.length; i++){
  //For all languages, normalized named entities contain a single space between tokens.
  //For Arabic text, the tokens are normalized tokens, if Arabic Base Linguistics is
  //configured to return normalized tokens.
  //To get the named entitites as they appear in the source text, use getRawNamedEntity().
 String normalizedNE = neData[i].getNormalizedNamedEntity();
 String typeName   = neData[i].toString();
  //Handle the named entity.
  out.println("Normalized Named entity (" + typeName + "): " + normalizedNE);
}
```

Chapter 10. RLP Runtime Configuration

This chapter describes how to configure RLP to run with other applications or to be redistributed with other applications.

10.1. Redistribution

When delivering your application to your customers, you may wish to include just a subset of RLP, either to reduce your distribution size or to improve performance. This section explains how to pare off unwanted processors, move portions of RLP into your own directory structure, and reconfigure your context(s).

To prepare to redistribute RLP with your application, do the following tasks:

1. Environment Configuration.
2. Context configuration; choose the processors you wish to distribute.
3. Alter the configuration files for each processor you include.
4. Include the necessary DLL or shared-object library files.
5. Don't forget the RLP license file.

These steps are discussed in further detail below.

10.2. Environment Configuration

When redistributing RLP with another application, you may have to move RLP out of its original directory . If so, the application must set a new Basis root directory (*`BT_ROOT`*), where root means the installation directory, the parent of the top-level **rlp** directory. See Setting the Basis Root Directory [21] .

Many RLP language processors use dictionaries and other resource files. The configuration file for each processor specifies resource pathnames. Each pathname begins with `<env name="root"/>`. At runtime, RLP replaces this element with the pathname to the ***`BT_ROOT`*/rlp** directory. When you distribute an application, the location of the resources relative to the Basis root directory should not change. If it does change, you must edit the processor configuration files accordingly.

10.3. Reducing Your Processors

If you intend to ship all RLP functionality with your application, stop here. Read on if you want to streamline the performance of your application and reduce the size of your distribution by distributing only those RLP processors that your application needs.

First, determine which processors you need. Then edit the context configuration XML document , as described in Creating an RLP Application [17] , to include only those processors. Remember to check each processor's dependencies, as discussed in RLP Processors [123] , and include all processors that your chosen processors rely upon to function. For example, if you wish to include the Arabic Base Linguistics language processor, you must also include the Tokenizer language processor.

For each processor that you remove, you can also remove the processor DLL or shared-object library file from the binary directory (***`BT_ROOT`*/rlp/bin/*`BT_BUILD`***), where *`BT_ROOT`* is the Basis root directory and *`BT_BUILD`* is the platform identifier embedded in your SDK package file name (see Supported Platforms [13]). Use **rlp-global.xml** to identify the processor DLL or shared-object library file (see below).

Many of the language processors use an options file to designate the location of dictionaries and data files that the processor uses. For these processors, **rlp-global.xml** specifies the pathname of the options file. If your context does not include a given processor, you do not need any of the resources listed in the options file for that processor.

For example, **rlp-global.xml** contains the following entry for Arabic Base Linguistics (ARBL):

```
<languageprocessor name="ARBL" preload="no">
  <path type="so">bt_lp_arbl</path>
  <optionspath><env name="root"/>/etc/arbl-options.xml</optionspath>
</languageprocessor>
```

The processor DLL or shared-object library is identified by the `path` element: `bt_lp_arb` (the filename of the Windows DLL also contains a version number). The options file is identified by the `optionspath` element. `<env name="root"/>` means the RLP root directory (***`BT_ROOT`*****/rlp**). So the ARBL options file is ***`BT_ROOT`*****/rlp/etc/arble-options.xml**. This file designates the dictionary and data files that ARBL uses.

```
<arblconfig>
  <compatpath><env name="root"/>/arla/dicts/compat_table-<env name="endian"/>.bin</compatpath>
  <prefixpath><env name="root"/>/arla/dicts/dictPrefixes-<env name="endian"/>.bin</prefixpath>
  <rootpath><env name="root"/>/arla/dicts/dictRoots-<env name="endian"/>.bin</rootpath>
  <stempath><env name="root"/>/arla/dicts/dictStems-<env name="endian"/>.bin</stempath>
  <suffixpath><env name="root"/>/arla/dicts/dictSuffixes-<env name="endian"/>.bin</suffixpath>
  <modelpath><env name="root"/>/arla/dicts/ar_pos_model-<env name="endian"/>.bin</modelpath>
  <model2path>
    <env name="root"/>/arla/dicts/ar_pos_model2-<env name="endian"/>.bin</model2path>
</arblconfig>
```

Any element that starts with `<env name="root"/>` designates the pathname of a resource file. If the pathname includes `<env name="endian"/>`, substitute `LE` or `BE` in the filename, depending on whether the platform byte order is little-endian or big-endian.

So if your context does not include ARBL (you are not processing Arabic text), you do not need the ARBL processor or the data files listed above.

For a list of the files associated with each language processor, see Language Processor Resources [109] .

10.3.1. Individual Processor Configuration

After selecting your processors, you may also need to edit their individual configuration files, which specify dictionary locations and default parameters that may need to be modified.

10.4. Testing the Redistribution

We strongly recommend that you test your redistribution of RLP before shipping it with your application. To check the basic functionality of RLP, you can run the RLP command-line utility [6] . You should also run your own application in a variety of situations to ensure that RLP continues to function correctly.

10.5. Minimal Configuration

In order to optimize the performance of your application while using RLP, you may wish to start with a minimal configuration, and then add processors and reset properties as necessary.

The following sections show configuration files designed to minimize RLP's memory usage.

10.5.1. Arabic Minimal Configuration

10.5.1.1. Context Configuration

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE contextconfig SYSTEM
  "http://www.basistech.com/dtds/2003/contextconfig.dtd">

<contextconfig>
   <languageprocessors>
    <languageprocessor>Unicode Converter</languageprocessor>
   <languageprocessor>Tokenizer</languageprocessor>
    <languageprocessor>SentenceBoundaryDetector</languageprocessor>
    <languageprocessor>ARBL</languageprocessor>
  </languageprocessors>
</contextconfig>
```

10.5.2. Chinese Minimal Configuration

10.5.2.1. Context Configuration

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE contextconfig SYSTEM
  "http://www.basistech.com/dtds/2003/contextconfig.dtd">

<contextconfig>
  <properties>
    <!-- To minimize memory usage-->
    <property name="com.basistech.cla.pos" value="no"/>
     <property name="com.basistech.cla.readings" value="no"/>
  </properties>

  <languageprocessors>
    <languageprocessor>Unicode Converter</languageprocessor>
    <languageprocessor>CLA</languageprocessor>
  </languageprocessors>
</contextconfig>
```

10.5.3. European (BL1) Languages Minimal Configuration

10.5.3.1. Context Configuration

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE contextconfig SYSTEM
  "http://www.basistech.com/dtds/2003/contextconfig.dtd">

<contextconfig>
  <languageprocessors>
    <languageprocessor>Unicode Converter</languageprocessor>
    <languageprocessor>BL1</languageprocessor>
  </languageprocessors>
</contextconfig>
```

10.5.4. Japanese Minimal Configuration

10.5.4.1. Context Configuration

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

```
 "http://www.basistech.com/dtds/2003/contextconfig.dtd">

<contextconfig>
   <languageprocessors>
   <languageprocessor>Unicode Converter</languageprocessor>
   <languageprocessor>JLA</languageprocessor>
  </languageprocessors>
</contextconfig>
```

10.5.5. Korean Minimal Configuration

10.5.5.1. Context Configuration

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE contextconfig SYSTEM
 "http://www.basistech.com/dtds/2003/contextconfig.dtd">

<contextconfig>
  <languageprocessors>
   <languageprocessor>Unicode Converter</languageprocessor>
   <languageprocessor>KLA</languageprocessor>
  </languageprocessors>
</contextconfig>
```

10.5.6. Farsi (Persian) Minimal Configuration

10.5.6.1. Context Configuration

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE contextconfig SYSTEM
 "http://www.basistech.com/dtds/2003/contextconfig.dtd">

<contextconfig>
   <languageprocessors>
   <languageprocessor>Unicode Converter</languageprocessor>
  <languageprocessor>Tokenizer</languageprocessor>
   <languageprocessor>SentenceBoundaryDetector</languageprocessor>
   <languageprocessor>FABL</languageprocessor>
  </languageprocessors>
</contextconfig>
```

10.5.7. Urdu Minimal Configuration

10.5.7.1. Context Configuration

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE contextconfig SYSTEM
 "http://www.basistech.com/dtds/2003/contextconfig.dtd">

<contextconfig>
   <languageprocessors>
   <languageprocessor>Unicode Converter</languageprocessor>
  <languageprocessor>Tokenizer</languageprocessor>
   <languageprocessor>SentenceBoundaryDetector</languageprocessor>
   <languageprocessor>URBL</languageprocessor>
  </languageprocessors>
</contextconfig>
```

10.6. Language Processor Resources

Each language processor has a DLL or shared-object library. Many language processors have an options file, which along with other settings, designates the dictionary and other data files that the processor uses.

The remainder of this section provides the following information about each language processor:

Name
Case-sensitive name used to designate the processor in a context configuration file.

DLL or Shared-Object File
The processor file in ***`BT_ROOT`*****/rlp/bin/*****`BT_BUILD`*** . The DLL file names also include a version number.

Options File
The processor configuration file.

Data Files
The dictionaries and other data files used by the processor. These files are specified in the options file and may include user-defined files, such as user dictionaries or stopword lists.

Many binary files exist in two forms depending on the byte order of your platform. For `(LE|BE)` in a file name, substitute `LE` if the platform byte order is little endian, `BE` if the byte order is big endian.

For more information about each processor, see RLP Processors [123] .

Arabic Base Linguistics

Name
ARBL

DLL or Shared-Object File
bt_lp_arbl

Options File
BT_ROOT/rlp/etc/arbl-options.xml

Data Files
BT_ROOT/rlp/arla/dicts/compat_table-(LE|BE).bin
BT_ROOT/rlp/arla/dicts/dictPrefixes-(LE|BE).bin
BT_ROOT/rlp/arla/dicts/dictRoots-(LE|BE).bin
BT_ROOT/rlp/arla/dicts/dictStems-(LE|BE).bin
BT_ROOT/rlp/arla/dicts/dictSuffixes-(LE|BE).bin
BT_ROOT/rlp/arla/dicts/ar_pos_model-(LE|BE).bin
BT_ROOT/rlp/arla/dicts/ar_pos_model2-(LE|BE).bin
BT_ROOT/rlp/arla/dicts/dictVocalizations-(LE|BE).bin
BT_ROOT/rlp/arla/dicts/dictGlosses-(LE|BE).bin
BT_ROOT/rlp/arla/dicts/dictLemmas-(LE|BE).bin

Base Noun Phrase Detector

Name
BaseNounPhrase

DLL or Shared-Object File
bt_lp_bnp

Options File
BT_ROOT/rlp/etc/bnp-config.xml

Data Files
BT_ROOT/rlp/rlp/dicts/de_bnp_(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/de_bnp_data.bin
BT_ROOT/rlp/rlp/dicts/nl_bnp_(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/nl_bnp_data.bin
BT_ROOT/rlp/rlp/dicts/en_bnp_(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/en_bnp_data.bin
BT_ROOT/rlp/rlp/dicts/ja_bnp_(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/ja_bnp_data.bin
BT_ROOT/rlp/rlp/dicts/es_bnp_(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/es_bnp_data.bin
BT_ROOT/rlp/rlp/dicts/pt_bnp_(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/pt_bnp_data.bin
BT_ROOT/rlp/rlp/dicts/fr_bnp_(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/fr_bnp_data.bin
BT_ROOT/rlp/rlp/dicts/it_bnp_(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/it_bnp_data.bin
BT_ROOT/rlp/rlp/dicts/zh_bnp_(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/zh_bnp_data.bin
BT_ROOT/rlp/rlp/dicts/ar_bnp_(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/ar_bnp_data.bin

Base Linguistics Language Analyzer

Name
BL1

DLL or Shared-Object File
bt_lp_bl1

Options File
BT_ROOT/rlp/etc/bl1-config.xml

Data Files: Greek
BT_ROOT/rlp/bl1/dicts/el/*

Data Files: French
BT_ROOT/rlp/bl1/dicts/fr/*

Data Files: English
BT_ROOT/rlp/bl1/dicts/en/*

Data Files: Dutch
BT_ROOT/rlp/bl1/dicts/nl/*

Data Files: English Uppercase
BT_ROOT/rlp/bl1/dicts/en_uc/*
BT_ROOT/rlp/bl1/dicts/en/*

Data Files: Portuguese
BT_ROOT/rlp/bl1/dicts/pt/*

Data Files: Russian
BT_ROOT/rlp/bl1/dicts/ru/*

Data Files: German
BT_ROOT/rlp/bl1/dicts/de/*

Data Files: Italian
BT_ROOT/rlp/bl1/dicts/it/*

Data Files: Hungarian
BT_ROOT/rlp/bl1/dicts/hu/*

Data Files: Polish
BT_ROOT/rlp/bl1/dicts/pl/*

Data Files: Czech
BT_ROOT/rlp/bl1/dicts/cs/*

Data Files: Spanish
BT_ROOT/rlp/bl1/dicts/es/*

Chinese Language Analyzer

Name
CLA

DLL or Shared-Object File
bt_lp_cla

Options File
BT_ROOT/rlp/etc/cla-options.xml

Data Files
BT_ROOT/rlp/cma/dicts/zh_lex_(LE|BE).bin
BT_ROOT/rlp/cma/dicts/zh_reading_(LE|BE).bin
BT_ROOT/rlp/cma/dicts/zh_stop.utf8

Chinese Script Converter

Name
CSC

DLL or Shared-Object File
bt_lp_csc

Options File
BT_ROOT/rlp/etc/csc-options.xml

Data Files
BT_ROOT/rlp/cma/dicts/zh_lex_(LE|BE).bin
BT_ROOT/rlp/c2c/dicts/SCTTCmpt_(LE|BE).bin
BT_ROOT/rlp/c2c/dicts/TCTSCmpt_(LE|BE).bin

Farsi Base Linguistics

Name
FABL

DLL or Shared-Object File
bt_lp_fabl

Options File
BT_ROOT/rlp/etc/fabl-options.xml

Data Files
BT_ROOT/rlp/fabl/dicts/compat_table-(LE|BE).bin
BT_ROOT/rlp/fabl/dicts/dictPrefixes-(LE|BE).bin
BT_ROOT/rlp/fabl/dicts/dictStems-(LE|BE).bin
BT_ROOT/rlp/fabl/dicts/dictSuffixes-(LE|BE).bin
BT_ROOT/rlp/fabl/dicts/dictVocalizations-(LE|BE).bin
BT_ROOT/rlp/fabl/dicts/dictGlosses-(LE|BE).bin

Gazetteer

Name
Gazetteer

DLL or Shared-Object File
bt_lp_gazetteer

Options File
BT_ROOT/rlp/etc/gazetteer-options.xml

Data Files
BT_ROOT/rlp/rlp/dicts/fr-titles-gazetteer-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/de-titles-gazetteer-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/it-titles-gazetteer-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/es-titles-gazetteer-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/nl-titles-gazetteer-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/pt-titles-gazetteer-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/fa-titles-gazetteer-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/ur-titles-gazetteer-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/ru-nat-rel-gazetteer-(LE|BE).bin

HTML Stripper

Name
HTML Stripper

DLL or Shared-Object File
bt_lp_htmlstripper

Options File
None

Data Files
None

iFilter

Name
iFilter

DLL or Shared-Object File
bt_lp_ifilter

Options File
None

Data Files
None

Japanese Language Analyzer

Name
JLA

DLL or Shared-Object File
bt_lp_jma

Options File
BT_ROOT/rlp/etc/jla-options.xml

Data Files
BT_ROOT/rlp/jma/dicts/JP_(LE|BE).bin
BT_ROOT/rlp/jma/dicts/JP_(LE|BE)_Reading.bin
BT_ROOT/rlp/jma/dicts/JP_stop.utf8

Korean Language Analyzer

Name
KLA

DLL or Shared-Object File
bt_lp_kla

Options File
BT_ROOT/rlp/etc/kla-options.xml

Data Files
BT_ROOT/rlp/kma/dicts/
BT_ROOT/rlp/utilities/data/
BT_ROOT/rlp/kma/dicts/kr_stop.utf8

Language Boundary Detector

Name
Language Boundary

DLL or Shared-Object File
bt_lp_lbd

Options File
None

Data Files
None

ManyToOneNormalizer

Name
ManyToOneNormalizer

DLL or Shared-Object File
bt_lp_m1norm

Options File
BT_ROOT/rlp/etc/normalizer-options.xml

Data Files
BT_ROOT/rlp/jma/dicts/jon_(LE|BE).bin

mime_detector

Name
mime_detector

DLL or Shared-Object File
bt_lp_mime_detector

Options File
None

Data Files
None

Named Entity Extractor

Name
NamedEntityExtractor

DLL or Shared-Object File
bt_lp_ne

Options File
BT_ROOT/rlp/etc/ne-config.xml

Data Files: Russian
BT_ROOT/rlp/rlp/dicts/ru-memm-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/ru-funcwords-(LE|BE).bin

Data Files: French
BT_ROOT/rlp/rlp/dicts/fr-memm-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/fr-funcwords-(LE|BE).bin

Data Files: English
BT_ROOT/rlp/rlp/dicts/en-memm-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/en-funcwords-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/en-gazetteer-(LE|BE).bin

Data Files: Dutch
BT_ROOT/rlp/rlp/dicts/nl-memm-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/nl-funcwords-(LE|BE).bin

Data Files: English Uppercase
BT_ROOT/rlp/rlp/dicts/en_uc-memm-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/en_uc-funcwords-(LE|BE).bin

Data Files: German
BT_ROOT/rlp/rlp/dicts/de-memm-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/de-funcwords-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/de-gazetteer-(LE|BE).bin

Data Files: Korean
BT_ROOT/rlp/rlp/dicts/ko-memm-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/ko-funcwords-(LE|BE).bin

Data Files: Italian
BT_ROOT/rlp/rlp/dicts/it-memm-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/it-funcwords-(LE|BE).bin

Data Files: Urdu
BT_ROOT/rlp/rlp/dicts/ur-memm-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/ur-funcwords-(LE|BE).bin

Data Files: Farsi (Persian)
BT_ROOT/rlp/rlp/dicts/fa-memm-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/fa-funcwords-(LE|BE).bin

Data Files: Arabic
BT_ROOT/rlp/rlp/dicts/ar-memm-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/ar-funcwords-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/ar-gazetteer-(LE|BE).bin

Data Files: Simplified Chinese
BT_ROOT/rlp/rlp/dicts/zh_sc-memm-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/zh-funcwords-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/zh_sc-gazetteer-(LE|BE).bin

Data Files: Japanese
BT_ROOT/rlp/rlp/dicts/ja-memm-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/ja-funcwords-(LE|BE).bin

Data Files: Spanish
BT_ROOT/rlp/rlp/dicts/es-memm-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/es-funcwords-(LE|BE).bin

Data Files: Traditional Chinese
BT_ROOT/rlp/rlp/dicts/zh_tc-memm-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/zh-funcwords-(LE|BE).bin
BT_ROOT/rlp/rlp/dicts/zh_tc-gazetteer-(LE|BE).bin

Named Entity Redactor

Name
NERedactLP

DLL or Shared-Object File
bt_lp_ne_redact

Options File
BT_ROOT/rlp/etc/neredact-config.xml

Data Files
None

Core Library for Unicode

Name
RCLU

DLL or Shared-Object File
bt_lp_rclu

Options File
None

Data Files
None

Other (File names Include the RCLU version number)
Windows (required at compile time): btuc
Windows (loaded at runtime): btchi, btjpn, btkor, btlat, btrow
Unix (required at compile time): *`BT_ROOT`*/rlp/lib/*`BT_BUILD`*/libbtunicode
Unix (loaded at runtime): btchi, btjpn, btkor, btlat, btrow

Regular Expression

Name
RegExpLP

DLL or Shared-Object File
bt_lp_regexp

Options File
BT_ROOT/rlp/etc/regex-config.xml

Data Files
None

REXML

Name
REXML

DLL or Shared-Object File
bt_op_rexml

Options File
None

Data Files
None

Rosette Language Identifier

Name
RLI

DLL or Shared-Object File
bt_lp_rli

Options File
None

Data Files
None

Other
Windows: bteuclid
Unix: *`BT_ROOT`*/rlp/lib/*`BT_BUILD`*/libbteuclid

Script Boundary

Name
Script Boundary

DLL or Shared-Object File
bt_lp_scrbd

Options File
None

Data Files
None

Sentence Boundary Detector

Name
SentenceBoundaryDetector

DLL or Shared-Object File
bt_lp_sbd

Options File
BT_ROOT/rlp/etc/sbd-config.xml

Data Files
BT_ROOT/rlp/rlp/dicts/de-dict.dict
BT_ROOT/rlp/rlp/dicts/en-dict.dict

Stopwords

Name
Stopwords

DLL or Shared-Object File
bt_lp_stop

Options File
BT_ROOT/rlp/etc/stop-options.xml

Data Files
BT_ROOT/rlp/etc/en-stopwords.txt

Text Boundary Detector

Name
Text Boundary

DLL or Shared-Object File
bt_lp_tbd

Options File
None

Data Files
None

Tokenizer

Name
Tokenizer

DLL or Shared-Object File
bt_lp_tokenizer

Options File
None

Data Files
None

Unicode Converter

Name
Unicode Converter

DLL or Shared-Object File
bt_lp_unicode_converter

Options File
None

Data Files
None

Urdu Base Linguistics

Name
URBL

DLL or Shared-Object File
bt_lp_urbl

Options File
BT_ROOT/rlp/etc/urbl-options.xml

Data Files
BT_ROOT/rlp/urbl/dicts/compat_table-(LE|BE).bin
BT_ROOT/rlp/urbl/dicts/dictPrefixes-(LE|BE).bin
BT_ROOT/rlp/urbl/dicts/dictStems-(LE|BE).bin
BT_ROOT/rlp/urbl/dicts/dictSuffixes-(LE|BE).bin
BT_ROOT/rlp/urbl/dicts/dictVocalizations-(LE|BE).bin
BT_ROOT/rlp/urbl/dicts/dictGlosses-(LE|BE).bin

10.7. Managing RLP Configuration Files

The RLP SDK uses a number of configuration files to determine the name and location of resources, including the RLP license, language processors, dictionaries, and other data files. This section details the measures you need to take if you want to move the RLP configuration files or associated resource files to locations other than their original location in the SDK distribution.

10.7.1. The Configuration Files

Their Original Location. The RLP configuration files are in ***BT_ROOT*/rlp/etc**, where *BT_ROOT* is the root of the RLP SDK installation and a value you specify during RLP initialization.

Environment Configuration. The environment configuration file (**rlp-global.xml**) provides the path to the RLP license, the named entities configuration file (**ne-types.xml**), and the language processor option files (**bl1-config.xml**, **arbl-options.xml**, **jla-options.xml**, etc.).

RLP License. The RLP license file (***BT_ROOT*/rlp/rlp/licenses/rlp-license.xml**) defines the scope of the RLP features that you are authorized to use in your applications.

Named Entities Configuration. The named entities configuration file (**ne-types.xml)** defines the names of entity types and subtypes and the weights for resolving conflicts between statistical analysis, regular expressions, and gazetteers when more than one processor identifies the same or overlapping text as an entity.

Option Files. The option files define processor-specific settings, including the paths to dictionaries and other resources.

If you move some or all of these files.

1. You must edit the configuration files [119] to reflect the new locations.
2. During RLP initialization, be sure to provide the correct location of the environment configuration file [121] .

The values you provide at initialization enable RLP to find the environment configuration file. The environment configuration file must include the correct path to the license and the other configuration files, which must include the correct path to the resources they use.

10.7.2. Editing Configuration Files

The only edits you need to make in the configuration files are to correct the paths. Each path is prepended with `<env name="root"/>`, which RLP interprets as ***BT_ROOT*/rlp**, using the *BT_ROOT* that you specified during initialization.

10.7.2.1. Editing the Environment Configuration File

The environment configuration file (**rlp-global.xml**) specifies the path to the processor options files. For each processor that uses an options file, the associated `<languageprocessor>` element contains an `<optionspath>` element, which uses `<env name="root/>` to begin each path with an absolute reference to ***BT_ROOT*/rlp**. For example:

```
<languageprocessor name="BaseNounPhrase" preload="no">
  <path type="so">bt_lp_bnp</path>
  <optionspath><env name="root"/>/etc/bnp-config.xml</optionspath>
</languageprocessor>
```

If RLP is installed in **C:/Program Files/Basis Technology/RLP SDK**, the pathname of the Base Noun Phrase Detector options file is **C:/Program Files/Basis Technology/RLP SDK/rlp/etc/bnp-config.xml**.

You do not need to edit the path unless you move the options file to a different location.

For any paths that you edit, we recommend that you use absolute paths. [1] You can prepend your paths with `<env name="root"/>`, or you can put in a literal string that provides the complete absolute path. If, for example, you move **bnp-config.xml** to **C:\configurations**, you can specify the path as follows:

```
<optionspath><env name="root"/>/../../../configurations/bnp-config.xml</optionspath>
```

or

```
<optionspath>C:/configurations/bnp-config.xml</optionspath>
```

Note: On Windows, you can also use the '\' directory delimiter, but '/' works fine and is consistent with the current path layout in the RLP configuration files.

10.7.2.2. Editing the Language Processor Option Files

Each of the language processor option files specifies the path to the resources the processor requires. As in the environment configuration file, each path specification begins with `<env name="root/>`, which at runtime RLP replaces with the absolute path to ***BT_ROOT*/rlp**.

For example, **bnp-config.xml** provides the path to a grammar file and a data file for each language it support. For Japanese:

```
<config language="ja">
  <grammarpath><env name="root"/>/rlp/dicts/ja_bnp_<env name="endian"/>.bin</grammarpath>
  <datapath><env name="root"/>/rlp/dicts/ja_bnp_data.bin</datapath>
</config>
```

If you move these files to **C:/my_application/dicts**, you could edit the entry to read:

```
<!-- BT_ROOT is C:/Program Files/Basis Technology/RLP SDK-->
<config language="ja">
  <grammarpath><env name="root"/>/../../../my_application/dicts/ja_bnp_<env name="endian"/>.bin</grammarpath>
  <datapath><env name="root"/>/../../../my_application/dicts/ja_bnp_data.bin</datapath>
</config>
```

or

```
<config language="ja">
  <grammarpath>C:/my_application/dicts/ja_bnp_<env name="endian"/>.bin</grammarpath>
```

[1]If you use a relative path, it is relative to the working directory of the current process, not the location of the file that designates the path. Accordingly you can only use relative paths if the working path is guaranteed to be consistent for all RLP applications that use these configuration files.

```
  <datapath>C:/my_application/dicts/ja_bnp_data.bin</datapath>
</config>
```

If you are only deploying on Windows, as the example implies, you can replace `<env name="endian"/>` with `LE` (little endian).

Review each of the option files for the processors you plan to use, and edit the path specifications, using the absolute path to the resource, or `<env name="root/>` plus the relative path from ***`BT_ROOT`*/rlp** to the resource.

10.7.3. Initializing the RLP Environment

After setting *`BT_ROOT`*, provide the path to the environment configuration file when you initialize the RLP environment.

As illustrated in the sample applicatons in the *RLP Application Developer's Guide*, the C++, Java, C, and .NET APIs include calls for setting *`BT_ROOT`* and the pathname of the environment configuration file.

See:

- C++ Sample [28]
- Java Sample [34]
- C Sample [59]
- .NET sample [69]

10.7.4. Note On the Sample Applications

The sample applications shipped with RLP and the scripts for building and running the samples are based on the distribution path structure. If you move the samples or the files they use (the environment configuration file, processor option files, input files, RLP context files), you must modify the scripts and sample sources accordingly.

Chapter 11. RLP Processors

11.1. Overview

The RLP language processors are documented in this chapter in the format described below.

Note that licenses are required on a per-language, per-feature basis. Before a language processor runs, it checks for a license for the given language and feature (e.g., English tokenizing). Licensing failures are recorded in a log if logging is turned on to report the warning channel. See Logging [21] for more details. RLP also provides an API for determining the scope of features enabled by your license. See Getting License Information [24] .

The documentation for each processor contains the following information:

Name
Name used to specify the processor in a context configuration file. Names are case-sensitive.

Dependencies
If the processor relies on the result(s) of another processor, then it is listed here by Name.

Note that the transitive closure of dependencies is not shown here; if processor C depends on the output of B, and B depends on the output of A, then the Dependencies section for C only includes B, not B and A.

Use processor dependencies to optimize the context configuration file. First, make sure that all desired language processors are included in the file. Then make sure that all dependencies are included for each of the processors. To maximize performance, remove everything else.

For example, to create a context that handles Unicode-encoded English and Japanese down to the sentence boundary detection level, with REXML formatted output, the context configuration file should look like the following:

```
<contextconfig>
  <languageprocessors>
    <languageprocessor>Unicode Converter</languageprocessor>
    <languageprocessor>BL1</languageprocessor>
    <languageprocessor>JLA</languageprocessor>
    <languageprocessor>SentenceBoundaryDetector</languageprocessor>
  <languageprocessor>REXML</languageprocessor>
  </languageprocessors>
</contextconfig>
```

For English, the BL1 processor provides tokenization, POS tagging, and sentence boundaries. The JLA does nothing, because it produces output only for Japanese. The SentenceBoundaryDetector does nothing, because BL1 has already produced sentence boundaries.

For Japanese, BL1 does nothing because Japanese is not a supported BL1 language. JLA provides tokenization and POS tagging (and possibly readings and compounds). SentenceBoundaryDetector provide sentence boundaries in this case, because there are no existing sentence boundary results when it is run.

Language Dependent
Indicates whether the processor is language specific and lists the languages it can process. You can employ either of the following techniques to inform the language-specific processors of the language of the input text. Processors in the context that do not process the specified language do nothing.

- Use the Rosette Language Identifier (RLI) [178] to detect the language.
- Use the RLP Context object to specify the language.

  For the C++ API, see `ProcessFile`, and `ProcessBuffer` in BT_RLP_Context [api-reference/cpp-reference/classBT__RLP__Context.html]. For the Java API, see the `process` methods in RLPContext [api-reference/java-reference/com/basistech/rlp/RLPContext.html].

XML-Configurable Options
If the processor has an XML options file that the user can configure, this section details the format of the file and describes each option.

Context Properties
Describes context properties that configure the runtime behavior of the processor. See also Global Context Properties [125] .

Many processors have options that may be configured in the context configuration file. Properties can be set via entries in the context configuration XML file, as determined by the **contextconfig** DTD:

```
<!ELEMENT properties (property+)>
<!ELEMENT property EMPTY>
<!ATTLIST property name CDATA #REQUIRED>
 <!ATTLIST property value CDATA #REQUIRED>
```

The syntax of the configuration is very important. When a context property is specified in a configuration file, its name must be prefixed by `com.basistech.` *<processor name>*. For example, The following setting specifies the REXML output file:

```
<property name="com.basistech.rexml.output_pathname"
   value="rlp-output.xml"/>
```

Boolean Properties

You must use one of the following case-sensitive values to set a boolean property: `"yes"`, `"true"`, `"no"`, `"false"`. For example,

```
<property name="com.basistech.cla.break_at_alphanum_intraword_punct"
   value="TRUE"/>
```

does not work. Depending on the processor, using an invalid value is either interpreted as false or is ignored (the default setting is used).

Context properties can also be specified via the API. See Setting Context Properties [26] .

Description
This section describes in detail the functionality provided by the processor.

11.1.1. Text Being Processed

An RLP language processor is designed either to process text in one language (the common case) or to process text in multiple languages.

One language at a time. Most RLP language processors are designed to process text in a single language. You either specify the language when you use the context object to process the input text, or you use the Language Identifier [178] to identify the language. The context may include multiple language processors, but only the processors applicable to the language of the input text are used. For example, if the input text is Japanese and the context includes the Japanese and Chinese language analyzers (JLA and CLA), JLA processes the text and CLA is inactive.

Input text in more than one language. Three of the processors are designed to process input text that may contain multiple languages and multiple writing scripts: Text Boundary Detector [186] , Script Boundary Detector [184] , and Language Boundary Detector [152] . Collectively, these three language processors make up the Rosette Language Boundary Locator (RLBL), formerly known as the Multilingual Language Identifier (MLI). You use these processors in the order listed to determine language regions. Then you can extract each region from the raw text and submit it to another context with the appropriate language identifier for detailed linguistic processing.

11.2. Global Context Properties

Most context properties apply to a single language processor (the processor name is embedded in the property name) and are described in the section on that processor. A global context property is not processor specific; it may apply to multiple languages and multiple language processors.

The mechanisms for setting global context properties and processor-specific context properties is the same; see Setting Context Properties [26] .

Global Context Property	Type	Default	Description
`com.basistech.bl.query`	boolean	false	If set to true, the base linguistics processor treats the input as a query (search terms), not as prose (sentences). Accelerates processing and avoids the errors identifying STEMs [87] that may occur when attempting to apply contextual analysis to a linguistically arbitrary collection of words. Given the absence of syntactic context, POS [86] tags are not returned. Currently applies to the Arabic Base Linguistics [126] , the Farsi Base Linguistics [141] , and Urdu Base Linguistics [188] processors.

11.3. Arabic Script Normalization

ARBL [126] , FABL [141] , and URBL [188] apply Arabic script normalization respectively to Arabic, Farsi (Persian), and Urdu input text prior to performing language-specific normalization.

Important

If processing text in Arabic script that includes characters from Arabic Presentation Forms A (U +FB50 - U+FDFF) and/or Arabic Presentation Forms B (U+FE70 - U+FEFF), use RCLU [168] with `com.basistech.rclu.FormKCNormalization` [169] set to `true` to normalize these characters to standard Arabic script characters (U+0600 - U+06FF). Otherwise, these characters are not recognized as Arabic-script characters and the words containing them are not recognized as Arabic, Farsi (Persian), or Urdu.

When you examine the results generated by RLP, keep in mind that some fonts do not accurately display all Arabic ligatures. The Scheherazade Sil International font does an excellent job of rendering Arabic ligatures.

- The following diacritics are removed: *kashida*, *dammatan*, *kasratan*, *fatha*, *damma*, *kasra*, *shadda*, *sukun*.
- The following characters are removed: *left-to-right marker*, *right-to-left marker*, *zero-width joiner*, *BOM*, *non-breaking space*, *soft hyphen*, *full stop*.
- *Alef maksura* is converted to *yeh* unless it is at the end of the word or followed by *hamza*.
- All numbers are are converted to Arabic numbers (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) [1] , thousand separators are removed, and the decimal separator changed to a period (U+002E). The normalizer handles cases where ر (*reh*) is (incorrectly) used as the decimal separator.
- *Alef with hamza above*: ٵ (U+0675), ٲ (U+0672), or ا (U+0627) combined with ٔ (U+0654) is converted to أ (U+0623).
- *Alef with madda above*: ا (U+0627) combined with ٓ (U+0653) is converted to آ (U+0622).
- *Alef with hamza below*: ٳ (U+0673) or ا (U+0627) combined with ٕ (U+0655) is converted to إ (U+0625).
- *Misra to Ain*: *Misra* (U+060F) is converted to ع (U+0639).
- *Swash kaf to kaf*: ڪ (U+06AA) is converted to ک (U+06A9).
- *Heh*: ە (U+06D5) is converted to ه (U+0647).
- *Teh marbuta*: ۃ (U+06C3) is converted to ة (U+0629).
- *Yeh with hamza above*: The following combinations are converted to ئ (U+0626).

  ی (U+06CC) combined with ٔ (U+0654)

  ى (U+0649) combined with ٔ (U+0654)

  ي (U+064A) combined with ٔ (U+0654)
- *Waw with hamza above*: و (U+0648) combined with ٔ (U+0654), ٷ (U+0677), or ٶ (U+0676) is converted to ؤ (U+0624).

11.4. Processors

11.4.1. Arabic Base Linguistics

Name
: **ARBL**

Dependencies
: Tokenizer, SentenceBoundaryDetector

[1]As distinguished from the Arabic-Indic numerals often used in Arabic script (٠, ١, ٢, ٣, ٤, ٥, ٦, ٧, ٨, ٩) or the Eastern Arabic-Indic numerals often used in Farsi (Persian) and Urdu Arabic script (۰, ۱, ۲, ۳, ۴, ۵, ۶, ۷, ۸, ۹).

Language Dependent

Arabic

XML-Configurable Options

None. The paths to the ARBL dictionaries and related resources are defined in ***BT_ROOT*/rlp/etc/arbl-options.xml**.

Context Properties

For brevity, the `com.basistech.arbl` prefix has been removed from the property names in the first column. Hence the full name for `roots` is `com.basistech.arbl.roots`.

Property	Type	Default	Description
`alternatives`	boolean	false	If set to true, ARBL posts an additional results of type `ALTERNATIVE_NORM`, `ALTERNATIVE_LEMMAS`, `ALTERNATIVE_ROOTS`, `ALTERNATIVE_STEMS`, and `ALTERNATIVE_PARTS_OF_SPEECH`.
`lemmas`	boolean	false	If set to true, ARBL posts an additional result of type `LEMMA`.
`roots`	boolean	false	If set to true, ARBL posts an additional result of type `ROOTS`.
`variations`	boolean	false	If set to true, ARBL posts an additional result of type `TOKEN_VARIATION`.

When processing a query (a collection of one or more search terms) rather than prose (one or more sentences), set the `com.basistech.bl.query` global context property [125] to true.

Description

The ARBL language processor performs morphological analysis and part-of-speech (POS) tagging for texts written in Modern Standard Arabic.

The processor generates the following result types:

- STEM [87]
- NORMALIZED_TOKEN [86]
- PART_OF_SPEECH [86]
- TOKEN_VARIATIONS [89] — if `com.basistech.arbl.variations` is true (default is false).
- ROOTS [87] — if `com.basistech.arbl.roots` is true (default is false).
- LEMMA [85] — if `com.basistech.arbl.lemmas` is true (default is false).
- Alternative Analyses — if `com.basistech.arbl.alternatives` is true (default is false). Unless `com.basistech.bl1.query` is set to true, the first analysis is the disambiguated analysis, and the others are not ordered. If `com.basistech.bl1.query` is set to true, disambiguation does not take place and the analyses are not ordered.
  - ALTERNATIVE_STEMS [84]
  - ALTERNATIVE_NORM [83]
  - ALTERNATIVE_PARTS_OF_SPEECH [83]
  - ALTERNATIVE_ROOTS [84] — `com.basistech.arbl.roots` must be set to true
  - ALTERNATIVE_LEMMAS [83] — `com.basistech.arbl.lemmas` must be set to true

Normalization

Each Arabic token is normalized prior to morphological analysis. Unless `com.basistech.bl.query` is set to true, ARBL may choose a token variant during morphological analysis and normalize it to produce the value for the `NORMALIZED_TOKEN` result.

For languages written in Arabic script, normalization is performed in two stages: generic Arabic script normalization [125] and language-specific normalization.

The following language-specific normalizations are performed on the output of the Arabic script normalization:

- *Zero-width non joiner* (U+200C) and *superscript alef* ٰ (U+0670) are removed.
- *Fathatan* ً (U+064B) is removed.
- Farsi *yeh* (U+06CC) is normalized to *yeh* (U+064A) if it is initial or medial; if final, it is normalized to *alef maksura* (U+0649).
- *Kaf* ک (U+06A9) is converted to ك (U+0643).
- *Heh* ہ (U+06C1) or ھ (U+06BE) is converted to ه (U+0647).

Following morphological analysis, the normalizer does the following:

- *Alef wasla* ٱ (U+0671) is replaced with *plain alef* ا (U+0627).
- If a word starts with the incorrect form of an *alef*, the normalizer retrieves the correct form: *plain alef* ا (U+0627), *alef with hamza above* أ (U+0623), *alef with hamza below* إ (U+0625), or *alef with madda above* آ (U+0622).

Variations

The analyzer can generate a number of variant forms for each Arabic token to account for the orthographic irregularity seen in contemporary written Arabic. Each variation is added to the output of the previous variation, starting with the normalized form:

- If a token contains a word-final *hamza* preceded by *yeh* or *alef maksura*, then a variant is created that replaces these with *hamza* seated on *yeh*.
- If a token contains *waw* followed by *hamza* on the line, a variant is created that replaces these with *hamza* seated on *waw*.
- Variants are created where word-final *heh* is replaced by *teh marbuta*, and word-final *alef maksura* is replaced by *yeh*.

When the `com.basistech.arbl.variations` property setting is `true` the generated orthographic variations for a token are returned in the `TOKEN_VARIATIONS` result. Regardless of the property setting, these variations are generated and considered in the morphological analysis and POS tagging.

The stem returned in the `STEM` result is the normalized token with affixes (such as prepositions, conjunctions, the definite article, proclitic pronouns, and inflectional prefixes) removed.

When the `com.basistech.arbl.roots` property is true, the consonantal root for the token is generated, if possible.

A Note on Stemming

In the process of stripping morphemes (affixes) from a token, ARBL produces a `STEM`, a `LEMMA`, and a `ROOT`. Stems and lemmas result from stripping most of the inflectional morphemes, while roots result from stripping derivational morphemes.

Inflectional morphemes indicate plurality or verb tense. Different forms, such as singular and plural noun, or past and present verb tense share the same stem if the forms are regular. If some of the forms are irregular, they do not share the same stem, but do share the same lemma. Since stems and lemmas preserve the meaning of words, they are very useful in text retrieval and search in general.

Words that have a more distant linguistic relationship share the same root.

Examples. The singular form الكتابة (*al-kitaaba*, the writing) and plural form كتابات (*kitaabaat*, writings) share the same stem: كتاب (*kitaab*). On the other hand, كُتُب (*kutub*, books) is an irregular form and does not have the same stem as كِتَاب (*kitaab*, book). But both forms do share the same lemma, which is the singular form كِتَاب (*kitaab)*. The words مكتبة (*maktaba*, library), المَكْتَب (*al-maktab*, the desk), كُتُب (*kutub*, books), and الكتابة (*al-kitaaba*, the writing) are related in the sense that a library contains books and desks, a desk is used to write on, and writings are often found in books. All of these words share the same root: كَتَبَ (*kataba*).

11.4.2. Base Linguistics Language Analyzer

Name
BL1

Dependencies
None (see below)

Language Dependent
Yes. This language processor processes many European languages. The supported languages are the following:

Language Code	Language
cs	Czech
de	German
el	Greek
en	English
en_uc	Upper-Case English[a]
es	Spanish
fr	French
hu	Hungarian
it	Italian
nl	Dutch

Language Code	Language
pl	Polish
pt	Portuguese
ru	Russian

[a]For more accurate processing of upper-case English text, specify the `en_uc` language code in place of `en`.

See Appendix B [209] for a listing of the POS tags for these languages.

XML-Configurable Options

BL1 uses a memory limit, language-specific machinery, and optional user dictionaries specified in ***`BT_ROOT`*****/rlp/etc/bl1-config.xml**.

For each language, the machinery may include a tokenizing FST (finite state transducer), a morphological lookup script, default part-of speech tags, special tags for internal use only, and an internal lemma dictionary used during disambiguation.

Memory limit. The `bl1config memory-limit` attribute defines the limit on the amount of language machinery BL1 will load into memory. Each time BL1 is called, it loads resources for the language being processed, and holds these resources for the remainder of the RLP environment session. If BL1 is called multiple times with input in different languages, the memory requirements increase. If the defined limit is reached, BL1 reports a warning, clears memory, re-initializes, and, continues processing the next language. By default, the limit is 200,000,000 bytes: `<bl1config memory-limit="200000000">`. You can modify this limit; if you set it to 0, there is no limit. Keep in mind that if the limit is too high, BL1 is forced to start paging, and performance deteriorates.

multiprogramming-limit. The `multiprogramming-limit` specifies the number of threads that may execute in the BL1 processor simultaneously. The default is 100. Simultaneously executing threads make simultaneous demands upon the virtual memory space available to BL1. If the application in use exploits multiple threads, and a multiple-core multi-processor is in use, it is advantageous to support at least that many threads. But if, in a highly parallel and multiprogrammed application such as a web server, thread memory contention results in decreased performance, it may be advantageous to experiment with bringing this number down nearer to the number of processor cores.

Morphological Lookup Script. The only setting you can modify for a language is the morphological lookup script. Two scripts are provided for each language:

lookup-mor.txt
: This script uses full morphological tagging internally. These extra tags are used by NamedEntityExtractor [156] for more accurate results. This is the default script.

lookup-lem.txt
: This script uses only part-of-speech tags. These are shorter than the full morphological tags, so less memory is used and the lookups are faster (around 10 percent). Do not use this script if NamedEntityExtractor results are needed.

For example, to change the morphological lookup script for English from full morphological tagging to part-of-speech tagging, revise the <morpho-script> in the English section:

```
<bl1-options language="en">
  <tokenizer><env name="root"/>/bl1/dicts/en/tokenize.fst</tokenizer>
  <morpho-dir><env name="root"/>/bl1/dicts/en</morpho-dir>
  <morpho-script>
    <env name="root"/>/bl1/dicts/en/lookup-mor.txt
  </morpho-script>
```

```
 ...
</bl1-options>
```

so that it reads

```
 <morpho-script>
   <env name="root"/>/bl1/dicts/en/lookup-lem.txt
 </morpho-script>
```

User Dictionaries. You can create one or more user dictionaries for each language that BL1 supports. To instruct BL1 to employ a user dictionary, you must add a `<user-dict>` element to the appropriate language section in **bl1-config.xml**. The following example adds a German user dictionary.

```
 <bl1-options language="de">
  <tokenizer><env name="root"/>/bl1/dicts/de/tokenize.fst</tokenizer>
  ...
<user-dict>
<env name="root"/>/bl1/dicts/de/userdict-<env name="endian"/>.bin</user-dict>
</bl1-options>
```

`<env-name="endian">` evaluates to LE on little-endian platforms and BE on big-endian platforms. To deploy a user dictionary on both little-endian platforms (such as an Intel x86 CPU) and big-endian platforms (such as Sun's SPARC), compile separate dictionaries for each. For more information, see European Language User Dictionaries [193] .

Morphological Caching. For both English and German, the configuration file includes a <morpho-cache> element with the path to a morphological dictionary for commonly used words. This dictionary is cached at runtime to improve the speed with which BL1 can analyze text in these languages.

Pathnames. The BL1 configuration file specifies pathnames to various resources, including the morphological directory where FSTs, data files, and scripts are kept. Each pathname begins with `<env name="root">`. At runtime, RLP replaces this element with the pathname to the RLP root directory (***BT_ROOT*/rlp**). When you distribute an application, the location of the resources relative to RLP root should not change. See Environment Configuration [105] .

Context Properties

None

Description

The Base Linguistics Language Processor provides tokenization, sentence boundary detection, stemming, and part-of-speech tagging for the supported languages. By default, it returns the following results:

- TOKEN [88]
- TOKEN_OFFSET [88]
- SENTENCE_BOUNDARY [87]
- PART_OF_SPEECH [86]
- STEM [87] [2]

For languages with compound words (German, Dutch, and Hungarian), the components are separated and returned in COMPOUND [84] results.

[2]Depending on the `PART_OF_SPEECH` BL1 assigns to a `TOKEN`, the `STEM` may vary. For example, the English "getting" may return "get" (verb: PARTPRES or VPROG) or "getting" (adjective or noun: ADJING or NOUNING), depending on context.

Note

Tokenizer and Sentence Boundary Detector are no longer necessary for BL1, and thus they do nothing when BL1 precedes them in the context.

User Dictionaries
: You may create your own dictionaries for the languages that BL1 supports. See European Language User Dictionaries [193] .

11.4.3. Base Noun Phrase Detector

Name
: **BaseNounPhrase**

Dependencies
: Tokenized text and part-of-speech tags: BL1, CLA, JLA, KLA; or Tokenizer and ARBL.

Language Dependent
: Arabic, Chinese, Dutch, English, French, German, Italian, Japanese, Korean, Portuguese, Spanish.

XML-Configurable Options
: The BaseNounPhrase options are defined in ***BT_ROOT*/rlp/etc/bnp-config.xml**. For example:

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

<bnpconfig version="2.0">
  <config language="de">
    <grammarpath><env name="root"/>/rlp/dicts/de_bnp.bin</grammarpath>
    <datapath><env name="root"/>/rlp/dicts/de_bnp_data.bin</datapath>
  </config>

  <config language="en">
    <grammarpath><env name="root"/>/rlp/dicts/en_bnp.bin</grammarpath>
    <datapath><env name="root"/>/rlp/dicts/en_bnp_data.bin</datapath>
  </config>

  <config language="ja">
    <grammarpath><env name="root"/>/rlp/dicts/ja_bnp.bin</grammarpath>
    <datapath><env name="root"/>/rlp/dicts/ja_bnp_data.bin</datapath>
  </config>

</bnpconfig>
```

This file conforms to **bnpoconfig.dtd**:

```
<!ENTITY % pathname "(#PCDATA | env)+"
<!ELEMENT bnpconfig (config)+>
<!ATTLIST bnpconfig version #FIXED "2.0">
<!ELEMENT config (grammarpath, datapath)>
<!ATTLIST config language CDATA #REQUIRED>
<!ELEMENT grammarpath %pathname>
<!ELEMENT datapath %pathname>
<!ELEMENT env EMPTY>
<!ATTLIST env name CDATA #REQUIRED>
```

BaseNounPhrase data is available for ten languages: Arabic, Chinese (Simplified and Traditional), Dutch, English, French, German, Italian, Japanese, and Spanish.

The names of the grammar and data files are all patterned on *ln* **_bnp.bin** and *ln* **_bnp_data.bin**, where *ln* is replaced by the ISO 639-1 two-letter language code.

Context Properties
None

Description
One of the most important kinds of structure to assign to a document is the identification of noun phrases (NP). A phrase is a self-contained group of words with a discrete meaning; a noun phrase is a phrase that functions as a noun in a sentence. Examples of noun phrases include (almost) every title of a book, movie, play and piece of music.

Noun phrases can also be recursive. That is, a noun phrase may contain other noun phrases as component parts. For instance, the following are all noun phrases:

```
it
apples
the apple
the green apple
the round red juicy apple
the green apple on the table
the red apple on the table in the kitchen
the red apple that I ate at lunch yesterday
```

A base noun phrase is a noun phrase that is not recursive, that is, it does not contain other noun phrases inside it. So, the first five noun phrases in the list above are base noun phrases, and the remaining ones are complex noun phrases that contain base noun phrases inside them. Below, the list of noun phrases is repeated with the base noun phrases bracketed:

```
[it]
[apples]
[the apple]
[the green apple]
[the round red juicy apple]
[the green apple] on [the table]
[the red apple] on [the table] in [the kitchen]
[the red apple] that [I] ate at [lunch] yesterday
```

Note

A noun phrase that involves an associative relationship between two nouns is treated as a single base noun phrase. For example, ' king of France' and 'ambitious student of linguistics' are single, non-recursive noun phrases.

For each supported language, RLP supplies a model of what constitutes a base noun phrase. At each point in the input, RLP detects the longest possible base noun phrase consistent with the model.

The BASE_NOUN_PHRASE [84] results consist of a pair of integers for each noun-phrase identified: index of the first token in the phrase and index + 1 of the last token in the phrase.

11.4.4. Chinese Language Analyzer

Name
CLA

Dependencies
None

Language Dependent

Chinese (Simplified and Traditional)

XML-Configurable Options

The options for the Chinese Language Processor are described by the ***BT_ROOT*/rlp/etc/cla-options.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 [196] . 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. To customize the stopwords list; see Editing the Stopwords List for Chinese, Korean, or Japanese [193] .

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 CLA processor. 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	If true, CLA 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".
`decomposecompound`	boolean	true	If true, CLA decomposes compound words into their components. A word is subject to decomposition if it is a user-dictionary entry with a decomposition pattern [197] or a noun in the CLA 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 [196] and a system dictionary in favor of the user dictionary.
`generate_all`	boolean	true	If true, all the readings for a token are returned. For single characters, these are returned in brackets, separated by semicolons.
`ignore_stopwords`	boolean	false	If false, stopwords are returned and the vector of STOPWORD results [88] is instantiated.[a]If true, tokens that are stopwords are not returned to the caller.

Property	**Type**	**Default**	**Description**
`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. CLA normally starts parsing after it detects a sentence boundary. When a limit is set, CLA 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. *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 CLA to split a valid token across buffers and not detect the token correctly.
`normalize_result_token`	false	boolean	If true, CLA generates STEM [87] 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 [86] results are calculated.

Property	Type	Default	Description
`reading_by_character`	boolean	false	If true, the reading for a polysyllabic token is determined on a per-character 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: • "tone_marks"[b] -- diacritics over the appropriate vowels • "tone_numbers" -- a number from 1-4, suffixed to each syllable • "no_tones" -- pinyin without tone presentation • "cjktex" -- pinyin generated as macros for the CJKTeX pinyin.sty style
`readings`	boolean	true	If true, READING [87] 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.
`separate_syllables`	boolean	false	If true, the syllables in the reading for a polysyllabic token are separated by a vertical line ("|").

Property	Type	Default	Description
`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	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. *Note:* the default behavior (whitespace is a numeric separator) yields different behavior than CLA versions prior to Release 4.3.

[a]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.

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

Description

The Chinese Language Processor segments Chinese text into separate tokens (words and punctuation) and assigns part-of-speech (POS) tags to each token. For the list of POS tags with examples, see Chinese POS Tags - Simplified and Traditional [210] . CLA also reports offsets for each token, and alternative readings, if any, for Hanzi or Hanzi compounds.

The Chinese Language Processor returns the following result types:

- TOKEN [88]
- TOKEN_OFFSET [88]
- PART_OF_SPEECH [86] — if `com.basistech.cla.pos` is true (the default)
- COMPOUND [84] — if `com.basistech.cla.decomposecompound` is true (the default)
- READING [87] (pinyin transcriptions) — if `com.basistech.cla.readings` is true (the default)
- STEM [87] — if `com.basistech.cla.normalize_result_token` is true (the default is false)
- STOPWORD [88] — 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 [196] .

11.4.5. Chinese Script Converter

Name
CSC

Dependencies
None

Language Dependent
Chinese (Simplified and Traditional)

XML-Configurable Options
The options for the Chinese Script Converter are defined in ***BT_ROOT*/rlp/etc/csc-options.xml**. Modify this file as necessary. A sample configuration follows:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE cscconfig SYSTEM "http://www.basistech.com/dtd/cscconfig.dtd">
<cscconfig>
  <worddictionarypath>
  <env name="root"/>/cma/dicts/zh_lex_<env name="endian"/>.bin</worddictionarypath>
  <s2tmappingdictionarypath>
<env name="root"/>/c2c/dicts/SCTTCmpt_<env name="endian"/>.bin</s2tmappingdictionarypath>
  <t2smappingdictionarypath>
<env name="root"/>/c2c/dicts/TCTSCmpt_<env name="endian"/>.bin</t2smappingdictionarypath>
</cscconfig>
```

This file conforms to the **cscconfig.dtd**:

```
<!ELEMENT cscconfig (worddictionarypath,
          s2tmappingdictionarypath, t2smappingdictionarypath)>
<!ELEMENT worddictionarypath (#PCDATA | env)*>
<!ELEMENT s2tmappingdictionarypath (#PCDATA | env)*>
<!ELEMENT t2smappingdictionarypath (#PCDATA | env)*>
<!ELEMENT env EMPTY >
<!ATTLIST env name CDATA #REQUIRED>
```

The *worddictionarypath* specifies the pathname to a dictionary used for segmentation, which is performed in the orthographic and lexemic modes of conversion as described below.

The *s2tmappingdictionarypath* specifies the pathname to a dictionary used in orthographic and lexemic conversion. from Simplified Chinese to Traditional Chinese.

The *t2smappingdictionarypath* specifies the pathname to a dictionary used in orthographic and lexemic conversion. from Traditional Chinese to Simplified Chinese.

Note that the conversion dictionaries are not in the same directory as the word dictionary.

Context Properties
The following table lists the context properties supported by the CSC processor. For brevity, the `com.basistech.csc.` prefix has been removed from the property names in the first column. For

example, the full name of `conversion_level` is `com.basistech.csc.conversion_level`.

Property	Type	Default	Description
`conversion_level`	string	"lexemic"	The type of conversion to perform. Possible values are "codepoint", "orthographic", and "lexemic", as described below.
`source_locale`	string	"cn"	Chinese-speaking region of the source text. Currently supported possible values are "cn" (People's Republic of China), "tw" (Taiwan), "hk" (Hong Kong), "sg" (Singapore), and "mo" (Macau).
`source_script`	string	"sc"	Script of the source text. Possible values are "sc" (simplified), "tc-tw" (traditional, Taiwan), "tc-hk" (traditional, Hong Kong), and "tc-cn" (traditional, People-s Republic of China).
`destination_locale`	string	"tw"	Chinese-speaking region of the destination text. Currently supported possible values are "cn" (People's Republic of China), "tw" (Taiwan), "hk" (Hong Kong), "sg" (Singapore), and "mo" (Macau).
`destination_script`	string	"tc-tw"	Script of the destination text. Possible values are "sc" (simplified), "tc-tw" (traditional, Taiwan), "tc-hk" (traditional, Hong Kong), and "tc-cn" (traditional, People-s Republic of China).

Description

There are two forms of standard written Chinese. Simplified Chinese and Traditional Chinese. Simplified Chinese (SC) is used in the People’s Republic of China (PRC). SC normally uses the GB2312-80 or GBK character set. Traditional Chinese (TC) is used in Taiwan, Hong Kong, and Macau. TC normally uses the Big Five character set. Conversion from one script to another is a complex matter. The main problem of SC to TC conversion is that the mapping is one-to-many. For example, the simplified form 发 maps to either of the traditional forms 發 or 髮. Conversion must also deal with vocabulary differences and context-dependence.

The Chinese Script Converter converts text in simplified script to text in traditional script, or vice versa. The conversion can be on any of three levels. The first is codepoint conversion, which uses a mapping table to convert characters on a codepoint-by-codepoint basis. For example, the simplified form 头发 might be converted to a traditional form by first mapping 头 to 頭, and then 发 to either 髮 or 發. Using this approach, however, there is no recognition of 头发 as a word, the choice could be 發, in which case the end result 頭發 would be nonsense. On the other hand, the choice of 髮 would lead to errors for other words. So while conversion mapping is straightforward, it is unreliable.

The second level of conversion is orthographic. This level relies upon identification of the words in a text. Within each word, orthographic variants of each character may be reflected in the conversion. In the above example, 头发 would be identified as a word. It would be converted to a traditional variant of the word, 头髮. There would be no basis for converting it to 頭發, because the conversion considers the word as a whole rather than the individual characters.

The third level of conversion is lexemic. This level also relies upon identification of words. But rather than converting a word to an orthographic variant, the aim here is to convert it to an entirely different word. For example, "computer" is usually 计算机 in SC but 電脳 in TC. Whereas codepoint conversion is strictly character-by-character and orthographic conversion is character-by-character within a word, lexemic conversion is word-by-word.

The Chinese Script Converter returns the following result types (replacing any values already set with the correct values resulting from the conversion):

- RAW_TEXT [87]
- TOKEN [88] — In the case of orthographic and lexemic conversion, the tokens are the converted words in the destination script. In the case of codepoint conversion, the tokens are the converted characters in the destination script.
- TOKEN_OFFSET [88]
- DETECTED_LANGUAGE [84]
- DETECTED_SCRIPT [85]

11.4.6. Farsi (Persian) Base Linguistics

Name
FABL

Dependencies
Tokenizer, SentenceBoundaryDetector

Language Dependent
Farsi (Persian)

XML-Configurable Options
None. The paths to the FABL dictionaries and related resources are defined in ***`BT_ROOT`*/rlp/etc/ fabl-options.xml**.

Context Properties

For brevity, the `com.basistech.fabl` prefix has been removed from the property name in the first column. Hence the full name for `variations` is `com.basistech.fabl.variations`.

Property	Type	Default	Description
`variations`	boolean	false	If set to true, FABL posts an additional result of type `TOKEN_VARIATION`.

Description

The FABL language processor performs morphological analysis for text written in Farsi (Persian). [3]

The processor generates the following result types:

- STEM [87]
- NORMALIZED_TOKEN [86]
- TOKEN_VARIATIONS [89] (if the `com.basistech.fabl.variations` property is true)

[3]We still use "Persian" in the API to refer to the language widely spoken in Iran. In a future release, we will complete the shift from "Persian" to "Farsi". This enables us to distinguish between the Persian dialects spoken in Iran (Farsi) and Afghanistan (Dari).

Normalization

Each Farsi token is normalized prior to morphological analysis. During morphological analysis, FABL may choose a token variant and normalize it to produce the value for the `NORMALIZED_TOKEN` result.

Normalization is performed in two stages: generic Arabic script normalization [125] and Farsi-specific normalization.

The following Farsi-specific normalizations are performed on the output of the Arabic script normalization:

- *Fathatan* (U+064B) and *superscript alef* (U+670) are removed.
- *Alef* أ (U+0623), إ (U+0625), or ٱ (U+0671) is converted to ا (U+0627).
- *Kaf* ك (U+0643) is converted to ک (U+06A9).
- *Heh Heh goal* (U+06C1) or *heh doachashmee* (U+06BE) is converted to *heh* (U+0647).
- *Heh with hamza* ۂ (U+06C2) is converted to ۀ (U+06C0).
- *Yeh* ي (U+064A) or ى (U+0649) is converted to ی (U+06CC).

Following morphological analysis:

- *Zero-width non joiner* (U+200C) and *superscript alef* ٰ (U+0670) are removed.

Variations

The analyzer can generate a variant form for some tokens to account for the orthographic irregularity seen in contemporary written Farsi. Each variation is generated with the normalized form:

- If a word contains *hamza on yeh* (U+626), a variant is generated replacing the *hamza on yeh* with *Farsi yeh* (U+06CC).
- If a word contains *hamza on waw* (U+0624), a variant is generated replacing the *hamza on waw* with *waw* (U+0648).
- If a word contains a *zero-width non joiner* (U+200C), a variant is generated without the *zero-width non joiner*.
- If a word ends in *teh marbuta* (U+0629), two variants are generated. The first replaces the *teh marbuta* with *teh* (U+062A); the second replaces the *teh marbuta* with *heh* (U+0647).

When the `com.basistech.fabl.variations` property setting is `true`, the generated orthographic variations for a token are returned in the `TOKEN_VARIATIONS` result. Regardless of the property setting, these variations are generated and considered in the morphological analysis.

A Note on Stemming

The stem returned in the `STEM` result is the normalized token with affixes (prefixes and suffixes) removed.

In removing prefixes and suffixes, FABL takes particular care with regard to the Unicode *zero-width non joiner* (U+200C). Farsi makes extensive use of compound words. In Farsi orthography, the components of a compound are not separated by whitespace. However, they are also not joined even when the last letter of the leading component could be joined to the first letter of the following component. The *zero-width non joiner* is used to prevent renderers from joining. When stemming, FABL does not stem compounds joined with the *zero-width non joiner*, such as روزنامه‌نگار (journalist); FABL does not remove either part of the compound.

As a general principle, words containing *zero-width non joiner* and/or *superscript alef* will have the same stem as the same words without *zero-width non joiner* or the *superscript alef*.

If `com.basistech.bl.query` is false (the default), FABL generates up to 20 orthographic variants for each token that might have other forms of spelling. For instance, words ending in *teh marbuta* could have an alternative orthographic variant with a final *teh* ت instead, especially if the borrowed word adheres to the Farsi derivational morphology rules. If the word is parseable, FABL performs morphological analysis to determime the stem. If the word is not parseable, FABL takes the next parseable alternative orthographic variant and returns its stem.

If `com.basistech.bl.query` is true, FABL does not generate orthographic variants. If the word is parseable, FABL performs morphological analysis to determine the stem. If the word is not parseable, FABL sets the stem to the value of the entire word.

11.4.7. FragmentBoundaryDetector

Name

FragmentBoundaryDetector

Dependencies

Requires tokens, which are generated by BL1 [129] , CLA [133] , JLA [147] , KLA [151] , SentenceBoundaryDetector [183] , or Tokenizer [187] .

FragmentBoundaryDetector should be placed in the RLP Context after other processors that locate sentence boundaries (BL1, Sentence Boundary Detector) and before processors that extract named entities (NamedEntityExtractor [156] , Regular Expression [163] if `com.basistech.regexp.respect_boundaries` is set to true).

Language Dependent

No

XML-Configurable Options

None

Context Properties

None

Description

Adds extra sentence boundaries [87] at tabs, newlines, and multiple whitespace characters (such as 3 or more spaces) in text fragments, such as lists and tables. The processor does not add sentence boundaries in portions of text that it judges to be standard prose.

If sentence boundaries already exist, FragmenetBoundaryDetector appends the sentence boundaries it finds. The result is a set of unique sentence boundaries in the order they appear in the document.

This processor may be useful if you are using NamedEntityExtractor to extract named entities from text that does not form sentences or that contains sections that do not form sentences. For example:

George Washington
John Adams
Thomas Jefferson

If the FragmentBoundaryDetector is not in the RLP Context, the NamedEntityExtractor tags the preceding text as a single PERSON entity. When the FragmentBoundaryDetector is in the RLP Context, this text is tagged as three separate PERSON entities.

11.4.8. Gazetteer

Name
Gazetteer

Dependencies
Tokenized text: Tokenizer or language analyzer.

Language Dependent
The processor is not language dependent, but individual gazetteers may be language specific or generic (for all languages).

XML-Configurable Options
The file ***BT_ROOT*/rlp/etc/gazetteer-options.xml** specifies normalization options and one or more gazetteers. For example:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE gazetteerconfig SYSTEM "gazetteerconfig.dtd">
<gazetteerconfig>
<option name="NormalizeCase" value="true" />
<option name="NormalizeSpace" value="true" />
<option name="NormalizeKana" value="false" />
<option name="NormalizeWidth" value="false" />
<option name="NormalizeDiacritics" value="false" />
<DictionaryPaths>
  <!-- BinDictionaryPath elements specify precompiled binary gazetteers for use by Basis Technology.
       They are not listed in this extract from the configuration file. -->

<!-- Insert your dictionaries here (each entry on a single line)
  <DictionaryPath><env name="root"/>/your/path/here</DictionaryPath>
-->
</DictionaryPaths>
</gazetteerconfig>
```

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

Normalization Options. These normalization options are applied to gazetteer entries when the gazetteers are loaded and to input text when scanned for matches.

`NormalizeCase`	If true, normalizes gazetteer entries and input text to lower case.
`NormalizeSpace`	If true, normalizes whitespace in gazetteer entries to a single space.
`NormalizeKana`	If true, convert Hiragana characters to Katakana in gazetteer entries and the input text.
`NormalizeWidth`	If true, normalizes half-width and full-width characters in the gazetteer and in the input text to "generic-width" characters.

`NormalizeDiacritics`	If true, strips diacritics and accents from gazetteer entries and input text.

Dictionary Paths. For each gazetteer, include a `DictionaryPath` element with an optional `language` attribute: the ISO639 language code for the language that the regular expression can be applied to (see ISO639 Language Codes [11]). If the attribute is left out, the gazetteer is applied to all languages.

For example,

```
<DictionaryPath language="ko"><env name="root"/>/kma/gaz/Korean-gaz.txt</DictionaryPath>
```

is used for Korean text only, whereas

```
<DictionaryPath><env name="root"/>/gaz/Generic-gaz.txt</DictionaryPath>
```

is used for text in any language.

The `BinDictionaryPath` elements define the language and path for the binary gazetteers that Basis provides for finding titles, nationality, and religion (see the Standard Set of Named Entities [47]).

Context Properties

The following table lists the context properties supported by the Gazetteer processor. Note that for brevity the `com.basistech.gazetteer` prefix has been removed from the property names in the first column. For example, the full name for `report_partial_matches` is `com.basistech.gazetteer.report_partial_matches`.

Property	**Type**	**Default**[a]	**Description**
`report_partial_matches`	boolean	false	Report matches that do not line up with token boundaries. For example, report a match for "the" in "thermal".
`space_matches_whitespace`	boolean	true	If true, the space character in gazetteer entires matches any whitespace in the input text. To normalize whitespace in gazetteer entries to a single space, set `NormalizeSpace` to true in **gazetteer-options.xml** (see above).

[a]For binary gazetteers, the default settings always apply.

Description

The Gazetteer processes text that has already passed through another language processor and isolates specific terms defined by the user in a text gazetteer or terms defined by Basis in a binary gazetteer. Users cannot read or edit the contents of a binary gazetteer. A text gazetteer has the following properties:

- UTF-8 encoded file
- Comment line(s) prefixed with #.
- The first non-comment line is the named entity type, which applies to all entries in the gazetteer, and will be used as the entity type name for output. The syntax for the name is *type:subtype*, where *subtype* is optional. If the type and subtype appear in ***BT_ROOT*/rlp/etc/ne-types.xml**, the Named

Entity Redactor [160] will use the weighting assigned in that file to resolve duplicates or overlaps. If the type does not appear in that file, it gets the default weighting of 10.

- User-defined entity strings, one per line.

For example:

```
# File: en-gazetteer.txt
#
# This is a user-defined gazetteer file.
# Gazetteer file is a UTF-8 text file.
# Comment line starts with # in the beginning of line.
# The first non-comment line is the entity type,
# which will be used to label named entities found.
# All other lines after the named entity type are gazetteer entries.
#
MESSAGE
message in a bottle
ordinary mail
```

The Gazetteer performs the following:

- Generates an internal lookup structure based on the gazetteer files specified in **gazetteer-options.xml**. Individual gazetteer files may be language specific or generic. A gazetteer that is specified to be language specific, is only applied to text in that language. A generic gazetteer is applied to text in any language. See Dictionary Paths [145] .
- Searches the input raw text for matches of entries in the gazetteer and generates NAMED_ENTITY [86] results. Each NE token consists of three integers:
  - Index of first token in the named entity
  - Index + 1 of last token in the named entity.
  - Entity type - which maps to the named entity type string that appears at the beginning of the gazetteer.

For detailed information about creating Gazetteer files, see Customizing Gazetteer [53] .

11.4.9. HTML Stripper

Name
: **HTML Stripper**

Dependencies
: None

Language Dependent
: No

XML-Configurable Options
: None

Context Properties
: None

Description

HTML input includes markup tags that degrade the accuracy of linguistic analysis. HTML Stripper strips HTML tags from the input, detects the encoding, and converts the plain text to the correct UTF-16 for the runtime platform. If the MIME type of the input is not text/html, HTML Stripper does nothing.

If the input is HTML, the HTML Stripper generates the following results: RAW_TEXT [87] and DETECTED_ENCODING [84] .

Notes On HTML Stripping

HTML Stripper strips the content of <script> elements and HTML comments (<!-- this is a comment-->), and converts character entity references (such as &amp; and &gt;) to characters.

See also the iFilter [147] processor.

11.4.10. iFilter

Name
: **iFilter**

Dependencies
: Windows only. Requires input file pathname and MIME type.

: To provide the pathname, use `BT_RLP_Context ProcessFile` (C++) or one of the `RLPContext process` methods (Java) that takes a `filename` parameter.

: To provide the MIME type, include the mime_detector [155] processor in the context before `iFilter`, or include it with the pathname in the API call to process the input.

Language Dependent
: No

XML-Configurable Options
: None

Context Properties
: None

Description
: Uses the Microsoft Indexing Service to extract plain text from the input file. The output is UTF-16 RAW_TEXT [87] . RLP recognizes the following MIME types:

- text/plain
- text/html
- text/xml
- text/rtf
- application/pdf
- application/msword
- application/vnd.ms-excel
- application/vnd.ms-powerpoint
- application/ms.access

11.4.11. Japanese Language Analyzer

Name
: **JLA**

Dependencies
None

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 [191] and create user dictionaries [199] .

```
<?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 JLA reading dictionary -->
<ReadingDictionaryPath><env name="root"/>/jma/dicts/JP_<env name="endian"/>_Reading.bin
  </ReadingDictionaryPath>

<StopwordsPath><env name="root"/>/jma/dicts/JP_stop.utf8</StopwordsPath>
</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 `StopwordsPath` specifies the pathname to the stopwords list used by the analyzer. To customize the stopwords list; see Editing the Stopwords List for Chinese, Korean, or Japanese [193] .

Context Properties

The following table lists the context properties supported by the JLA 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**
`decomposecompound`	boolean	true	If true, JLA decomposes compound words into their components. [a]
`deep_compound_decomposition`	boolean	false	If true, JLA recursively decomposes into smaller components the components marked in the dictionary as being decomposable.[a]

Property	**Type**	**Defaul t**	**Description**
`favor_user_dictionary`	boolean	false	If true, JLA favors words in the user dictionary (over the standard Japanese dictionary) during tokenization.
`generate_token_sources`	boolean	false	If true, JLA generates a TOKEN_SOURCE_ID [88] result for each token. You can use the `TOKEN_SOURCE_ID` to get the TOKEN_SOURCE_NAME [89] of the dictionary.
`ignore_separators`	boolean	true	If true, JLA ignores whitespace separators when tokenizing input text. If false, JLA treats whitespace separators as token delimiters. Note that Japanese orthography allows a newline to occur in the middle of a word.
`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 [88] is instantiated.[b]
`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. JLA normally starts parsing after it detects a sentence boundary. When a limit is set, JLA 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. *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 JLA to split a valid token across buffers and not detect the token correctly.

Property	Type	Default	Description
`normalize_result_token`	boolean	false	If true, JLA generates STEM [87] results with middle dots 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 with normalized number tokens: zenkaku (full-width) Arabic numerals such as １０，０００ are converted to half-width numerals and commas are removed (e.g., 2,000 becomes 2000); Kanji numerals are converted to half-width numerals (e.g., 四千三百 becomes 4300).
`segment_non_japanese`	boolean	true	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, JLA returns numbers and their counters as separate tokens. Warning: If you set it to false, you degrade the accuracy of the Base Noun Phrase Detector and Named Entity Extractor.
`separate_place_name_from_suffix`	boolean	true	If true, JLA 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 Detector and Named Entity Extractor.

[a]To access the components that make up the compound, use the COMPOUND [84] result.

[b]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.

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 [224] . The Japanese Language Processor returns the following result types:

- TOKEN [88]
- TOKEN_OFFSET [88]
- PART_OF_SPEECH [86]
- STEM [87] — if `com.basistech.jla.normalize_result_token` is true (the default is false)
- COMPOUND [84] — if `com.basistech.jla.decomposecompound` is true (the default)

- READING [87] (Furigana transcriptions rendered in Hiragana)
- STOPWORD [88] — if `com.basistech.jla.ignore_stopwords` is false (the default)
- TOKEN_SOURCE_ID [88] — if `com.basistech.jla.generate_token_sources` is true (the default is false)
- TOKEN_SOURCE_NAME [89] — if `com.basistech.jla.generate_token_sources` is true (the default is false)

Japanese User Dictionaries

JLA includes the capability to create and use one or more 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. For more information, see Japanese User Dictionaries [199] .

11.4.12. Korean Language Analyzer

Name

KLA

Dependencies

None

Language Dependent

Korean

XML-Configurable Options

The options for the Korean Language Processor 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 Processor, the `dictionarypath` points to the directory that contains the required dictionaries. This is different from the Japanese and Chinese Language Processor 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. To customize the stopwords list; see Editing the Stopwords List for Chinese, Korean, or Japanese [193] .

Context Properties

The following table lists the context property supported by the KLA 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**
`ignore_stopwords`	boolean	false	If false, stopwords are returned and the vector of STOPWORD results [88] is instantiated.[a]If true, tokens that are stopwords are not returned to the caller.

[a]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.

Description

The Korean Language Processor segments Korean text into separate words and compounds, reports the length of each word and the stem, and assigns a Part-of-Speech (POS) tag to each word; see Korean POS Tags [225] . KLA also returns a list of compound analyses (may be empty).

The Korean Language Processor returns the following result types:

- TOKEN [88]
- TOKEN_OFFSET [88]
- PART_OF_SPEECH [86]
- COMPOUND [84]
- STEM [87]
- STOPWORD [88] — if `com.basistech.kla.ignore_stopwords` is false (the default)

Korean User Dictionary

KLA provides a user dictionary that users can edit and recompile. For more information, see Korean User Dictionary [202] .

11.4.13. Language Boundary Detector

Name

Language Boundary

Dependencies

Text Boundary Detector, Script Boundary Detector

Language Dependent

No

XML-Configurable Options

None

Context Properties

The context properties available to control the runtime behavior of the Language Boundary Detector are listed below. As with all context properties, they can be specified either in the context or via the API. For brevity, the `com.basistech.lbd` prefix has been removed from the property names in the first column. Hence the full name for `max_region` is `com.basistech.lbd.max_region`.

Property	Type	Default	Description
`max_region`	integer	65536	The maximum number of Unicode characters to be analyzed at one time. If a potential language region (see the **Description** below) has more than this number of characters, only the first `max_region` characters are analyzed.
`min_region`	integer	20	The minimum number of Unicode characters to be analyzed at one time. If a potential language region (see the **Description** below) does not have at least `min_region` characters, it is combined with the next potential region before being analyzed.

Description

The Language Boundary Detector is used in conjunction with Script Boundary Detector [184] and Text Boundary Detector [186] to identify language regions within input text that contains multiple languages. All elements within a language region belong to the same language.

How it works. Each script region is evaluated as a potential language region. If the evaluation is ambiguous, each text region (sentence) within the script region is evaluated. Mid-sentence language boundaries are not detected unless the script changes as well.

Language Boundary Detector returns a result, LANGUAGE_REGION [85] , consisting of an array of integer sextuplets (of which three integers are reserved and currently not used). Each sextuplet consists of the beginning character offset of a region, the end offset + 1 of the region, the nesting level of the region (currently unused), the type of the region (currently unused), the script of the region (currently unused), and the language of the region.

You can use the results to submit individual regions (from the raw text) to an RLP context designed to perform linguistic processing for an individual language and script.

For more information about using the Language Boundary Detector to handle multilingual text, see Processing Multilingual Text [73] .

11.4.14. ManyToOneNormalizer

Name

ManyToOneNormalizer

Dependencies

Requires tokenization of the input text. For example, requires JLA for Japanese, CLA for Chinese, and BL1 for the European languages.

For complete coverage, requires the analyzers that produce tokens for the languages specified in the options file (described below).

Language Dependent

The languages of the normalization dictionaries specified in the options file determine the ManyToOneNormalizer's language scope.

XML-Configurable Options

The ManyToOneNormalizer uses an options file, ***BT_ROOT*/rlp/etc/normalizer-options.xml**, that specifies the language and path for each normalization dictionary. The RLP distribution includes a Japanese normalization dictionary. You can add normalization dictionaries for any of the languages supported by RLP.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE normoptions SYSTEM "normconfig.dtd">
<normoptions>
  <dictionaries>
    <dictionarypath language="ja">
        <env name="root"/>/jma/dicts/jon_<env name="endian"/>.bin</dictionarypath>
    <!-- Add additional normalizer dictionaries here -->
  </dictionaries>
</normoptions>
```

Context Properties

None

Description

For each token in the input text, the ManyToOneNormalizer performs a lookup in the normalization dictionary or dictionaries for the corresponding language and returns a MANY_TO_ONE_NORMALIZED_TOKEN [85] . The normalization dictionary contains normalized tokens and token variants. If the token does not appear as a token variant in the dictionary, the token and normalized token are identical.

With the exception of a sample Japanese normalization dictionary (described below), normalization dictionaries are user defined. See Creating Normalization Dictionaries [205] .

Sample Japanese Normalization Dictionary

The sample Japanese normalization dictionary that RLP provides supports the following normalizations:

- Normalization of words written in Katakana. Foreign and borrowed words are expressed phonetically and thus may vary in their transcription to Japanese Katakana.

  Examples:

Spelling Variants	Normalized Form
ダンスセラピ	ダンスセラピー
ダンステラピ	
ダンステラピー	
エクスポ	エキスポ
バーミューダ	バーミューダー
バミューダ	
ファミリーコンピュータ/	ファミコン
ファミリーコンピューター	
ベニス	ベネチア
ベネツィア	
ヴェネチア	
ヴェネチア	

- Normalization of older forms of Kanji to more common-use, modern Kanji forms.

  Examples:

Older Kanji Form	Normalized Form
渡邊	渡辺
國語	国語
大學	大学
關數	関数

11.4.15. mime_detector

Name
: **mime_detector**

Dependencies
: None: pathname of input file is optional.

Language Dependent
: No

XML-Configurable Options
: None

Context Properties
: The context properties available to control the runtime behavior of the mime_detector processor are listed below. For brevity, the `com.basistech.mime_detector` prefix has been removed from the property names in the first column. Hence the full name for `ignore_pathname` is `com.basistech.mime_detector.ignore_pathname`.

Property	Type	Default	Description
`ignore_pathname`	boolean	false	If true, causes `mime_detector` to ignore pathname when attempting to detect the MIME type. Set to true if file extension of input file is likely to be wrong or ambiguous.
`force_detection`	boolean	false	If true, causes `mime_detector` to run, even if MIME type has already been posted (in user API call to process the input).

Description
: Detects the MIME_TYPE [86] of the input file. Often used in conjunction with iFilter [147] or HTML Stripper [146] to extract plain text from files with markup. The mime_detector processor can use the file extension or analyze the contents to detect the following MIME types:

- text/plain
- text/html
- text/xml
- text/rtf
- application/pdf
- application/msword
- application/vnd.ms-excel
- application/vnd.ms-powerpoint
- application/ms.access

11.4.16. Named Entity Extractor

Name

NamedEntityExtractor

Dependencies

Depends on the language; see below.

Arabic	Tokenizer, SentenceBoundaryDetector, ARBL, BaseNounPhrase
Simplified Chinese	CLA, SentenceBoundaryDetector, BaseNounPhrase
Traditional Chinese	CLA, SentenceBoundaryDetector, BaseNounPhrase
Dutch	BL1, BaseNounPhrase
English	BL1, BaseNounPhrase
French	BL1, BaseNounPhrase
German	BL1, BaseNounPhrase
Italian	BL1, BaseNounPhrase
Japanese	JLA, SentenceBoundaryDetector, BaseNounPhrase
Korean	KLA, SentenceBoundaryDetector
Farsi	Tokenizer, SentenceBoundaryDetector, FABL
Russian	BL1
Spanish	BL1, BaseNounPhrase
Urdu	Tokenizer, SentenceBoundaryDetector, URBL

Language Dependent

Arabic, Simplified and Traditional Chinese, Dutch, English, French, German, Italian, Japanese, Korean, Farsi (Persian), Russian, Spanish, and Urdu.

XML-Configurable Options

The Named Entity Extractor uses language-specific binary data files to locate named entities when it is processing input. These files are not user configurable. The data file pathnames for each language are specified in the named entities configuration file: ***BT_ROOT*/rlp/etc/ne-config.xml**. Each pathname begins with `<env name="root">`. At runtime, RLP replaces this element with the pathname to the RLP root directory (***BT_ROOT*/rlp**). When you distribute an application, the location of the data files relative to RLP root should not change. See Defining an RLP Environment [17] .

Context Properties

None. The maximum number of tokens that may be included in a named entity returned by the Named Entity Extractor is defined by the `com.basistech.neredact.max_entity_tokens` (see NamedEntityRedactor [160]).

Description

A named entity is a proper noun or adjective, such as the name of a person ("George W. Bush"), an organization ("Red Cross"), a location ("Mt. Washington"), a geo-political entity ("New York"), a facility ("Fenway Park"), a nationality ("American"), a religion ("Christian"), or a title ("Professor"). The Named Entity Extractor has been statistically trained to identify entities of these eight types in some or all of the languages listed above: PERSION, ORGANIZATION, LOCATION, GPE, FACILITY, NATIONALITY, RELIGION, and TITLE. For more information, see Named Entities [45] .

The Named Entity Extractor returns a list of identified entities. Each entity is defined by three integers: index of the first token in the entity, index of the last token + 1 in the entity, and the entity type. See `NAMED_ENTITY` [86] .

Table 11.1. Named Entity Type Constants

Name	C++ Constant	Java Constant
LOCATION	BT_NE_TYPE_LOCATION	RLPNEConstants.NE_TYPE_LOCATION
ORGANIZATION	BT_NE_TYPE_ORGANIZATION	RLPNEConstants.NE_TYPE_ORGANIZATION
PERSON	BT_NE_TYPE_PERSON	RLPNEConstants.NE_TYPE_PERSON
GPE	BT_NE_TYPE_GPE	RLPNEConstants.NE_TYPE_GPE
FACILITY	BT_NE_TYPE_FACILITY	RLPNEConstants.NE_TYPE_FACILITY
RELIGION	BT_NE_TYPE_RELIGON	RLPNEConstants.NE_TYPE_RELIGION
NATIONALITY	BT_NE_TYPE_NATIONALITY	RLPNEConstants.NE_TYPE_NATIONALITY
TITLE	BT_NE_TYPE_TITLE	RLPNEConstants.NE_TYPE_TITLE

To find other entity types (such dates, email addresses, or weapons), use Gazetteer [144] and the Regular Expression [163] processors. *Note:* Gazetteer also uses binary gazetteers to identify TITLE, NATIONALITY, and RELIGION entities in several languages for which the Named Entity Extractor has not yet been trained to find TITLE.

Currently, named entity extraction is supported for the following languages: Arabic, Simplified Chinese, Traditional Chinese, Dutch, English, Upper-Case English [4] , French, German, Italian, Japanese, Korean, Farsi, Russian, Spanish, and Urdu.

11.4.16.1. Examples of Extracted Named Entities in Different Languages [5]

Language	Label	Named Entity
Arabic	LOCATION	شمال القاهرة
	ORGANIZATION	الحزب الوطني الديمقراطي
	PERSON	محمد سيد طنطاوي
	GPE	المنوفية
	FACILITY	مدرسة دانباي
	RELIGION	المسيحيين
	NATIONALITY	المصريين
	TITLE	الملك

[4]Use the `en_uc` language code when you process upper-case English text.

[5]. For the named entity types that are extracted for various languages, see the Standard Set of Named Entites [47] . For the definition of named entity types, see Named Entity Type Definitions [49] .

Language	**Label**	**Named Entity**
Simplified Chinese	LOCATION	亚洲
	ORGANIZATION	新华网
	PERSON	王祥林
	GPE	武汉市
	FACILITY	广佛地铁
	RELIGION	逊尼派
	NATIONALITY	亚裔
	TITLE	多羅郡王
Traditional Chinese	LOCATION	歐洲
	ORGANIZATION	新華網
	PERSON	王優玲
	GPE	北京
	FACILITY	布達拉宮
	RELIGION	天主教
	NATIONALITY	華人
	TITLE	多羅郡王
Dutch	LOCATION	Duitsland
	ORGANIZATION	Europese Commissie
	PERSON	Franz Beckenbauer
English	LOCATION	Mt. Washington
	ORGANIZATION	Red Cross
	PERSON	George W. Bush
	GPE	New York
	FACILITY	Fenway Park
	RELIGION	Christian
	NATIONALITY	American
	TITLE	prime minister
French	LOCATION	Mer Méditerranée
	ORGANIZATION	Organisation Européenne pour la Recherche Nucléaire
	PERSON	Eric Borgeaux
German	LOCATION	Berlin
	ORGANIZATION	Bayerische Motoren Werke
	PERSON	Karl Gauss
Italian	LOCATION	Monte Vesuvio
	ORGANIZATION	Nazioni Unite
	PERSON	Dominique Villepin

Language	**Label**	**Named Entity**
Japanese	LOCATION	硫黄島
	ORGANIZATION	朝日新聞社
	PERSON	石原
	GPE	東京都
	FACILITY	横須賀基地
	RELIGION	イスラム
	NATIONALITY	イラン人
	TITLE	女史
Korean	LOCATION	아시아
	ORGANIZATION	연합뉴스
	PERSON	이승만
	GPE	서울
	FACILITY	프라자 호텔
	TITLE	사장
Farsi	LOCATION	گرینویچ
	ORGANIZATION	بوئینگ
	PERSON	ایما کولاتا
	GPE	اندونزی
	FACILITY	فرودگاه مانادو
	RELIGION	شیعه
	NATIONALITY	هندی
RUSSIAN	LOCATION	Ниагарский водопад
	ORGANIZATION	Организация Объединенных Наций
	PERSON	Сергей Бодров-младший
	GPE	Бостон
	FACILITY	Эйфелева башня
	TITLE	Председатель
Spanish	LOCATION	Andalucía
	ORGANIZATION	El Pais
	PERSON	Manuel Pizarro

Language	Label	Named Entity
Urdu	LOCATION	جنوبی ایشیا
	ORGANIZATION	آسٹریلوی افواج
	PERSON	سعود الفیصل
	GPE	اسلام آباد
	FACILITY	امریکی کونسلیٹ
	RELIGION	مسلم
	NATIONALITY	بھارتی

11.4.17. Named Entity Redactor

Name
: **NERedactLP**

Dependencies
: Any of the following language processors: Gazetteer, NamedEntityExtractor, RegExpLP

Language Dependent
: No

XML-Configurable Options
: NERedactLP uses two configuration files, one to remove duplicates, the other to join adjacent named entities and blacklist (exclude) named entities that you are not interested in seeing.

Removing Duplicates. NERedactLP uses ***BT_ROOT*/rlp/etc/ne-types.xml** to remove duplicates when named entities are returned by more than one processor or different processors tag the same or an overlapping set of tokens as more than one named entity. For each named entity type, this file assigns three integer weight values.

Weight Name	Processor that returns the named entity
`statistical`	Named Entity Extractor [156]
`gazetteer`	Gazetteer [144]
`regex`	Regular Expression [163]

Here is a fragment from **ne-types.xml**.

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

<ne_types>
  <ne_type>
    <name>PERSON</name>
    <weight name="statistical" value="9" />
    <weight name="gazetteer" value="10" />
    <weight name="regex" value="10" />
  </ne_type>
  <ne_type>
    <name>ORGANIZATION</name>
    <subtypes>
      <name>GOVERNMENT</name>
      <name>COMMERCIAL</name>
      <name>EDUCATIONAL</name>
      <name>NONPROFIT</name>
    </subtypes>
    <weight name="statistical" value="9" />
    <weight name="gazetteer" value="10" />
    <weight name="regex" value="10" />
  </ne_type>
   <!-- other ne_types -->
   ...
   ...
</ne_types>
```

As the file is shipped, statistical weights are `9` and gazetteer and regular expression weights are `10`. Adjust these weights to instruct NeRedactLP which processor it should favor if more than one processor returns the same named entity, or which entity type it should favor if processors return different types for the same set of tokens in the input.

If, for example, you want to favor gazetteer entries over regular expressions, and favor both over values returned by statistical analysis, you could set the weights as follows:

```
    <weight name="statistical" value="0" />
    <weight name="gazetteer" value="10" />
    <weight name="regex" value="5" />
```

If different processors identify a given string as different types, processor weights determine which type is returned. If, for example, statistical (Named Entity Extractor) identifies "Foo" as an ORGANIZATION and gazetteer (Gazetteer) identifies it as MY_TYPE, the weights in the preceding example specify that gazetteer outranks statistical, so the entity is returned as MY_TYPE.

If `com.basistech.neredact.prefer_length` is true (the default), a conflict between alternative overlapping entities is resolved in favor of the longer candidate.

When you define new entity types for gazetteers and regular expressions, you should add those entity types to **ne-types.xml** if you want to control how the redactor resolves conflicts. Types that do not appear in this file receive weights of 10 for all three processors.

Apart from setting weights, it is a good idea to put the entity types you define for gazetteers and regular expressions in **ne-types.xml** so that the file also serves as a central repository for the entity types that you are using.

neredact-config.xml. NERedactLP uses ***`BT_ROOT`*****/rlp/etc/neredact-config.xml** to determine when and how to combine named entitites that are adjacent to each other, and to blacklist named entities that you do not want to see.

```
<neredactconfig>
 <joiners>
  <joiner left='TITLE' right='TITLE' joined='TITLE'/>
  <joiner left='TITLE' right='PERSON' joined='PERSON'/>
  <joiner left='PERSON' right='TITLE' joined='PERSON'/>
 </joiners>
 <blacklists>

  <blacklist type='PERSON'><env name="root"/>/blacklist/bl-person-<env name="endian"/>.bin</blacklist>
  <blacklist type='GPE'><env name="root"/>/blacklist/bl-gpe-<env name="endian"/>.bin</blacklist>
 </blacklists>
 <blacklistlog><env name="root"/>/blacklist/blacklist.log</blacklistlog>
</neredactconfig>
```

Joining Adjacent Entities. When NERedactLP joins named entitites that are adjacent to each other, the result of the join operation as a single named entity.

By default, entities are considered adjacent if they are separated by 0 to 5 whitespace characters. To override the default, a joiner may specify the `adjacency-regex` attribute, shown here with the default written explicitly:

```
<joiner left='TITLE' adjacency-regex='\s{0,5}' right='PERSON' joined='PERSON'/>
```

For example, to join "George Bush" and "President" in "George Bush, President" you can with `adjacency-regex='[\s,]+'` . NERedactLP uses the Tcl regular expression engine used by the Regular Expression [163] processor.

As shipped, **neredact-config.xml** specifies that adjacent TITLE elements are to be joined into a single TITLE element, and that TITLE and PERSON elements in either order are to be joined into a single PERSON element:

```
 <joiners>
  <joiner left='TITLE' right='TITLE' joined='TITLE'/>
  <joiner left='TITLE' right='PERSON' joined='PERSON'/>
  <joiner left='PERSON' right='TITLE' joined='PERSON'/>
 </joiners>
```

You can edit this file to add new join specifications and to edit or remove the existing join specifications. Each `joiner` element has three attributes:

- `left` -- The element that appears on the left side of the adjacent entities
- `right` -- The element that appears on the right side of the adjacent entities
- `joined` -- The element that this combination of adjacent elements produces

Blacklisting Named Entities. If you want to exclude certain entities that are sometimes returned by the Named Entity Extractor [156] , you can create one or more blacklist dictionaries. Each dictionary you create applies to a specific entity type. The Named Entity Redactor does not return entities for this type that are in this dictionary. You can instruct the Named Entity Redactor to log occurrences of blacklisted entities to a file. For information about creating blacklist dictionaries, see Blacklisting Named Entities [50] .

The following fragment specifies blacklist dictionaries for PERSON and GPE and a file (optional) where occurrences of blacklist entries are logged.

```
 <blacklists>
    <!-- No more than one dictionary per entity type-->
  <blacklist type='PERSON'><env name="root"/>/blacklist/bl-person-<env name="endian"/>.bin</blacklist>
  <blacklist type='GPE'><env name="root"/>/blacklist/bl-gpe-<env name="endian"/>.bin</blacklist>
  <blacklistlog><env name="root"/>/blacklist/blacklist.log</blacklistlog>
 </blacklists>
```

Context Properties

For brevity, the `com.basistech.neredact` prefix has been removed from the property name in the first column. Hence the full name for `prefer_length` is `com.basistech.neredact.prefer_length`.

Property	Type	Default	Description
`prefer_length`	boolean	true	If true, NERedactLP resolves a conflict between overlapping entities in favor of the longer candidate entity rather than the weight associated with each candidate entity and its source (weight is used if both candidates are of the same length). If set to false, NERedactLP uses weight to resolve the conflict.
`max_entity_tokens`	positive integer	8	The maximum number of tokens allowed in an entity returned by NamedEntityExtractor [156] . NERedactLP discards entities from NamedEntityExtractor with more than this number of tokens.

Description

A named entity is a proper noun, such as the name of a person ("Bill Gates") or an organization ("Microsoft") or a location ("New York City"). It can also be a specific date ("July 14, 1789"). Three RLP processors detect named entities: Gazetteer [144] , Named Entity Extractor [156] , and Regular Expressions [163] .

A named entity may be detected more than once or may overlap with another named entity (the same token may appear in more than one named entity). NERedactLP detects and eliminates duplication and overlapping. For each duplicate, NERedactLP assigns the named entity to a single source, using the weights assigned in ***`BT_ROOT`*****/rlp/etc/ne-types.xml**. Types and subtypes that do not appear in that file (such as types defined in gazetteers or the regular expression configuration file, and not included in **ne-types.xml**), are assigned the default weight of 10. If Gazetteer, Regular Expressions, and Named Entity Extractor have the same weight for a given named entity, the choice is arbitrary.

If alternative candidate entities overlap, the longer candidate is returned if `com.basistech.fabl.prefer_length` is true (the default). For example, "January 3, 1754 inches" contains two overlapping entities: "January 3, 1754" (TEMPORAL:DATE) and "1754 inches" (IDENTIFIER:DISTANCE). The Named Entity Redactor returns the TEMPORAL:DATE entity. On the other hand, if `com.basistech.fabl.prefer_length` is set to false and IDENTIFIER:DISTANCE is assigned a greater weight than TEMPORAL:DATE for regular expressions, the Named Redactor returns the IDENTIFIER:DISTANCE entity.

The joining of adjacent named entities, as described above, is the final step in the redaction process.

11.4.18. Regular Expression

Name

RegExpLP

Dependencies

Any tokenizing processor. FragmentBoundaryDetector [143] if `com.basistech.regexp.respect_boundaries` is set to true.

Language Dependent

The processor is not language dependent, but individual regular expressions may be language specific or generic (for all languages).

XML-Configurable Options

The regular expressions are defined in ***`BT_ROOT`*****/rlp/etc/regex-config.xml**. We suggest you take a look at this file before reading further.

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

```
<!ELEMENT regexps (regexp|define)+>
<!ELEMENT regexp (#PCDATA)>

<!ATTLIST regexp type CDATA #REQUIRED>
<!ATTLIST regexp note CDATA #IMPLIED>
<!ATTLIST regexp allow-partial-matches CDATA #IMPLIED>
<!ATTLIST regexp lang CDATA #IMPLIED>

<!ELEMENT define (#PCDATA)>
<!ATTLIST define name CDATA #REQUIRED>
<!ATTLIST define lang CDATA #IMPLIED>
```

`type`. The `type` attribute is the named-entity type to which text matched by the regular expression is assigned.

In the XML file, values take the form *`"TYPE[:SUBTYPE]"`* (such as `"IDENTIFIER"` or `"IDENTIFIER:PHONE_NUMBER"`). Regular expressions are particularly suited for identifying named entities that display a fixed pattern, such as the following:

TYPE:SUBTYPE	Description
TEMPORAL:DATE	A date
TEMPORAL:TIME	A time
IDENTIFIER:EMAIL	An email address
IDENTIFIER:URL	A URL
IDENTIFIER:IP_ADDRESS	An Internet IP address
IDENTIFIER:PHONE_NUMBER	A phone number
IDENTIFIER:PERSONAL_ID_NUM	A personal ID, such as social security number
IDENTIFIER:NUMBER	A number
IDENTIFIER:LATITUDE_LONGITUDE	Longitude and latitude
IDENTIFIER:DISTANCE	A distance

For the named entity types that are extracted for various languages by Regular Expression with the configuration file as shipped, see the Standard Set of Named Entites [47]

`lang`. The optional `lang` attribute is the ISO639 language code for the language for which the regular expression applies (see ISO639 Language Codes [11] . If the attribute is left out, the regular expression applies to all languages.

`note`. The optional `note` attribute lets you include a note about the regular expressions for your own use when maintaining this file.

`allow-partial-matches`. The optional `allow-partial-matches="yes"` attribute setting allows you to return entities that do not begin and end on token (word) boundaries. We recommend that you not use this setting. If you do, for example, and you have defined a COLOR entity of "red", a possible return value is "Frederick".

The `define` Element. The `define` element allows you to define regular expressions that you can use repeatedly by reference to its `name` attribute. To use the content in a `regexp` element, include `${name}` where *`name`* matches the `name` attribute in the `define` element. You may include the `lang` attribute in the `define` element, in which case RLP will attempt to match the `lang` attribute of the `regexp` element that contains the reference.

Example:

```
<define lang="en" name="time_ampm">(?:[pa]\.?\s?m\.?)</define>
<!-- ... stands for the rest of the regular expression in which the reference is embedded -->
<regexp lang="en" type="TEMPORAL:TIME">...${time_ampm}...<regexp>
```

If `${time_ampm}` appears in a `regexp lang="en"` element, RLP substitutes this expression. If it does not find a `define name="time_ampm lang="en"` element, RLP looks for a `define name="time_ampm` element without the `lang` attribute. If it does not find such an element, an error occurs.

Context Properties

For brevity, the `com.basistech.regexp` prefix has been removed from the property name in the first column. Hence the full name for `respect_boundaries` is `com.basistech.regexp.respect_boundaries`.

Property	Type	Default	Description
`respect_boundaries`	boolean	false	If set to true, RegExpLP evaluates the input fragment by fragment so that named entities do not cross fragment boundaries set by FragmentBoundaryDetector [143] .

Performance is slower if `com.basistech.regexp.respect_boundaries` is set to true. But if the input text contains non-prose fragments, setting this property to true may improve the results. For example, a regular expression coded as `\d+\s\d+` matches 100 000 (the space is a thousands separator). If `com.basistech.regexp.respect_boundaries` is false (the default), the expression also matches

123
456
789

If FragmentBoundaryDetector has run and the property is set to true, this expression matches each of the three numbers separately.

Description

RegExpLP lets you define regular expressions for named entities. The expressions can be associated with a single language (in which case they are only applied to documents in that language) or can be applied to all languages.

The content of each `regexp` element is a Tcl regular expression. For information about regular expression syntax, see Creating Regular Expressions [56] .

RegExpLP returns the following result type: NAMED_ENTITY [86] .

11.4.19. REXML

Name

REXML

Dependencies

None

Language Dependent

No

XML-Configurable Options

None

Context Properties

The following context properties control the runtime behavior of the REXML processor. For brevity, the `com.basistech.rexml` prefix has been removed from the property names in the first column. Hence the full name for `output_pathname` is `com.basistech.rexml.output_pathname`.

Property	**Type**	**Default**	**Description**
`output_pathname`	string		A directive to deliver output to a specified file pathname. The file is overwritten. If this property is absent or an empty string, output is delivered to standard output (stdout).
`raw_text`	string	"none"	Causes REXML to output RAW_TEXT [87] in the given format. Legal values (case-sensitive) are as follows: "cdata" -- Text is written in an XML CDATA section. "as_is" -- Text is written as is. This may result in invalid XML, if for example, the text contains &, <, or >. "escape" -- Text is written with XML entities for the following characters: &, <, >, ". "none" -- No text is written.
`suppress_header_comment`	boolean	false	Governs inclusion in the REXML output of the RLP version used to generate the file.
`token_positions`	boolean	true	A directive to deliver (or not deliver) the XML elements that specify the token offsets: the `position` element. If true, the processor delivers `position` elements as sub-elements of a token element.

Property	Type	Default	Description
`transcribed_text`	string	"none"	Causes REXML to output TRANSCRIBED_TEXT [89] in the given format. Legal values (case-sensitive) are as follows: "cdata" -- Text is written in an XML CDATA section. "as-is" -- Text is written as is. This may result in invalid XML, if for example, the text contains &, <, or >. "escape" -- Text is written with XML entities for the following characters: &, <, >, ". "none" -- No text is written.
`xsl_pathname`	string		A directive to write an XSL style sheet reference into the output REXML. Used most naturally in conjunction with `output_pathname`. Existence, validity, and proper location of the style sheet is the user's responsibility. Pathname must be in a form appropriate for XSL, not the local file system.

Description

The REXML processor converts the results of language processing into an XML format, specified by ***`BT_ROOT`*****/rlp/config/DTDs/rexml.dtd**. REXML does not generate any RLP results.

For example, suppose the following:

1. The input text is UTF-8 and consists of the single sentence, "The Patriots won."

2. The Unicode Converter has converted the text to UTF-16.

3. The Language Identifier (RLI) has determined that the language is English.

4. The Base Linguistics Analyzer (BL1) has parsed the text.

5. The REXML processor is run on the results.

REXML generates the following XML report:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rexml:document SYSTEM 'http://www.basistech.com/DTDs/rexml.dtd'>
<!DOCTYPE rexml:document SYSTEM 'http://www.basistech.com/DTDs/rexml.dtd'>
<rexml:document xmlns:rexml='http://www.basistech.com/2003/rexml'>
<header>
  <language>en</language>
<original-encoding>UTF8</original-encoding>
</header>
<contents>
<tokens>
  <token index='0'>
    <word>The</word>
  <position start='0' end='3' />
  <pos>DET</pos>
  <stem>the</stem>
 </token>
 <token index='1'>
  <word>Patriots</word>
  <position start='4' end='12' />
  <pos>NOUN</pos>
  <stem>patriot</stem>
 </token>
 <token index='2'>
  <word>won</word>
  <position start='13' end='16' />
  <pos>VPAST</pos>
  <stem>win</stem>
 </token>
 <token index='3'>
  <word>.</word>
  <position start='16' end='17' />
  <pos>SENT</pos>
  <stem>.</stem>
 </token>
</tokens>
<sentences>
 <sentence>
  <sentenceStart>0</sentenceStart>
  <sentenceEnd>4</sentenceEnd>
 </sentence>
</sentences>
</contents>
</rexml:document>
```

REXML returns no RLP result type.

11.4.20. Rosette Core Library for Unicode

Name
: **RCLU**

Dependencies
: Encoding: supplied by RLI [178] or by user. See RCLU Encodings [171]

Language Dependent
: No

XML-Configurable Options
: None

Context Properties
: The following context properties control the runtime behavior of RCLU.
All of the following context properties activate character transformations when set to the value "yes" or "true". The default setting for all these properties is "false". For brevity, the `com.basistech.rclu` prefix has been removed from the property names in the first column. Hence the full name for `BackSlashToYen` is `com.basistech.rclu.BackslashToYen`.

Property	Type	Default	Description
`BackslashToYen`	boolean	false	Converts back slash character U+005C to Yen sign U+00A5.
`BackslashToWon`	boolean	false	Converts back slash character U+005C to Won sign U+20A9.
`FormCNormalization`	boolean	false	Implements the Form C normalization forms as defined by the Unicode 3.1 standard, i.e., performs canonical decomposition followed by canonical composition.
`FormDNormalization`	boolean	false	Implements the Form D normalization forms as defined by the Unicode 3.1 standard, i.e., performs canonical decomposition.
`FormKCNormalization`	boolean	false	Implements the Form KC normalization forms as defined by the Unicode 3.1 standard, i.e., performs compatibility decomposition followed by canonical composition. If processing text in Arabic script that includes characters from Arabic Presentation Forms A (U +FB50 - U+FDFF) and/or Arabic Presentation Forms B (U+FE70 - U+FEFF), set `FormKCNormalization` to true to normalize these characters to standard Arabic script characters (U+0600 - U+06FF); if you do not, these characters are not recognized as Arabic-script characters. See also Arabic Script Normalization [125] .
`FormKDNormalization`	boolean	false	Implements the Form KD normalization forms as defined by the Unicode 3.1 standard, i.e., performs compatibility decomposition.
`FromSGMLEntity`	boolean	false	Converts SGML entities within the Unicode text to their Unicode character equivalents. For example, the string "&amp;" is converted to the Unicode ampersand character, U+0026. Hexadecimal SGML entities are also converted.
`HankakuToZenkaku`	boolean	false	Identical to ToFullWidthTransform except that this transform also takes care to combine decomposed half-width Katakana characters into their composed full-width counterparts.
`KanaToHebonRomaji`	boolean	false	This transform transliterates the Katakana and Hiragana characters in the text to Latin characters using the Hebon system of phonetic transliteration.
`KanaToKunreiRomaji`	boolean	false	This transform transliterates the Katakana and Hiragana characters in the text to Latin characters using the Kunrei system of phonetic transliteration.

Property	**Type**	**Default**	**Description**
`mapoffsets`	boolean	false	If true, and `FormCNormalization`, `FormDNormalization`, `FormKDNormalization`, or `FormKCNormalization` is true, returns `MAP_OFFSETS` [85] . Normalization may transform one Unicode character into two or three characters or vice versa. Each element in the array is the original text character index corresponding to the character of transformed text indicated by the element's position in the array. If `mapoffsets` is false or none of the transformations listed above are performed, the `MAP_OFFSETS` result is null. If `mapoffsets` is true, one of the transformations listed above is performed, and one or more additional transformations not listed above are performed, the `MAP_OFFSETS` result is undefined.
`RomajiToHiragana` *and* `RomajiToKatakana`	boolean	false	These two transforms are supplied mostly for symmetry. They attempt to convert Latin characters to a phonetic equivalent in either Hiragana or Katakana, but since this operation is only loosely defined it should not be relied upon for accurate output.
`ToCombiningMark`	boolean	false	ToCombiningMark transforms a diacritical character in its "spacing mark" form to its "combining mark" form. It is usually used in conjunction with FormC.
`ToCR`	boolean	false	Standardizes the line/paragraph separators in the text to match Macintosh standards.
`ToCRLFTransform`	boolean	false	Standardizes the line/paragraph separators in the text to match Windows standards.
`ToEBCDICNewLine`	boolean	false	Standardizes the line/paragraph separators in the text to match EBCDIC standards.
`ToFullwidth`	boolean	false	Converts half-width Latin and Katakana characters to their full-width equivalents.
`ToHalfwidth`	boolean	false	Converts full-width Latin and Katakana characters to their half-width equivalents.
`ToHiragana`	boolean	false	Converts all Japanese Katakana characters to their Hiragana equivalents.
`ToKatakana`	boolean	false	Converts all Japanese Hiragana characters to their Katakana equivalents.
`ToLargeKana`	boolean	false	Transforms small kana characters (Hiragana or Katakana) to their large equivalents.
`ToLatinNumber`	boolean	false	Converts sequences of digits in other script systems to their Latin equivalents. Special handling is provided for Japanese numbers.

Property	Type	Default	Description
`ToLF`	boolean	false	Standardizes the line/paragraph separators in the text to match Unix standards.
`ToLineSeparator`	boolean	false	Standardizes the line separators in the text to match Unicode standards.
`ToLowercase`	boolean	false	Converts letters to lower case. This is the recommended transform for case-insensitive string comparison. If a test is supplied, the transform only applies to the characters for which the test is true.
`ToParagraphSeparator`	boolean	false	Standardizes the paragraph separators in the text to match Unicode standards.
`ToSmallKana`	boolean	false	Transforms large kana characters to their small equivalents.
`ToSpacingMark`	boolean	false	Transforms a diacritical character in its "combining mark" form to its "spacing mark" form. It is usually used in conjunction with `FormDTransform`.
`ToUppercase`	boolean	false	ToUppercase transforms all lower case Latin letters to upper case (this includes both "half-width" and "full-width" Latin characters).
`YenToBackslashAndOverbarToTildeTransform`	boolean	false	Converts Yen sign U+00A5 to back slash U+005C and overbar U+203 E to tilde U+007E.
`ZenkakuToHankaku`	boolean	false	Identical to `ToHalfWidthTransform` except that this transform also takes care to divide composed full-width Katakana characters into their decomposed half-width counterparts.

Description

The RCLU language processor converts the input text to UTF-16 (RAW_TEXT [87]) as required by other language processors. RCLU also performs transformations, as determined by the context properties described above and in the order the properties are listed in the context definition. If RCLU normalizes the text and `com.basistech.rclu.mapoffsets` [170] is set to true (the default is false), RCLU returns MAP_OFFSETS [85] .

If you do not provide an encoding, RLI must precede RCLU to detect the encoding. For more information, see Language Identifier (RLI) [178] .

External Encoding	Other Names
Adobe-Standard-Encoding	csAdobeStandardEncoding
Adobe-Symbol-Encoding	csHPPSMath
Adobe-Zapf-Dingbats-Encoding	csZapfDingbats
Arabic	ISO-8859-6, csISOLatinArabic, iso-ir-127, ISO_8859-6, ECMA-114, ASMO-708
ASCII	US-ASCII, ANSI_X3.4-1968, iso-ir-6, ANSI_X3.4-1986, ISO646-US, us, IBM367, csASCII

External Encoding	Other Names
big-endian	ISO-10646-UCS-2, BigEndian, 68k, PowerPC, Mac, Macintosh, UTF-16BE
Big5	csBig5, cn-big5, x-x-big5
Big5Plus	Big5+, csBig5Plus
BMP	ISO-10646-UCS-2, BMPstring
CCSID-1027	csCCSID1027, CCSID1027, IBM1027
CCSID-1047	csCCSID1047, CCSID1047, IBM1047
CCSID-1390	csCCSID1390, CCSID1390, IBM1390
CCSID-290	csCCSID290, CCSID290, IBM290
CCSID-300	csCCSID300, CCSID300, IBM300
CCSID-930	csCCSID930, CCSID930, IBM930
CCSID-935	csCCSID935, CCSID935, IBM935
CCSID-937	csCCSID937, CCSID937, IBM937
CCSID-939	csCCSID939, CCSID939, IBM939
CCSID-942	csCCSID942, CCSID942, IBM942
ChineseAutoDetect	csChineseAutoDetect, Candidate, encodings:, GB2312, Big5, GB18030, UTF32:UTF8, UCS2, UTF32
CNS-11643	EUC-H, csCNS11643EUC, EUC-TW, TW-EUC, H-EUC, CNS-11643-1992, EUC-H-1992, csCNS11643-1992-EUC, EUC-TW-1992, TW-EUC-1992, H-EUC-1992
CNS-11643-1986	EUC-H-1986, csCNS11643_1986_EUC, EUC-TW-1986, TW-EUC-1986, H-EUC-1986
CP10000	csCP10000, windows-10000
CP10001	csCP10001, windows-10001
CP10002	csCP10002, windows-10002
CP10003	csCP10003, windows-10003
CP10004	csCP10004, windows-10004
CP10005	csCP10005, windows-10005
CP10006	csCP10006, windows-10006
CP10007	csCP10007, windows-10007
CP10008	csCP10008, windows-10008
CP10010	csCP10010, windows-10010
CP10017	csCP10017, windows-10017
CP10029	csCP10029, windows-10029
CP10079	csCP10079, windows-10079
CP10081	csCP10081, windows-10081
CP10082	csCP10082, windows-10082
CP1026	csCP1026, windows-1026
CP1250	csCP1250, windows-1250

External Encoding	Other Names
CP1251	WinCyrillic, csCP1251, windows-1251
CP1252	WinLatin1, csCP1252, windows-1252
CP1253	csCP1253, windows-1253
CP1254	csCP1254, windows-1254
CP1255	csCP1255, windows-1255
CP1256	csCP1256, windows-1256
CP1257	csCP1257, windows-1257
CP1258	csCP1258, windows-1258
CP1361	csCP1361, windows-1361
CP20105	csCP20105, windows-20105
CP20261	csCP20261, windows-20261
CP20269	csCP20269, windows-20269
CP20273	csCP20273, windows-20273
CP20277	csCP20277, windows-20277
CP20278	csCP20278, windows-20278
CP20280	csCP20280, windows-20280
CP20284	csCP20284, windows-20284
CP20285	csCP20285, windows-20285
CP20290	csCP20290, windows-20290
CP20297	csCP20297, windows-20297
CP20420	csCP20420, windows-20420
CP20423	csCP20423, windows-20423
CP20833	csCP20833, windows-20833
CP20838	csCP20838, windows-20838
CP20866	KOI8-R, KOI8, csCP20866, windows-20866
CP20871	csCP20871, windows-20871
CP20880	csCP20880, windows-20880
CP20905	csCP20905, windows-20905
CP21025	csCP21025, windows-21025
CP21027	csCP21027, windows-21027
CP21866	KOI8-RU, KOI8-U, csCP21866, windows-21866
CP28591	csCP28591, windows-28591
CP28592	csCP28592, windows-28592
CP28593	csCP28593, windows-28593
CP28594	csCP28594, windows-28594
CP28595	csCP28595, windows-28595
CP28596	csCP28596, windows-28596
CP28597	csCP28597, windows-28597

External Encoding	Other Names
CP28598	csCP28598, windows-28598
CP28599	csCP28599, windows-28599
CP38598	csCP38598, windows-38598
CP437	IBM437, 437, csPC8CodePage437, csCP437, windows-437
CP500	IBM500, csCP500, windows-500
CP708	csCP708, windows-708
CP720	csCP720, windows-720
CP737	csCP737, windows-737
CP775	csCP775, windows-775
CP850	IBM850, 850, csPC850Multilingual, csCP850, windows-850
CP852	IBM852, csCP852, windows-852
CP855	IBM855, csCP855, windows-855
CP857	IBM857, csCP857, windows-857
CP860	IBM860, csCP860, windows-860
CP861	IBM861, csCP861, windows-861
CP862	IBM862, csCP862, windows-862
CP863	IBM863, csCP863, windows-863
CP864	IBM864, csCP864, windows-864
CP865	IBM865, csCP865, windows-865
CP866	IBM866, DosCyrillic, csCP866, windows-866
CP869	IBM869, csCP869, windows-869
CP870	IBM870, csCP870, windows-870
CP874	csCP874, windows-874
CP875	csCP875, windows-875
CP936	GBK, csCP936, windows-936
CP949	csCP949, windows-949
CP950	csCP950, windows-950
csISCIIGujarati	ISCII-Gujarati
csRoman8	hp-roman8, roman8, r8, csHPRoman8
EBCDIC	IBM037, CP037, ebcdic-cp-us, ebcdic-cp-ca, ebcdic-cp-wt, ebcdic-cp-nl, csIBM037, CP37, csCP37, windows-37
EUC-JP	EUC-J, csEUCPkdFmtJapanese, Extended_UNIX_Code_Packed_Format_for_Japanese, J-EUC, JP-EUC, x-euc-jp
EUC-JP-JIS-Roman	EUC-JP, csEUCJPJISRoman
EUC-JP-JIS-RomanRoundtrip	EUC-JP, csEUCJPJISRomanRoundtrip, EUC-JP-JIS-RomanRT, csEUCJPJISRomanRT

External Encoding	Other Names
EUC-JPRoundtrip	EUC-JP, EUC-JRoundtrip, csEUCPkdFmtJapaneseRoundtrip, Extended_UNIX_Code_Packed_Format_for_JapaneseRoundtrip, J-EUCRoundtrip, JP-EUCRoundtrip, x-euc-jpRoundtrip, EUC-JPRT, EUC-JRT, csEUCPkdFmtJapaneseRT, Extended_UNIX_Code_Packed_Format_for_JapaneseRT, J-EUCRT, JP-EUCRT, x-euc-jpRT
EUC-KR	csEUCKR, KS_C_5861-1992, K-EUC
EUC-KR:HP-Printer	
GB12345	GB12345-80, GB12345-90
GB18030	GB18030, csGB18030
GB2312	GB231280, csGB2312, csGB231280, GB_2312-80, EUC-CN
Greek	ISO-8859-7, greek8, csISOLatinGreek, iso-ir-126, ISO_8859-7, ELOT_928, ECMA-118
Hebrew	ISO-8859-8, csISOLatinHebrew, iso-ir-138, ISO_8859-8
HKSCS	csHKSCS, Big5-HKSCS, csBig5-HKSCS
HZ-GB-2312	HZ, csHZGB2312
ISCII-Bengali	csISCIIBengali, x-iscii-be, windows-57003
ISCII-Devanagari	csISCIIDevanagari, x-iscii-de, windows-57002
ISCII-Kannada	csISCIIKannada, x-iscii-ka, windows-57008
ISCII-Malayalam	csISCIIMalayalam, x-iscii-ma, windows-57009
ISCII-Tamil	csISCIITamil, x-iscii-ta, windows-57004
ISCII-Telugu	csISCIITelugu, x-iscii-te, windows-57005
ISO-2022-CN	csISO2022CN
ISO-2022-JP	csISO2022JP
ISO-2022-JPRoundtrip	csISO2022JPRoundtrip, ISO-2022-JPRT
ISO-2022-KR	csISO2022KR
ISOLatinCyrillic	ISO-8859-5, Cyrillic, csISOLatinCyrillic, iso-ir-144, ISO_8859-5
JapaneseAutoDetect	csJapaneseAutoDetect, Candidate, encodings:, EUC-JP, EUC-JP-JIS-Roman, ISO-2022-JP, UTF32:UTF8, UCS2
Java	
JIS_X0201	X0201, csHalfWidthKatakana, IBM897
JIS_X_0208	JIS-X-0208, JIS_X0208-1983, csISO87JISX0208, x0208, iso-ir-87, JIS_C6226-1983
Johab	csJohab
KoreanAutoDetect	csKoreanAutoDetect, Candidate, encodings:, EUC-KR, CP949, UTF32:UTF8, UCS2
Latin1	ISO-8859-1, l1, IBM819, csISOLatin1, iso-ir-100, ISO_8859-1
Latin2	ISO-8859-2, l2, csISOLatin2, iso-ir-101, ISO_8859-2
Latin3	ISO-8859-3, l3, csISOLatin3, iso-ir-109, ISO_8859-3
Latin4	ISO-8859-4, l4, csISOLatin4, iso-ir-110, ISO_8859-4

External Encoding	**Other Names**
Latin5	ISO-8859-9, l5, csISOLatin5, iso-ir-148, ISO_8859-9
Latin6	ISO-8859-10, l6, csISOLatin6, iso-ir-157, ISO_8859-10
Latin7	iso-8859-13, l7, csISOLatin7, ISO_8859-13, ISO/IEC, 8859-13
Latin8	iso-8859-14, l8, csISOLatin8, iso-ir-199, ISO_8859-14, ISO/IEC, 8859-14
Latin9	ISO-8859-15, l9, csISOLatin9, ISO_8859-15, ISO/IEC, 8859-15
little-endian	ISO-10646-UCS-2, LittleEndian, x86, UTF-16LE
MacArabic	csMacArabic, x-mac-arabic
MacCentralEuropean	csMacCentralEuropean, MacPolish, MacCzech, MacSlovak, MacHungarian, MacEstonian, MacLatvian, MacLithuanian, x-mac-ce, x-mac-centraleurroman
MacChineseSimplified	csMacChineseSimplified, x-mac-chinesesimp
MacChineseTraditional	csMacChineseTraditional, x-mac-chinesetrad
MacCroatian	csMacCroatian, x-mac-croatian
MacCyrillic	csMacCyrillic, x-mac-cyrillic
MacDevanagari	csMacDevanagari, x-mac-devanagari
MacDingbats	csMacDingbats, x-mac-dingbats
MacGreek	csMacGreek, x-mac-greek
MacGujarati	csMacGujarati, x-mac-gujarati
MacGurmukhi	csMacGurmukhi, x-mac-gurmukhi
MacHebrew	csMacHebrew, x-mac-hebrew
MacIcelandic	csMacIcelandic, x-mac-icelandic
MacJapanese	csMacJapanese, x-mac-japanese
MacKorean	csMacKorean, x-mac-korean
MacRoman	csMacRoman, x-mac-roman
MacRomanian	csMacRomanian, x-mac-romanian
MacSymbol	csMacSymbol, x-mac-symbol
MacThai	csMacThai, x-mac-thai
MacTurkish	csMacTurkish, x-mac-turkish
MacUkrainian	csMacUkrainian, x-mac-ukrainian
NextStep	csNextStep
Shift-JIS	Shift_JIS, csShiftJISMS, csShiftJIS, CP932, csCP932, windows-932, MS_Kanji, Windows-31J, csWindows31J, SJIS, ShiftJIS, Shift, JIS, X-SJIS, x-ms-cp932, Shift-JIS-ASCII
Shift-JIS78	Shift_JIS, csShiftJIS78, SJIS78, ShiftJIS78, Shift-JIS-Roman
Shift-JIS78Roundtrip	Shift_JIS, csShiftJIS78Roundtrip, SJIS78Roundtrip, ShiftJIS78Roundtrip, Shift-JIS-RomanRoundtrip

External Encoding	**Other Names**
Shift-JISRoundtrip	csShiftJISMSRoundtrip, CP932Roundtrip, windows-932Roundtrip, MS_KanjiRoundtrip, SJISRoundtrip, ShiftJISRoundtrip, JISRoundtrip, X-SJISRoundtrip, x-ms-cp932Roundtrip, Shift-JIS-ASCIIRoundtrip, Shift-JISRT, CP932RT, windows-932RT, MS_KanjiRT, SJISRT, ShiftJISRT, Shift, JISRT, X-SJISRT, x-ms-cp932RT, Shift-JIS-ASCIIRT
Shift_JIS-2004	csShiftJIS2004, ShiftJis2004, ShiftJISX0213, Shift_JISX0213, ShiftJIS-X
TCVN	NSCII
Thai	csISOLatinThai, ISO_8859-11
UCS2	UCS-2, unicode, ISO-10646-UCS-2, UTF-16
Unicode11:big-endian	UNICODE-1-1-UCS-2
Unicode11:BOM:big-endian	UNICODE-1-1-UCS-2
Unicode11:BOM:Java	
Unicode11:BOM:little-endian	UNICODE-1-1-UCS-2
Unicode11:BOM:UCS2	UNICODE-1-1-UCS-2
Unicode11:BOM:UTF-EBCDIC	
Unicode11:BOM:UTF7	UNICODE-1-1-UTF-7
Unicode11:BOM:UTF8	UNICODE-1-1-UTF-8
Unicode11:Java	
Unicode11:little-endian	UNICODE-1-1-UCS-2
Unicode11:UCS2	UNICODE-1-1-UCS-2
Unicode11:UTF-EBCDIC	
Unicode11:UTF7	UNICODE-1-1-UTF-7
Unicode11:UTF8	UNICODE-1-1-UTF-8
Unicode20:BOM:Java	
Unicode20:BOM:UTF-EBCDIC	
Unicode20:BOM:UTF7	UTF-7
Unicode20:BOM:UTF8	UTF-8
Unicode20:little-endian	ISO-10646-UCS-2
Unicode20:UCS2	ISO-10646-UCS-2
UTF-EBCDIC	UTF8-EBCDIC, UTF-8-EBCDIC
UTF32	UTF-32
UTF32:big-endian	UTF-32
UTF32:BOM:big-endian	UTF-32
UTF32:BOM:little-endian	UTF-32
UTF32:little-endian	UTF-32
UTF32:UCS2	UTF-32

External Encoding	Other Names
UTF32:UTF8	UTF-8
UTF7	UTF-7
UTF8	UTF-8
UTF8BOM	UTF-8
VIQR	Vietnet
VISCII	viscii
VNI	
VPS	

11.4.21. Rosette Language Identifier

Name
: **RLI**

Dependencies
: None

Language Dependent
: No

XML-Configurable Options
: None

Context Properties

The following context properties control the runtime behavior of the Language Identifier. For brevity, the `com.basistech.rli` prefix has been removed from the property names in the first column. Hence the full name for `hint_language` is `com.basistech.rli.hint_language`.

Property	Type	Default	Description
`hint_language`	string		Form: "*ISOLanguageCode*, *weight*". Generates an RLI language hint. The language is denoted by a (usually) two-letter ISO639 language code [11] . The weight is a float from 1-99. The hint reduces the distance of the input profile from the specified language's profile by the specified weight (treated as a percentage). For example, "de, 50.005" reduces the distance of the input profile from the German language profile by 50.005%. Default weight is 1.0; accordingly, "nl" reduces the distance from the Dutch language profile by 1.0%. If the language is known with a very high confidence, then a large hint weight can be used to suppress language detection, leaving only encoding detection to be performed.

Property	Type	Default	Description
`min_valid_characters`	integer	4	Sets the minimum number of bytes of required valid (non-whitespace) characters in the input data. 4 bytes may include 1 to 4 UTF-8 characters. If the input data has less than the minimum number of bytes of valid characters, no detection is performed. Value range is an integer 1 or greater.
`profile_depth`	integer	1200	Sets the maximum number (depth) of n-grams to be used in input profile. If depth is 100, the 100 most frequent n-grams are included in the input profile. A small depth improves detection speed but reduces detection accuracy. Value range is an integer 1 or greater.

Description

The Language Identifier (RLI) identifies the language, encoding, and writing script of the input.

If the input is (or may be) Unicode, put Unicode Converter [187] in the context in front of RLI.

Use RLI with RCLU [168] to convert non-Unicode text to UTF-16.

RLI compares the input document against the statistical profile for every supported language and encoding (see RLI Languages and Encodings [180]).

RLI uses an n-gram algorithm for its language and encoding detection. Each built-in profile contains the quad-grams (i.e., four consecutive bytes) that are most frequently encountered in documents using a given language and encoding. The default number of n-grams is 10,000 for double-byte encodings and 5,000 for single-byte encodings. When input text is submitted for detection, a similar n-gram profile is built based on that data. The input profile is then compared with all the built-in profiles (a vector distance measure between the input profile and the built-in profile is calculated). The pre-built profiles are then returned in ascending order by the (shortest) distance of the input from the pre-built profiles.

Note: If you are interested in ranking and analyzing multiple language possibilities, rather than simply obtaining the most likely language, use the Basis Technology ***Rosette Language Identifier*** *standalone product, which allows you to set thresholds for tagging possible matches as ambiguous or invalid, to influence rankings with language-hint settings, and to iterate over all possible language matches with their respective rankings.*

RLI returns three results associated with the profile that ranks highest:

- DETECTED_LANGUAGE [84]

  An integer corresponding to one of the language IDs in the table below. The language IDs are defined in **bt_language_names.h** (C++) and **com.basistech.BTLanguageCodes** (Java).

- DETECTED_ENCODING [84]

  A string corresponding to one of the encodings in the table below.

- DETECTED_SCRIPT [85]

  An integer representing the ISO15924 code for the writing script. As noted in the following table, RLI can detect certain languages (such as Arabic, Kurdish, Pashto, Farsi, Serbian, Urdu, and Uzbek) when the text is transliterated into Latin script, in addition to text in the native script for that language.

For the ISO15924 codes, see see **bt_iso_15924_codes.h** or `com.basistech.util.ISO15924`.

Table 11.2. RLI Languages and Encodings

Language	Encoding(s)	Language ID	Code[a]
Unknown	Unknown	BT_LANGUAGE_UNKNOWN	un
Albanian	windows-1252, ISO-8859-1, UTF-8	BT_LANGUAGE_ALBANIAN	sq
Arabic	windows-1256, windows-720, ISO-8859-6, UTF-8	BT_LANGUAGE_ARABIC	ar
Arabic (Transliterated)	windows-1252, ISO-8859-1, windows-1256, UTF-8	BT_LANGUAGE_ARABIC	ar
Bengali	ISCII-Bengali, UTF-8	BT_LANGUAGE_BENGALI	bn
Bulgarian	windows-1251, ISO-8859-5, KOI8-R, UTF-8	BT_LANGUAGE_BULGARIAN	bg
Catalan	windows-1252, ISO-8859-1, UTF-8	BT_LANGUAGE_CATALAN	ca
Chinese (Simplified)	HZ-GB-2312, GB2312, ISO-2022-CN, UTF-8	BT_LANGUAGE_SIMPLIFIEDCHINESE	zh_sc
Chinese (Traditional)	Big5, UTF-8	BT_LANGUAGE_TRADITIONALCHINESE	zh_tc
Croatian	windows-1250, ISO-8859-2, UTF-8	BT_LANGUAGE_CROATIAN	hr
Czech	windows-1250, ISO-8859-2, UTF-8	BT_LANGUAGE_CZECH	cs
Danish	windows-1252, ISO-8859-1, UTF-8	BT_LANGUAGE_DANISH	da
Dutch	windows-1252, ISO-8859-1, UTF-8	BT_LANGUAGE_DUTCH	nl
English	windows-1252, ISO-8859-1, UTF-8	BT_LANGUAGE_ENGLISH	en
English (Upper Case)	windows-1252, ISO-8859-1, UTF-8	BT_LANGUAGE_ENGLISH_UC	en_uc
Estonian	windows-1257, ISO-8859-13, UTF-8	BT_LANGUAGE_ESTONIAN	et
Finnish	windows-1252, ISO-8859-1, UTF-8	BT_LANGUAGE_FINNISH	fi
French	windows-1252, ISO-8859-1, UTF-8	BT_LANGUAGE_FRENCH	fr

Language	**Encoding(s)**	**Language ID**	**Code**[a]
German	windows-1252, ISO-8859-1, UTF-8	BT_LANGUAGE_GERMAN	de
Greek	windows-1253, ISO-8859-7, UTF-8	BT_LANGUAGE_GREEK	el
Gujarati	ISCII-Gujarati, UTF-8	BT_LANGUAGE_GUJARATI	gu
Hebrew	windows-1255, ISO-8859-8, UTF-8	BT_LANGUAGE_HEBREW	he
Hindi	ISCII-Devanagari, UTF-8	BT_LANGUAGE_HINDI	hi
Hungarian	windows-1250, ISO-8859-2, UTF-8	BT_LANGUAGE_HUNGARIAN	hu
Icelandic	windows-1252, ISO-8859-1, UTF-8	BT_LANGUAGE_ICELANDIC	is
Indonesian	windows-1252, ISO-8859-1, UTF-8	BT_LANGUAGE_INDONESIAN	id
Italian	windows-1252, ISO-8859-1, UTF-8	BT_LANGUAGE_ITALIAN	it
Japanese	EUC-JP, Shift_JIS, ISO-2022-JP, UTF-8	BT_LANGUAGE_JAPANESE	ja
Kannada	ISCII-Kannada, UTF-8	BT_LANGUAGE_KANNADA	kn
Korean	EUC-KR, ISO-2022-KR, UTF-8	BT_LANGUAGE_KOREAN	ko
Kurdish (Cyrillic)	windows-1256, UTF-8	BT_LANGUAGE_KURDISH	ku
Kurdish (Latin)	windows-1252, ISO-8859-1, windows-1256, UTF-8	BT_LANGUAGE_KURDISH	ku
Latvian	windows-1257, ISO-8859-13, UTF-8	BT_LANGUAGE_LATVIAN	lv
Lithuanian	windows-1257, ISO-8859-13, UTF-8	BT_LANGUAGE_LITHUANIAN	lt
Macedonian	windows-1251, ISO-8859-5, UTF-8	BT_LANGUAGE_MACEDONIAN	mk
Malay	windows-1252, ISO-8859-1, UTF-8	BT_LANGUAGE_MALAY	ms
Malayalam	ISCII-Malayalam, UTF-8	BT_LANGUAGE_MALAYALAM	ml
Norwegian	windows-1252, ISO-8859-1, UTF-8	BT_LANGUAGE_NORWEGIAN	no
Pashto	windows-1256, UTF-8	BT_LANGUAGE_PASHTO	ps

Language	**Encoding(s)**	**Language ID**	**Code**[a]
Pashto (Transliterated)	windows-1252, ISO-8859-1, windows-1256, UTF-8	BT_LANGUAGE_PASHTO	ps
Farsi (Persian)	windows-1256, UTF-8	BT_LANGUAGE_PERSIAN	fa
Farsi (Transliterated)	windows-1252, ISO-8859-1, windows-1256, UTF-8	BT_LANGUAGE_PERSIAN	fa
Polish	windows-1250, ISO-8859-2, UTF-8	BT_LANGUAGE_POLISH	pl
Portuguese	windows-1252, ISO-8859-1, UTF-8	BT_LANGUAGE_PORTUGUESE	pt
Romanian	windows-1250, ISO-8859-2, UTF-8	BT_LANGUAGE_ROMANIAN	ro
Russian	windows-1251, ISO-8859-5, KOI8-R, IBM866, x-mac-cyrillic, UTF-8	BT_LANGUAGE_RUSSIAN	ru
Serbian (Cyrillic)	windows-1251, ISO-8859-5, UTF-8	BT_LANGUAGE_SERBIAN	sr
Serbian (Latin)	windows-1250, ISO-8859-2, UTF-8	BT_LANGUAGE_SERBIAN	sr
Slovak	windows-1250, ISO-8859-2, UTF-8	BT_LANGUAGE_SLOVAK	sk
Slovenian	windows-1250, ISO-8859-2, UTF-8	BT_LANGUAGE_SLOVENIAN	sl
Somali	windows-1252, ISO-8859-1, UTF-8	BT_LANGUAGE_SOMALI	so
Spanish	windows-1252, ISO-8859-1, UTF-8	BT_LANGUAGE_SPANISH	es
Swedish	windows-1252, ISO-8859-1, UTF-8	BT_LANGUAGE_SWEDISH	sv
Tagalog	windows-1252, ISO-8859-1, UTF-8	BT_LANGUAGE_TAGALOG	tl
Tamil	ISCII-Tamil, UTF-8	BT_LANGUAGE_TAMIL	ta
Telugu	ISCII-Telugu, UTF-8	BT_LANGUAGE_TELUGU	te
Thai	windows-874, UTF-8	BT_LANGUAGE_THAI	th
Turkish	windows-1254, UTF-8	BT_LANGUAGE_TURKISH	tr

Language	Encoding(s)	Language ID	Code[a]
Ukrainian	windows-1251, ISO-8859-5, KOI8-R, UTF-8	BT_LANGUAGE_UKRAINIAN	uk
Urdu	windows-1256, UTF-8	BT_LANGUAGE_URDU	ur
Urdu (Transliterated)	windows-1252, ISO-8859-1, windows-1256, UTF-8	BT_LANGUAGE_URDU	ur
Uzbek (Cyrillic)	windows-1251, ISO-8859-5, KOI8-R, UTF-8	BT_LANGUAGE_UZBEK	uz
Uzbek (Latin)	windows-1251, ISO-8859-5, UTF-8	BT_LANGUAGE_UZBEK	uz
Vietnamese	TCVN, VIQR, VISCII, VPS, VNI, UTF-8	BT_LANGUAGE_VIETNAMESE	vi

[a]ISO639 two-letter language codes, except for Simplified and Traditional Chinese, and upper-case English, which include Basis Technology two-letter extensions for .

11.4.22. Sentence Boundary Detector

Name

SentenceBoundaryDetector

Dependencies

Requires tokens. See the following table:

Language	Dependency
Farsi (Persian)	Tokenizer
Urdu	Tokenizer
Arabic, Farsi, Urdu	Tokenizer
Chinese (Simplified and Traditional)	CLA
Japanese	JLA
Korean	KLA
English[a]	Tokenizer
German[a]	Tokenizer
French[a]	Tokenizer
Italian[a]	Tokenizer
Spanish[a]	Tokenizer

[a]Not recommended. See below.

Language Dependent

Arabic, Simplified and Traditional Chinese, Japanese. For more accurate results with English, German, French, Italian, Spanish, use BL1 [129] to detect sentence boundaries. SentenceBoundaryDetector does nothing when it follows BL1. If you do want to use SentenceBoundaryDetector with a European language, do not include BL1.

XML-Configurable Options

None. If you use SentenceBoundaryDetector to process German or English text, it uses a dictionary for more accurate boundary detection. The paths to the dictionaries are specified in ***`BT_ROOT`*****/rlp/etc/sbd-config.xml**. These dictionaries are not user configurable.

Context Properties

None

Description

In each language, RLP detects where sentence boundaries are in documents. This is more straightforward in Japanese than in other languages, as the end-of-sentence marker in Japanese is not used for other purposes. In German and English, in addition to marking sentence boundaries the period is also used to mark abbreviations. This creates a great deal of ambiguity. Consider the following sentences:

> Mr. Smith went to N.Y.U. Law School. He then worked for John J. Jones in Mass.
> He said, "I always sail on weekends." Finally, "Or on weekdays," he added.

Note that periods need not end sentences when they end an abbreviation; however, periods can end abbreviations and at the same time end a sentence. Also they interact with other punctuation, especially quotation marks.

The Sentence Boundary Detector returns a list of SENTENCE_BOUNDARY [87] indexes representing the end token + 1 of each sentence in the input string. It also reads and, if necessary, reposts the TOKEN [88] results.

11.4.23. Script Boundary Detector

Name

Script Boundary

Dependencies

None

Language Dependent

No

XML-Configurable Options

None

Context Properties

None

Description

The Script Boundary Detector determines regions of homogeneous script within input text. That is, all characters within a script region belong to a common script. The Script Boundary Detector returns a result, SCRIPT_REGION [87] , consisting of an array of integer triples. Each triple consists of the beginning character offset of a region, the end offset + 1 of the region, and an ISO15924script code. RLP provides utilities for mapping ISO15924 integer values to 4-character codes and English names. For C++, see **bt_iso15924.h**. For Java, see `com.basistech.util.ISO15924`.

You may use the Script Boundary Detector in conjunction with the Text Boundary Detector [186] and Language Boundary Detector [152] to identify individual language and script regions within input text that contains multiple languages and scripts. You may want to use the results to submit individual regions (from the raw text) to an RLP context designed to perform linguistic processing for an individual language and script.

Given that the Script Boundary Detector is mainly used to provide input to the Language Boundary Detector, it has the following characteristics:

- Script-neutral characters like punctuation and digits do not produce a change in script.
- Hiragana and Katakana are reported as CJK script so as not to produce script changes in a region containing typical Japanese text.

For more information about using the Language Boundary Detector to handle multilingual text, see Processing Multilingual Text [73] .

11.4.24. Stopwords

Name
Stopwords

Dependencies
Tokenized text: Tokenizer or language analyzer.

Note: You cannot use the Stopwords processor with Chinese, Japanese, or Korean input. The Chinese [133] , Japanese [147] , and Korean [151] language analyzers have an `ignore_stopwords`context property that can be set to return (or not return) stopwords. The Stopwords processor does nothing if run after any of these processors.

Language Dependent
You can create stopwords [191] for Arabic, Czech, Dutch, English, French, German, Greek, Hungarian, Italian Polish, Portuguese, Russian, and Spanish.

XML-Configurable Options

The Stopwords options are specified in ***BT_ROOT*/rlp/etc/stop-options.xml**. For example:

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

<stopoptions>
<dictionaries>
<!-- Paths to your stopword dictionary files go here. Each file is a
list of words, one-per-line, in UTF-8. Stopword processing is applied
to the tokens, not the stems.
-->
<dictionarypath language="en"><env name="root"/>/etc/en-stopwords.txt
  </dictionarypath>

</dictionaries>
</stopoptions>
```

Each `dictionarypath` specifies the pathname of the user-defined stopwords file for the indicated language. The stopwords file is a list of words, one per line, encoded in UTF-8. Blank lines and lines starting with "#" are ignored.

Context Properties

Description

The Stopword processor marks tokens considered to be stopwords, based on their presence in the appropriate language-specific stopword dictionary.

The STOPWORD [88] result type contains the results: the token number of each stopword. For example, the English input sentence, "We went to the movie." has the following six tokens:

```
We
went
to
the
movie
.
```

(Note that token offsets start with 0.) Assuming that "to" and "the" are in the English stopwords file, the `STOPWORD` result will contain two entries: 2 and 3, indicating that tokens number 2 and 3 are stopwords.

In output from the REXML output processor, tokens identified as stopwords have the attribute `stopword="yes"` attached to its token element.

Stopword processing applies to the token's surface form, not to the stem. Comparison is case sensitive; e.g., "The" and "the" are different tokens.

For more detailed information about creating Stopwords files, see User-Defined Data: Customizing Stopwords [191] .

11.4.25. Text Boundary Detector

Name

Text Boundary

Dependencies

None

Language Dependent

No

XML-Configurable Options

None

Context Properties

None

Description

For use with the Script Boundary Detector [184] and Language Boundary Detector [152] to identify individual language and script regions within input text that contains multiple languages and scripts. You can use the results to submit individual regions (from the raw text) to an RLP context designed to perform linguistic processing for an individual language and script.

For more information about using the Language Boundary Detector to handle multilingual text, see Processing Multilingual Text [73] .

The Text Boundary Detector determines sentence unit boundaries as defined by Unicode Standard Annex #29 [http://www.unicode.org/reports/tr29/]. It does not search for sentence units but rather the boundaries between them. Though there are language-dependent factors here (see Sentence Boundary

Detector [183]), the processor uses Unicode requirements to find likely boundaries without knowing the language of the text. The boundaries are defined in terms of character properties. The Text Boundary Detector returns a result, TEXT_BOUNDARIES [88] , consisting of an array of character offsets. Each offset represents the end character + 1 of each sentence unit in the input text. For example, a value of 12 indicates that character 11 is the last character before the boundary.

11.4.26. Tokenizer

Name
Tokenizer

Dependencies
None

Language Dependent
No

XML-Configurable Options
None

Context Properties
None.

Description
The Tokenizer provides word tokenization functionality based on the algorithms provided in Chapter 5 of *The Unicode Standard 3.0* and *UAX #29 Text Boundaries in Unicode 4.0*.

If BL1 [129] , CLA [133] , JLA [147] , and/or KLA [151] are in the context, they should precede the Tokenizer. These language processors do their own tokenization, and the Tokenizer does nothing if one of them has already run. **DO NOT** place Tokenizer before BL1, CLA, JLA, or KLA in the context; if you do so, Tokenizer performs language-neutral tokenization, and the other processors are unable to generate language-specific tokens, part-of-speech tags, and other critical information.

ARBL [126] , FABL [141] , and URBL [188] depend on the Tokenizer (and Sentence Boundary Detector [183]).

11.4.27. Unicode Converter

Name
Unicode Converter

Dependencies
Input must be in a UTF-8, UTF-16, or UTF-32 encoding. If the user supplies the encoding in an API call to a context object, it must be one of the following strings:

- UTF-8
- UTF-8BOM
- CESU-8
- UTF-16BE
- UTF-16LE
- UTF-16
- UTF-32BE
- UTF-32LE
- UTF-32

Language Dependent
No

XML-Configurable Options
None

Context Properties
None

Description
The Unicode Converter takes text in any of the Unicode encoding forms (UTF-8, big- and little-endian UTF-16 and UTF-32) and converts it to UTF-16 (RAW_TEXT) [87] . The processor does not perform language detection.

If present, the Unicode byte-order mark (BOM) is removed prior to conversion to UTF-16; subsequent language processors in the RLP context do not see the BOM.

Sections 2.5 and 2.6 of *The Unicode Standard*, Version 4.0 discusses the Unicode encoding forms and encoding schemes at great length.

11.4.28. Urdu Base Linguistics

Name
URBL

Dependencies
Tokenizer, SentenceBoundaryDetector

Language Dependent
Urdu

XML-Configurable Options
None. The paths to the URBL dictionaries and related resources are defined in **urbl-options.xml**.

Context Properties
When processing a query (a collection of one or more search terms) rather than prose (one or more sentences), set the `com.basistech.bl.query` global context property [125] to true.

Description

The URBL language processor performs morphological analysis for texts written in Urdu.

The processor generates the following result types:

- STEM [87]
- NORMALIZED_TOKEN [86]

Normalization

Each Urdu token is normalized prior to morphological analysis. The normalized form is returned in the `NORMALIZED_TOKEN` result.

Normalization is performed in two stages: generic Arabic script normalization [125] and Urdu-specific normalization.

The following language-specific normalizations are performed on the output of the Arabic script normalization:

- *Fathatan* (U+064B), *zero-width non joiner* (U+200C), and *jazm* (U+06E1) are removed.
- *Alef* أ (U+0623), إ (U+0625), or ٱ (U+0671) is converted to ا (U+0627).
- *Kaf* ك (U+0643) is converted to ک (U+06A9).
- *Heh with hamza* ۀ (U+06C0) is converted to ۂ (U+06C2).
- *Yeh* ي (U+064A) or ى (U+0649) is converted to ی (U+06CC).
- *Small high dotless head of khah* ۡ (U+06E1) is removed.

Variations

The analyzer can generate a number of variant forms for each Urdu token to account for the orthographic irregularity seen in contemporary written Urdu. Each variation is generated over the output of the previous, starting with the normalized form:

- If a word contains *hamza on yeh* (U+0626), a variant is generated replacing the *hamza on yeh* with *Farsi yeh* (U+06CC).
- If a word contains *hamza on waw* (U+0624), a variant is generated replacing the *hamza on waw* with *waw* (U+0648).
- If a word contains a *superscript alef* (U_0670), a variant is generated without the *superscript alef*.
- If a word contains *heh doachasmee* (U+06BE), a variant is generated replacing the *heh doachasmee* with *heh goal* (U+06C1).
- If a word ends with *teh marbuta* (U+0647), a variant is generated replacing the *teh marbuta* with *heh goal* (U+06C1).

A Note on Stemming

The stem returned in the `STEM` result is the normalized token with affixes (such as prepositions, conjunctions, the definite article, proclitic pronouns, and inflectional prefixes) removed.

Normalization before stemming removes all diacritics, and performs orthographic normalization, such as converting all *alifs* to the *plain alif* or converting all non-Urdu Arabic Unicode characters such as *yeh* (U+064A) or *alif maksura* (U+0649) to the *Farsi yeh* (U +06CC).

As a general principle, words containing *zero-width non joiner* and/or *superscript alef* will have the same stem as the same words without *zero-width non joiner* or the *superscript alef*.

If `com.basistech.bl.query` is false (the default), URBL generates up to 20 orthographic variants for each token that might have other forms of spelling. If the word is parseable, URBL performs morphological analysis to determime the stem. If the word is not parseable, URBL takes the next parseable alternative orthographic variant and returns its stem.

If `com.basistech.bl.query` is true, URBL does not generate orthographic variants. If the word is parseable, URBL performs morphological analysis to determine the stem. If the word is not parseable, URBL sets the stem to the value of the entire word.

Chapter 12. User-Defined Data

Several of the RLP language processors use or even require user-defined data of some form. This chapter describes how to create and integrate user-defined data with those RLP processors:

User-Defined Data	Language Processor
Stopwords [191]	Stopwords [185]
User Dictionaries [193]	(European) Base Linguistics Analyzer (BL1) [129]
	Chinese Language Analyzer (CLA) [133]
	Japanese Language Analyzer (JLA) [147]
	Korean Language Analyzer (KLA) [151]
	(Any RLP-supported language) ManyToOneNormalizer [153]

For information about expanding or modifying the scope of Named Entities (Named Entity Extractor (NamedEntityExtractor) [156] , Gazetteer [144] , Regular Expressions (RegExpLP) [163]), see Extending the Coverage of Named Entities [52] .

12.1. Customizing Stopwords

In the context of information retrieval, a stopword is a word that appears so frequently in a language that it is statistically insignificant when performing a search. For example, in English, the words "a" and "the" appear in virtually any document, and thus do not aid in finding a specific category of documents. Some information retrieval systems choose to ignore stopwords when performing a search.

For supported languages other than Chinese, Korean, and Japanese, the Stopwords language processor uses user-defined dictionaries very similar to the Gazetteer Source Files to identify and exclude stopwords. A sample stopword dictionary for English, **en-stopwords.txt**, is in the **rlp/etc** directory. Each dictionary file is a list of words, one word per line, in UTF-8 format. Stopword processing of these files is applied to the tokens, not the stems.

For Chinese, Korean, and Japanese, the corresponding language processor uses a stopwords list in **UTF-8** encoding. See Editing the Stopwords List for Chinese, Korean, or Japanese [193] .

12.1.1. Creating Stopword Dictionaries (Not for Chinese, Korean, or Japanese)

If you are processing text in multiple languages, you may want to create a stopword file for each of the languages you intend to process. For example, you might create a simple Spanish stopword dictionary, **es-stopwords.txt**, as follows:

```
a
adonde
al
como
con
conmigo
contigo
cuando
cuanto
de
del
```

```
donde
el
ella
ellas
ellos
eres
es
esta
estamos
estan
estas
la
las
los
mas
mucho
muchos
nosotros
pero
pues
que
quien
son
soy
todo
todos
tu
tus
usted
ustedes
yo
```

This list is by no means comprehensive. As you use RLP, you may find more words that should be treated as stopwords.

12.1.2. Configuring Stopwords

The Stopwords options are specified by the **stop-options.xml** file using the **stopconfig.dtd**. For example:

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

<stopoptions>
<dictionaries>
<!-- Paths to your stopword dictionary files go here. Each file is a list
of words, one-per-line, in UTF-8. Stopword processing is applied to the
tokens, not the stems.
-->
<dictionarypath language="en"><env name="root"/>/etc/en-stopwords.txt
  </dictionarypath>
<dictionarypath language="es"><env name="root"/>/etc/es-stopwords.txt
  </dictionarypath>
</dictionaries>
</stopoptions>
```

The DTD file includes:

```
<!ELEMENT stopoptions (dictionaries)>
<!ELEMENT dictionaries (dictionarypath+)>
```

```
<!ELEMENT dictionarypath (#PCDATA)>
<!ATTLIST dictionarypath language CDATA #REQUIRED>
```

The config file will now use both the sample English stopword dictionary and your newly created Spanish stopword dictionary.

For complete information on using the Stopwords language processor, see Stopwords [185] .

12.1.3. Editing the Stopwords List for Chinese, Korean, or Japanese

The Chinese Language Analyser (CLA) [133] , Korean Language Analyzer (KLA) [151] , and Japanese Language Analyzer (JLA) [147] each use a stopwords list to define stopwords. The path to the stopwords list is specified in the processor configuration file.

Language	Configuration File	Default Stopwords Dictionary
Chinese	***BT_ROOT*/rlp/etc/cla-options.xml**	***BT_ROOT*/rlp/cma/dicts/zh_stop.utf8**
Korean	***BT_ROOT*/rlp/etc/kla-options.xml**	***BT_ROOT*/rlp/kma/dicts/kr_stop.utf8**
Japanese	***BT_ROOT*/rlp/etc/jla-options.xml**	***BT_ROOT*/rlp/jma/dicts/JP_stop.utf8**

You may want to add stopwords to these files. When you edit one of these files, you must follow these rules:

- The file must be encoded in UTF-8.
- Each line represents exactly one lexeme (stopword).
- Prefix comments with #.
- The file may include blank lines.

12.2. Creating User Dictionaries

Some of the RLP language analyzers can extract information from user dictionaries. Depending on the language, this information may include part of speech and morphological data to help the analyzer identify named entities, decompose compound forms, and perform other tasks.

For the languages supported by these analyzers, you may want to create user dictionaries for words specific to a particular industry or application.

RLP currently supports user dictionaries for the following languages:

- European Languages supported by the Base Linguistics Analyzer (BL1) [193]
- Chinese (CLA) [196]
- Japanese (JLA) [199]
- Korean (KLA) [202]

You can also create normalization dictionaries [205] for any of the languages the RLP supports.

12.2.1. European Language User Dictionaries

You can create one or more user dictionaries for each of the languages supported by Base Linguistics Analyzer (BL1) [129] : Czech, Dutch, English, Upper-Case English, French, German, Greek, Hungarian, Italian, Polish, Portuguese, Russian, and Spanish.

For optimal performance, keep the number of dictionaries you create per language to a minimum.

Procedure for Creating a User Dictionary

1. Create a source file [194] .
2. Compile the source file [195] .
3. Put the binary file in the BL1 dictionary directory for the language [195] .
4. Edit the BL1 configuration file to include the user dictionary [196] .

12.2.1.1. Creating the Source File

The source file for a user dictionary is UTF-8 encoded. The file may begin with a byte order mark (BOM). Empty lines are ignored.

Each entry is a single line. The **Tab** character separates *word* from *analysis*:

word **Tab** *analysis*

In some cases (as described below) the *word* or *analysis* may be empty.

The *word* is a rlp-results.stem TOKEN [88] . In a few cases (such as "New York"), it may contain one or more **space** characters. The three characters '[', ']', and '\' must be escaped by prefixing the character with '\'. If, for example, the *word* is 'ABC\XYZ', enter it as 'ABC\\XYZ'.

Important

BL1 user dictionary lookups occur after tokenization. If your dictionary contains a *word* like 'hi there', it will not be found because the tokenizer identifies 'hi' and 'there' as separate tokens.

The *analysis* is the STEM [87] with 0 or more morphological tags and special tags, and a required POS [86] tag. Tags are placed in square brackets ([ ]). POS tags and morphological tags begin with "+". The maximum size of an analysis is 128, where normal characters count as 1 and tags count as 2.

Morphological tags are used by the Named Entity Extractor [156] to help identify named entities (Dutch, English, French, German, Italian, and Spanish).

Special tags are used to divide compound words into their components (German, Dutch, and Hungarian) and to define boundaries for multi-word baseforms, contractions and elisions, and words with clitics (German, Dutch, Hungarian, English, French, Italian, and Portuguese).

Morphological and special tags appear in Appendix C: Morphological and Special Tags [233] .

A POS tag identifying part of speech is required and is the last (right-most) tag in the analysis. Valid POS tags appear in Appendix B: Part-of-Speech Tags [209] .

English examples:

```
dog	dog[+NOUN]
Peter	Peter[+Masc][+PROP]
NEW YORK	New[^_]York[+Place][+City][+PROP]
doesn't	does[^=]not[+VDPRES]
```

Variations: You may want to provide more than one *analysis* for a *word* or more than one version of a *word* for an *analysis*. To avoid unnecessary repetition, include lines with empty analyses (*word* + **Tab**), and lines with empty words (**Tab** + *analysis*). A line with an empty analysis uses the previous non-empty analysis. A line with an empty word uses the previous non-empty word.

The following example includes two analyses for "telephone" (noun and verb), and two renditions of "dog" for the same analysis (noun). *Note:* the dictionary lookup is case sensitive.

```
telephone  telephone[+NOUN]
           telephone[+VI]
dog        dog[+NOUN]
Dog
```

12.2.1.2. Compiling the User Dictionary

BL1 uses the dictionary in a binary form. The byte order of the binary dictionary must match the byte order of the runtime platform. To use the dictionary on both a little-endian platform (such as an Intel x86 CPU) and a big-endian platform (such as Sun's SPARC), generate two binary dictionaries. The platform on which you generate the dictionary determines the byte order of the output.

The script for generating a binary dictionary is ***BT_ROOT*/rlp/bl1/dicts/tools/build_user_dict.sh**.

Prerequisites

- Unix or Cygwin (for Windows).
- Python 2.4 on your command path.
- The Unix `sort` command on your command path.
- 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**. [1]
- The `BT_BUILD` environment variable must be set to the platform identifier embedded in your SDK package file name (see Supported Platforms [13]).[1]

To compile the dictionary into a binary format that BL1 can use, issue the following command:

```
build_user_dict.sh lang input output
```

lang is the two-letter language code (`en_uc` for Upper-Case English). See BL1 [129] .

input is the pathname of the dictionary source file.

output is the pathname of the binary dictionary file. If you are generating a little-endian and a big-endian dictionary, use *user_dict*-LE.bin for the little-endian file and *user_dict*-BE.bin for the big-endian dictionary. *Note:* choose a descriptive name for *user_dict*.

Windows example with English dictionary:

```
./build_user_dict.sh en user_dict.utf8 user_dict-LE.bin
```

Unix example with English dictionary:

```
./build_user_dict.sh en user_dict.utf8 user_dict-BE.bin
```

12.2.1.3. Where to Put the Binary Dictionary

You can put binary dictionaries where you want, but you must put the pathname to the dictionary in the BL1 configuration file (see the next section).

[1]In place of setting `BT_ROOT` and `BT_BUILD`, you can set a single environment variable (`BINDIR`) to ***BT_ROOT*/rlp/bin/*BT_BUILD***.

To organize the placement of user dictionaries, you may want to put the binary dictionary file in a ***BT_ROOT*/rlp/bl1/dicts** language directory where the directory name matches the language code. For example, put an English user dictionary in ***BT_ROOT*/rlp/bl1/dicts/en**.

12.2.1.4. Updating the BL1 Configuration File

For each user dictionary, add a `<user-dict>` element to the appropriate language section in **bl1-config.xml**.

Example: English user dictionary is available as both a little-endian and a big-endian binary dictionary.

```
<bl1-options language="en">
 ...
 ...
 <user-dict><env name="root"/>/bl1/dicts/en/userdict-<env name="endian"/>.bin</user-dict>
</bl1-options>
```

`<env-name="endian">` evaluates to LE on little-endian platforms and BE on big-endian platforms. You must compile separate dictionaries for each.

Example: German user dictionary available in only one form (little-endian or big-endian).

```
<bl1-options language="de">
 ...
 ...
 <user-dict><env name="root"/>/bl1/dicts/de/userdict.bin</user-dict>
        </bl1-options>
```

Note

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

For more information about the BL1 configuration file, see BL1 [129] .

12.2.2. 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 [197] .

2. Compile the user dictionary [198] .

3. Put the dictionary in the ***BT_ROOT*/rlp/cma/dicts** directory [199] .

4. Edit the CLA configuration file to include the user dictionary [199] .

12.2.2.1. Creating the User Dictionary

The source file for a Chinese user dictionary is UTF-8 encoded. 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 [84] results.

User Dictionary POS Tags (case-insensitive)

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

For example, the user dictionary entry

```
深圳发展銀行	organization	2,2,2
```

indicates that 東京三菱銀行 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.

12.2.2.2. Compiling the User Dictionary

CLA 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 SPARC), generate a binary dictionary on each of these platforms.

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

Prerequisites

- Unix or Cygwin (for Windows).
- Python 2.4 on your command path.
- 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 Supported Platforms [13]).

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.

12.2.2.3. Non-Compiled User Dictionaries

For backwards compatibility, CLA 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:

  ```
  深圳发展銀行	24
  ```

12.2.2.4. 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**.

12.2.2.5. Updating the CLA Configuration File

To instruct CLA 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 CLA 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.

If you are not compiling a Chinese user dictionary, you can put a reference to the source file in the CLA configuration file. For example, suppose the user dictionary is named **userdic.utf8** 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>
 <dictionarypath><env name="root"/>/cma/dicts/userdict.utf8</dictionarypath>
</claconfig>
```

12.2.3. Japanese User Dictionaries

JLA includes the capability to create and use one or more 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.

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

Procedure for Creating and Using a Japanese User Dictionary

1. Create the user dictionary [200] .
2. Compile the source file [201] .
3. Put the user dictionary in the JLA dictionary directory [202] .
4. Edit the JLA configuration file to include the user dictionary [202] .

12.2.3.1. Creating the Source File

The source file for a Japanese user dictionary is UTF-8 encoded. 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 (see TOKEN_SOURCE_NAME [89]) where JLA found each token, you must assign each user dictionary a name, and you must compile the dictionary [201] . 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*

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 [84] results.

User Dictionary POS Tags

- NOUN
- PROPER_NOUN
- PLACE
- PERSON
- ORGANIZATION
- GIVEN_NAME
- SURNAME
- FOREIGN_PLACE_NAME
- FOREIGN_GIVEN_NAME
- FOREIGN_SURNAME

Examples:

```
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
```

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 JLA to decompose this compound entry into

```
東京
証券
取引所
```

A user dictionary may also contain entries that include Private Use Area (PUA) characters. See Entering Non-Standard Characters in a User Dictionary [205] .

12.2.3.2. Compiling the User Dictionary

JLA 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 SPARC), generate a binary dictionary on each of these platforms.

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

Prerequisites

- Unix or Cygwin (for Windows).
- Python 2.4 on your command path.
- The Unix `egrep` command on your command path.
- 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 Supported Platforms [13]).

To compile the dictionary into a binary format that JLA can use, 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.

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

12.2.3.3. Non-Compiled User Dictionaries

For backwards compatibility, JLA 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
```

12.2.3.4. Where to Put the User Dictionary

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

12.2.3.5. Updating the JLA Configuration File

To use **user_dict.bin** with JLA, modify the **jla-options.xml** file to include it. For example, if you put your user dictionary 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 user dictionary -->
  <DictionaryPath>
    <env name="root"/>/jma/dicts/user_dict.bin
  </DictionaryPath>
</DictionaryPaths>
```

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 JLA 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>
```

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.

12.2.4. Korean User Dictionary

Korean Language Analyzer (KLA) [151] 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 KLA options file [151] `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 [203] .
2. Recompile the dictionary [204] .

12.2.4.1. 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 stem 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). KLA uses a decomposition algorithm to decompose compound nouns that contain no *DecompPattern*. The individual components that make up the compound are in the COMPOUND [84] 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

POS	Meaning
B	Adverb
D	Determiner
L	Interjection (exclamation)

Examples:

```
개배때기    N
그러더니    B
그러던    D
퀘이    TC
개인홈페이지    c
경품대축제    c    2,3
```

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

```
경품
대축제
```

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

12.2.4.2. 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).
- Python 2.4 on your command path.
- The Unix `sort` command on your command path.
- The `BT_ROOT` environment variable must be set to *`BT_ROOT`*, the Basis Technology root directory. For example, if JLA 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 JLA package name, such as `ia32-glibc22-gcc32`. For a list of the `BT_BUILD` values, see Supported Platforms and BT_BUILD Values [13] .

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.

12.2.4.3. 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 [151] . 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>
```

12.2.5. Entering Non-Standard Characters in a Japanese User Dictionary

In a Japanese user dictionary, you may want to include terms that include Unicode Private Use Area (PUA) characters for user-defined characters (UDC) .

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

12.2.6. Creating Normalization Dictionaries

If your context configuration includes the ManyToOneNormalizer [153] , that processor uses any normalization dictionaries specified in ***BT_ROOT*/rlp/etc/normalizer-options.xml** for the language of the input text to return a MANY_TO_ONE_NORMALIZED_TOKEN [85] for each token in the input text.

Use one or more normalization dictionaries if you want to normalize word variants to their standard representation.

A Sample Japanese Normalization Dictionary

The RLP SDK includes a compiled sample Japanese normalization dictionary [154] . You can also create and use your own Japanese normalization dictionaries.

Important

When you create a Japanese, Chinese, or Korean normalization dictionary, you should also add the variants to a Japanese, Chinese or Korean user dictionary. If you do not, JLA, CLA, or KLA may not be able to tokenize the variants correctly, given the absence of spaces between words in Chinese and Japanese text and the variable use of spaces in Korean text.

Procedure for Creating and Using a Normalization Dictionary

1. Create the normalization dictionary [206] .

2. Compile the source file [206] .

3. Put the normalization dictionary in the appropriate dictionary directory [207] .
4. Edit the ManyToOneNormalizer configuration file to include the normalization dictionary [207] .

12.2.6.1. Creating the Normalization Dictionary

The source file for a normalization dictionary is UTF-8 encoded. 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 containing a normalized word and one or more word variants:

normalized_word **Tab** *variant_word_1* **Tab** *variant_word_2* ...

where *normalized_word* is the normalized word and each *variant_word_n* is a word variant that you want to normalize.

The following entry in an English normalization dictionary normalizes *manoeuvre* or *manoeuver* to *maneuver* (the standard US spelling):

```
maneuver	manoeuvre	manoeuver
```

12.2.6.2. Compiling the Source File

The ManyToOneNormalizer 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 SPARC), generate a binary dictionary on each of these platforms.

The script for generating a binary dictionary is ***BT_ROOT*/rlp/rlp/tools/build_normalize_dict.sh**.

Prerequisites

- Unix or Cygwin (for Windows).
- Python 2.4 on your command path.
- The Unix `uniq` command on your command path.
- 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 Supported Platforms [13]).

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

```
build_normalize_dict.sh input output
```

For example, if you have a German normalization dictionary in ***BT_ROOT*/rlp/bl1/dicts/de** named **normalize_dict_de.utf8**, build the binary normalization dictionary **normalize_dict_de.bin** with the following command in the ***BT_ROOT*/rlp/rlp/tools** directory:

```
./build_normalize_dict.sh ../../bl1/dicts/de/normalize_dict_de.utf8 ../../bl1/dicts/de/normalize_dict_de.bin
```

Note

If you are making the normalization 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.

12.2.6.3. Where to Put the Normalization Dictionary

You can put the binary dictionary where you want, but you must put the pathname to the dictionary in the ManyToOneNormalizer configuration file (see the next section).

To organize the placement of normalization dictionaries, you may want to put the binary dictionary in the **dicts** directory for the processor that tokenizes that language, or, in the case of BL1, in a subdirectory that matches the language code for the dictionary. So for example, you could place a Chinese normalization dictionary in ***BT_ROOT*/rlp/cma/dicts** and a German normalization dictionary in ***BT_ROOT*/rlp/bl1/dicts/de**.

12.2.6.4. Updating the ManyToOneNormalizer Options File

To instruct ManyToOneNormalizer to use your normalization dictionary, add a `<dictionarypath>` element to **normalize-options.xml**. For example, suppose the binary normalization dictionary is named **normalize_dict_de.bin** and is in ***BT_ROOT*/rlp/bl1/dicts/de**. Modify ***BT_ROOT*/rlp/etc/normalize-options.xml** to include the new `<dictionarypath>` element.

```
<normoptions>
 <dictionaries>
  ...
  ...
  <dictionarypath language="de">
   <env name="root"/>/bl1/dicts/de/normalize_dict_de.bin</dictionarypath>
 </dictionaries>
</normoptions>
```

For a list of the language codes you can use, see ISO639 Language Codes [11] .

If you are making the normalization 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 CLA configuration file to choose the correct binary for the runtime platform:

```
<normoptions>
 <dictionaries>
  ...
  ...
   <dictionarypath language="de">
 <env name="root"/>/cma/dicts/normalize_dict_de_<env name="endian"/>.bin</dictionarypath>
 </dictionaries>
</normoptions>
```

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 normalization dictionaries in the options file. If you have multiple normalization dictionaries for a given language, and the same variant appears in more than one dictionary, the ManyToOneNormalizer returns the first normalization it finds.

Important

To include many-to-one normalization when you process data, your context configuration must include the ManyToOneNormalizer, as illustrated in the sample context configuration file in Unicode Input and Base Linguistics Analysis for One Language [20] . In the context configuration, ManyToOneAnalyzer must appear after the analyzers that tokenize text for the languages specified in the options file (such as JLA for Japanese, CLA for Chinese, and BL1 for English).

Appendix A. Part-of-Speech Tags

A.1. Arabic POS Tags

Tag	Description	Example
ABBREV	abbreviation	ا ف ب
ADJ	adjective	اَلأَمْرِيكِيّ، عَرَبِي
ADV	adverb	هُنَاكَ، ثُمَّ
CONJ	conjunction	وَ
CV	verb (imperative)	أَضِفْ
DEM_PRON	demonstrative pronoun	هٰذَا
DET	determiner	لِل
EOS	end of sentence	! ؟ .
EXCEPT_PART	exception particle	إلا
FOCUS_PART	focus particle	أما
FUT_PART	future particle	سَوْفَ
INTERJ	interjection	آه
INTERROG_PART	interrogative particle	هَلْ
IV	verb (imperfect)	يَكْتُبُ، يَأْكُلُ
IV_PASS	verb (passive imperfect)	يُضَافُ، يُشَارُ
NEG_PART	negative particle	لَن
NON_ARABIC	not Arabic script	a b c
NOUN	noun	طَائِرْ، كُمْبِيُوتَرْ، بَيْتْ
NOUN_PROP	proper noun	طُونِي، مُحَمَّدْ
NO_FUNC	unknown part of speech	
NUM	numbers (Arabic-Indic numbers, Latin, and text-based cardinal)	أَرْبَعَة عَشَرْ، ١٤، 14
PART	particle	أَيَّتُهَا، إِيَّاهُ
PREP	preposition	أَمَامْ، فِي
PRONOUN	pronoun	هُوَ
PUNC	punctuation	( ) ؛.،؟:

Tag	Description	Example
PV	perfective verb	كَانَت، قَالَ
PV_PASS	passive perfective verb	أُعْتَبَرَ
RC_PART	resultative clause particle	فَلَمَا
REL_ADV	relative adverb	حَيْثُ
REL_PRON	relative pronoun	اَلَّذِي، اَللَّذَان
SUB_CONJ	subordinating conjunction	إِذَا، إِذ
VERB_PART	verbal particle	لَقَدْ

A.2. Chinese POS Tags - Simplified and Traditional

Tag	Description	Simplified Chinese	Traditional Chinese
A	adjective	可爱	可愛
D	adverb	必定	必定
E	idiom/phrase	胸有成竹	胸有成竹
EOS	sentence final punctuation	。	。
F	non-derivational affix	鸳	鴛
I	interjection	吧	吧
J	conjunction	但是	但是
M	onomatope	丁丁	丁丁
NA	abbreviation	日	日
NC	common noun	水果	水果
NM	measure word	个	個
NN	numeral	３, 2, 一	３, 2, 一
NP	proper noun	英国	英國
NR	pronoun	我	我
NT	temporal noun	一月	一月
OC	construction	越～越～	越～越～
PL	particle	之	之
PR	preposition	除了	除了
PUNCT	non-sentence-final punctuation	，「」（）；	，《》（）
U	unknown		
V	verb	跳舞	跳舞
W	derivational suffix	家	家
WL	direction word	下	下

Tag	Description	Simplified Chinese	Traditional Chinese
WV	word element - verb	以	以
X	generic affix	老	老
XP	generic prefix	可	可
XS	generic suffix	员	員

A.3. Czech POS Tags

Tag	Description	Example
ADJ	adjective: nominative	[vál] silný [vítr]
	adjective: genitive	[k uvedení] zahradní [slavnosti]
	adjective: dative	[k] veselým [lidem]
	adjective: accusative	[jak zdolat] ekonomické [starosti] [vychutná] jeho [radost]
	adjective: instrumental	první bushovou [zastávkou]
	adjective: locative	[na] druhém [okraji silnice]
	adjective: vocative	ty rudá [krávo]
	ordinal	[obsadil] 12. [místo]
ADV	adverb	velmi, nejvíce, daleko, jasno
CLIT	clitic	bych, by, bychom, byste
CM	comma	,
CONJ	conjunction	a, i, ale, aby, nebo, však, protože
DATE	date	11. 12. 1996, 11. 12.
INTJ	interjection	ehm, ach
NOUN	noun: nominative	[je to] omyl
	noun: genitive	[krize] autority státu
	noun: dative	[dostala se k] moci
	noun: accusative	[názory na] privatizaci
	noun: instrumental	[Marx s naprostou] jistotou
	noun: locative	[ve vlastním] zájmu
	noun: vocative	[ty] parlamente
	abbreviation, initial, unit	v., mudr., km/h, m3
NUM_ACC	numeral: accusative	[máme jen] jednu [velmoc]
NUM_DAT	numeral: dative	[jsme povinni] mnoha [lidem]
NUM_DIG	digit	123, 2:0, 1:23:56, -8.9, -8 909
NUM_GEN	numeral: genitive	[po dobu] dvou [let]
NUM_INS	numeral: instrumental	[s] padesáti [hokejisty]
NUM_LOC	numeral: locative	[po] dvou [závodech]

Tag	Description	Example
NUM_NOM	numeral: nominative	oba [kluby tají, kde]
NUM_ROM	Roman numeral	V
NUM_VOC	numeral: vocative	[vy] dva [, zastavte]
PREP	preposition	dle [tebe], ke [stolu], do [roku], se [mnou]
PREPPRON	prepositional pronoun	nač
PRON_ACC	pronoun: accusative	[nikdo] je [nevyhodí]
PRON_DAT	pronoun: dative	[kdy je] mu [vytýkána]
PRON_GEN	pronoun: genitive	[u] nás [i kolem] nás
PRON_INS	pronoun: instrumental	[mezi] nimi [být]
PRON_LOC	pronoun: locative	[aby na] ní [stál]
PRON_NOM	pronoun: nominative	já [jsem jedinou]
PRON_VOC	pronoun: vocative	vy [dva, zastavte]
PROP	proper noun	Pavel, Tigrid, Jacques, Rupnik, Evropy
PTCL	particle	ano, ne
PUNCT	punctuation	( ) { } [ ] ;
REFL_ACC	reflexive pronoun: accusative	se
REFL_DAT	reflexive pronoun: dative	si
REFL_GEN	reflexive pronoun: genitive	sebe
REFL_INS	reflexive pronoun: instrumental	sebou
REFL_LOC	reflexive pronoun: locative	sobě
SENT	sentence final punctuation	. ! ? ...
VERB_IMP	verb: imperative	odstupte !
VERB_INF	verb: infinitive	[mohli si] koupit
VERB_PAP	verb: past participle	mohli [si koupit]
VERB_PRI	verb: present indicative	[trochu nás] mrzí
VERB_TRA	verb: transgressive	maje [ode mne]

A.4. Dutch POS Tags

Tag	Description	Example
ADJA	attributive adjective	[een] snelle [auto]
ADJD	adverbial or predicative adjective	[hij rijdt] snel
ADV	non-adjectival adverb	[hij rijdt] vaak
ART	article	een [bus], het [busje]
CARD	cardinals	vijf
CIRCP	right part of circumposition	[hij viel van dit dak] af
CM	comma	,
CMPDPART	right truncated part of compound	honden- [kattenvoer]

Tag	**Description**	**Example**
COMCON	comparative conjunction	[zo groot] als, [groter] dan
CON	co-ordinating conjunction	[Jan] en [Marie]
CWADV	interrogative adverb or subordinate conjunction	wanneer [gaat hij weg ?], wanneer [hij nu weggaat]
DEMDET	demonstrative determiner	deze [bloemen zijn mooi]
DEMPRO	demonstrative pronoun	deze [zijn mooi]
DIG	digits	1, 1.2
INDDET	indefinite determiner	geen [broer]
INDPOST	indefinite postdeterminer	[de] beide [broers]
INDPRE	indefinite predeterminer	al [de broers]
INDPRO	indefinite pronoun	beide [gingen weg]
INFCON	infinitive conjunction	om [te vragen]
ITJ	interjections	Jawel, och, ach
NOUN	common noun or proper noun	[de] hoed, [het goede] gevoel, [de] Betuwelijn
ORD	ordinals	vijfde, 125ste, 12de
PADJ	postmodifying adjective	[iets] aardigs
PERS	personal pronoun	hij [sloeg] hem
POSDET	possessive pronoun	mijn [boek]
POSTP	postposition	[hij liep zijn huis] in
PREP	preposition	[hij is] in [het huis]
PROADV	pronominal adverb	[hij praat] hierover
PTKA	adverb modification	[hij wil] te [snel]
PTKNEG	negation	[hij gaat] niet [snel]
PTKTE	infinitive particle	[hij hoopt] te [gaan]
PTKVA	separated prefix of pronominal adverb or verb	[daar niet] mee [hij loopt] mee
PUNCT	other punctuation	" ' ` { } [ ] < > - ---
RELPRO	relative pronoun	[de man] die [lachte]
RELSUB	relative conjunction	[Het kind] dat, [Het feit] dat
SENT	sentence final punctuation	; . ?
SUBCON	subordinating conjunction	Hoewel [hij er was]
SYM	symbols	@, %
VAFIN	finite auxiliary verb	[hij] is [geweest]
VAINF	infinite auxiliary verb	[hij zal] zijn
VAPP	past participle auxiliary verb	[hij is] geweest
VVFIN	finite substantive verb	[hij] zegt
VVINF	infinitive substantive verb	[hij zal] zeggen
VVPP	past participle substantive verb	[hij heeft] gezegd

Tag	Description	Example
WADV	interrogative adverb	waarom [gaat hij]
WDET	interrogative or relative determiner	[de vrouw] wier [man....]
WPRO	interrogative or relative pronoun	[de vraag] wie ...

A.5. English POS Tags

Tag	Description	Example
ADJ	(basic) adjective	[a] blue [book], [he is] big
ADJCMP	comparative adjective	[he is] bigger, [a] better [question]
ADJING	adjectival ing-form	[the] working [men]
ADJPAP	adjectival past participle	[a] locked [door]
ADJPRON	pronoun (with determiner) or adjective	[the] same; [the] other [way]
ADJSUP	superlative adjective	[he is the] biggest; [the] best [cake]
ADV	(basic) adverb	today, quickly
ADVCMP	comparative adverb	sooner
ADVSUP	superlative adverb	soonest
CARD	cardinal (except *one*)	two, 123, IV
CARDONE	cardinal one	[at] one [time] ; one [dollar]
CM	comma	,
COADV	coordination adverbs *either, neither*	either [by law or by force]; [he didn't come] either
COORD	coordinating conjunction	and, or
COSUB	subordinating conjunction	because, while
COTHAN	conjunction *than*	[bigger] than
DET	determiner	the [house], a [house], this [house], my [house]
DETREL	relative determiner *whose*	[the man] whose [hat ...]
INFTO	infinitive marker *to*	[he wants] to [go]
ITJ	interjection	oh!
MEAS	measure abbreviation	[50] m. [wide], yd
MONEY	currency plus cardinal	$1,000
NOT	negation *not*	[he will] not [come in]
NOUN	common noun	house
NOUNING	nominal ing-form	[the] singing [was pleasant], [the] raising [of the flag]
ORD	ordinal	3rd, second
PARTPAST	past participle (in subclause)	[while] seated[, he instructed the students]; [the car] sold [on Monday]
PARTPRES	present participle (in subclause), gerund	[while] doing [it];[they began] designing [the ship];having [said this ...]

Tag	**Description**	**Example**
POSS	possessive suffix *'s*	[Peter] 's ; [houses] '
PREDET	pre-determiner *such*	such [a way]
PREP	preposition	in [the house], on [the table]
PREPADVAS	preposition or adverbial *as*	as [big] as
PRON	(non-personal) pronoun	everybody, this [is] mine
PRONONE	pronoun *one*	one [of them]; [the green] one
PRONPERS	personal pronoun	I, me, we, you
PRONREFL	reflexive pronoun	myself, ...
PRONREL	relative pronoun *who, whom, whose; which; that*	[the man] who [wrote that book], [the ship] that capsized
PROP	proper noun	Peter, [Mr.] Brown
PUNCT	punctuation (other than SENT and CM)	"
QUANT	quantifier *all, any, both, double, each, enough, every, (a) few, half, many, some*	many [people]; half [the price]; all [your children]; enough [time]; any [of these]
QUANTADV	quantifier or adverb *much, little*	much [better] , [he cares] little
QUANTCMP	quantifier or comparative adverb *more, less*	more [people], less [expensive]
QUANTSUP	quantifier or superlative adverb *most, least*	most [people], least [expensive]
SENT	sentence final punctuation	. ! ? :
TIT	title	Mr., Dr.
VAUX	auxiliary (modal)	[he] will [run], [I] won't [come]
VBI	infinitive or imperative of *be*	[he will] be [running]; be [quiet!]
VBPAP	past participle of *be*	[he has] been [there]
VBPAST	past tense of *be*	[he] was [running], [he] was [here]
VBPRES	present tense of *be*	[he] is [running], [he] is [old]
VBPROG	ing-form of *be*	[it is] being [sponsored]
VDI	infinitive of *do*	[He will] do [it]
VDPAP	past participle of *do*	[he has] done [it]
VDPAST	past tense of *do*	[we] did [it], [he] didn't [come]
VDPRES	present tense of *do*	[We] do [it], [he] doesn't [go]
VDPROG	ing-form of *do*	[He is] doing [it]
VHI	infinitive or imperative of *have*	[he would] have [come]; have [a look!]
VHPAP	past participle of *have*	[he has] had [a cold]
VHPAST	past tense of *have*	[he] had [seen]
VHPRES	present tense of *have*	[he] has [been watching]
VHPROG	ing-form of *have*	[he is] having [a good time]
VI	verb infinitive or imperative	[he will] go, [he comes to] see; listen [!]

Tag	**Description**	**Example**
VPAP	verb past participle	[he has] played, [it is] written
VPAST	verb past tense	[I] went, [he] loved
VPRES	verb present tense	[we] go, [she] loves
VPROG	verb ing-form	[you are] going
VS	verbal *'s* (short for is or has)	[he] 's [coming]
WADV	interrogative adverb	when [did ...], where [did ...], why [did ...]
WDET	interrogative determiner	which [book], whose [hat]
WPRON	interrogative pronoun	who [is], what [is]

A.6. French POS Tags

Tag	**Description**	**Example**
ADJ2_INV	special number invariant adjective	gros
ADJ2_PL	special plural adjective	petites, grands
ADJ2_SG	special singular adjective	petit, grande
ADJ_INV	number invariant adjective	heureux
ADJ_PL	plural adjective	gentils, gentilles
ADJ_SG	singular adjective	gentil, gentille
ADV	adverb	finalement, aujourd'hui
CM	comma	,
COMME	reserved for the word *comme*	comme
CONJQUE	reserved for the word *que*	que
CONN	connector subordinate conjunction	si, quand
COORD	coordinate conjunction	et, ou
DET_PL	plural determiner	les
DET_SG	singular determiner	le, la
MISC	miscellaneous	miaou, afin
NEG	negation particle	ne
NOUN_INV	number invariant noun	taux
NOUN_PL	plural noun	chiens, fourmis
NOUN_SG	singular noun	chien, fourmi
NUM	numeral	treize, 13, XIX
PAP_INV	number invariant past participle	soumis
PAP_PL	plural past participle	finis, finies
PAP_SG	singular past participle	fini, finie
PC	clitic pronoun	[donne-]le, [appelle-]moi, [donne-]lui
PREP	preposition (other than à, au, de, du, des)	dans, après
PREP_A	preposition "à"	à, au, aux

Tag	**Description**	**Example**
PREP_DE	preposition "de"	de, d', du, des
PRON	pronoun	il, elles, personne, rien
PRON_P1P2	1st or 2nd person pronoun	je, tu, nous
PUNCT	punctuation (other than comma)	: -
RELPRO	relative/interrogative pronoun (except "que")	qui, quoi, lequel
SENT	sentence final punctuation	. ! ? ;
SYM	symbols	@ %
VAUX_INF	infinitive auxiliary	être, avoir
VAUX_P1P2	1st or 2nd person auxiliary verb, any tense	suis, as
VAUX_P3PL	3rd person plural auxiliary verb, any tense	seraient
VAUX_P3SG	3rd person singular auxiliary verb, any tense	aura
VAUX_PAP	past participle auxiliary	eu, été
VAUX_PRP	present participle auxiliary verb	ayant
VERB_INF	infinitive verb	danser, finir
VERB_P1P2	1st or 2nd person verb, any tense	danse, dansiez, dansais
VERB_P3PL	3rd person plural verb, any tense	danseront
VERB_P3SG	3rd person singular verb, any tense	danse, dansait
VERB_PRP	present participle verb	dansant
VOICILA	reserved for *voici*, *voilà"*	voici, voilà

A.7. German POS Tags

Tag	**Description**	**Example**
ADJA	(positive) attributive adjective	[ein] schnelles [Auto]
ADJA2	comparative attributive adjective	[ein] schnelleres [Auto]
ADJA3	superlative attributive adjective	[das] schnellste [Auto]
ADJD	(positive) predicative or adverbial adjective	[es ist] schnell, [es fährt] schnell
ADJD2	comparative predicative or adverbial adjective	[es ist] schneller, [es fährt] schneller
ADJD3	superlative predicative or adverbial adjective	[es ist am] schnellsten, [er meint daß er am] schnellsten [fährt].
ADV	non-adjectival adverb	oft, heute, bald, vielleicht
ART	article	der [Mann], eine [Frau]
CARD	cardinal	1, eins, 1/8, 205
CIRCP	circumposition, right part	[um der Ehre] willen

Tag	**Description**	**Example**
CM	comma	,
COADV	adverbial conjunction	aber, doch, denn
COALS	conjunction *als*	als
COINF	infinitival conjunction	ohne [zu fragen], anstatt [anzurufen]
COORD	coordinating conjunction	und, oder
COP1	coordination 1st part	entweder [... oder]
COP2	coordination 2nd part	[weder ...] noch
COSUB	subordinating conjunction	weil, daß, ob [ich mitgehe]
COWIE	conjunction *wie*	wie
DATE	date	27.12.2006
DEMADJ	demonstrative adjective	solche [Mühe]
DEMDET	demonstrative determiner	diese [Leute]
DEMINV	invariant demonstrative	solch [ein schönes Buch]
DEMPRO	demonstrative pronoun	jener [sagte]
FM	foreign word	article, communication
INDADJ	indefinite adjective	[die] meisten [Leute], viele [Leute], [die] meisten [sind da], viele [sind da]
INDDET	indefinite determiner	kein [Mensch]
INDINV	invariant indefinite	manch [einer]
INDPRO	indefinite pronoun	man [sagt]
ITJ	interjection	oh, ach, weh, hurra
NOUN	common noun, nominalized adjective, nominalized infinitive, or proper noun	Hut, Leute, [das] Gute, [das] Wollen, Peter, [die] Schweiz
ORD	ordinal	2., dritter
PERSPRO	personal pronoun	ich, du, ihm, mich, uns
POSDET	possessive determiner	mein [Haus]
POSPRO	possessive pronoun	[das ist] meins
POSTP	postposition	[des Geldes] wegen
PREP	preposition	in, auf, wegen, mit
PREPART	preposition article	im, ins, aufs
PTKANT	sentential particle	ja, nein, bitte, danke
PTKCOM	comparative particle	desto [schneller]
PTKINF	particle: infinitival *zu*	[er wagt] zu [sagen]
PTKNEG	particle: negation *nicht*	nicht
PTKPOS	positive modifier	zu [schnell], allzu [schnell]
PTKSUP	superlative modifier	am [schnellsten]
PUNCT	other punctuation, bracket	; : ( ) [ ] - "
REFLPRO	reflexive *sich*	sich

Tag	Description	Example
RELPRO	relative pronoun	[der Mann,] der [lacht]
REZPRO	reciprocal *einander*	einander
SENT	sentence final punctuation	. ? !
SYMB	symbols	@, %
TRUNC	truncated word, (first part of a compound or verb prefix)	Ein- [und Ausgang], Kinder- [und Jugendheim], be- [und entladen]
VAFIN	finite auxiliary	[er] ist, [sie] haben
VAINF	auxiliary infinitive	[er will groß] sein
VAPP	auxiliary past participle	[er ist groß] geworden
VMFIN	finite modal	[er] kann, [er] mochte
VMINF	modal infinitive	[er wird kommen] können
VPREF	separated verbal prefix	[er kauft] ein, [sie sieht] zu
VVFIN	finite verb form	[er] sagt
VVINF	infinitive	[er will] sagen, einkaufen
VVIZU	infinitive with incorporated *zu*	[um] einzukaufen
VVPP	past participle	[er hat] gesagt
WADV	interrogative adverb	wieso [kommt er?]
WDET	interrogative determiner	welche [Nummer?]
WINV	invariant interrogative	welch [ein ...]
WPRO	interrogative pronoun	wer [ist da?]

A.8. Greek POS Tags

Tag	Description	Example
ADJ	(basic) adjective	παιδικό
ADV	(basic) adverb	ευχαρόστως
ART	article	η, της
CARD	cardinal	χᾶια
CLIT	clitic (pronoun)	τον, τοό
CM	comma	,
COORD	coordinating conjunction	και
COSUBJ	conjunction with subjunctive	αντϕ[να]
CURR	currency	$
DIG	digits	123
FM	foreign word	article
FUT	future tense particle	θα
INTJ	interjection	χό
ITEM	item	1.2

Tag	Description	Example
NEG	negation particle	ή
NOUN	common noun	βιβλό
ORD	ordinal	τρόα
PERS	personal pronoun	εγό
POSS	possessive pronoun	άς, τους
PREPART	preposition with article	στο
PRON	pronoun	αυτοό
PRONREL	relative pronoun	οποές
PROP	proper noun	Μαρά
PTCL	particle	ας
PUNCT	punctuation (other than SENT and CM)	: -
QUOTE	quotation marks	"
SENT	sentence final punctuation	. ! ?
SUBJ	subjunctive particle	να
SUBORD	subordinating conjunction	πως
SYMB	special symbol	*, %
VIMP	verb (imperative)	γρψ́ε
VIND	verb (indicative)	γρφεις
VINF	verb (infinitive)	γρφει
VPP	past participle	δικασόνο

A.9. Hungarian POS Tags

Tag	Description	Example
ADJ	(invariant) adjective	kis
ADV	adverb	jól
ADV_PART	adverbial participle	állva
ART	article	az
AUX	auxiliary	szabad
CM	comma	,
CONJ	conjunction	és
DEICT_PRON_NOM	deictic pronoun: nominative	ez
DEICT_PRON_ACC	deictic pronoun: accusative	ezt
DEICT_PRON_CASE	deictic pronoun: other case	ebbe
FUT_PART_NOM	future participle: nominative	teendõ
FUT_PART_ACC	future participle: accusative	teendõt
FUT_PART_CASE	future participle: other case	teendõvel
GENE_PRON_NOM	general pronoun: nominative	minden

Tag	**Description**	**Example**
GENE_PRON_ACC	general pronoun: accusative	mindent
GENE_PRON_CASE	general pronoun: other case	mindenbe
INDEF_PRON_NOM	indefinite pronoun: nominative	más
INDEF_PRON_ACC	indefinite pronoun: accusative	mást
INDEF_PRON_CASE	indefinite pronoun: other case	mással
INF	infinitive (verb)	csinálni
INTERJ	interjection	jaj
LS	list item symbol	1)
MEA	measure, unit	km
NADJ_NOM	noun or adjective: nominative	ifjú
NADJ_ACC	noun or adjective: accusative	ifjút
NADJ_CASE	noun or adjective: other case	ifjúra
NOUN_NOM	noun: nominative	asztal
NOUN_ACC	noun: accusative	asztalt
NOUN_CASE	noun: other case	asztalra
NUM_NOM	numeral: nominative	három
NUM_ACC	numeral: accusative	hármat
NUM_CASE	numeral: other case	háromra
NUM_PRON_NOM	numeral pronoun: nominative	kevés
NUM_PRON_ACC	numeral pronoun: accusative	keveset
NUM_PRON_CASE	numeral pronoun: other case	kevéssel
NUMBER	numerals (digits)	1
ORD_NUMBER	ordinal	1.
PAST_PART_NOM	past participle: nominative	meghívott
PAST_PART_ACC	past participle: accusative	meghívottat
PAST_PART_CASE	past participle: other case	meghívottakkal
PERS_PRON	personal pronoun	én
POSTPOS	postposition	alatt
PREFIX	prefix	át
PRES_PART_NOM	present participle: nominative	csináló
PRES_PART_ACC	present participle: accusative	csinálót
PRES_PART_CASE	present participle: other case	csinálónak
PRON_NOM	pronoun: nominative	milyen
PRON_ACC	pronoun: accusative	milyet
PRON_CASE	pronoun: other case	milyenre
PROPN_NOM	proper noun: accusative	Budapestet
PROPN_ACC	proper noun: other case	Budapestre
PROPN_CASE	proper noun: nominative	Budapest

Tag	Description	Example
PUNCT	punctuation (other than SENT or CM)	( )
REFL_PRON_NOM	reflexive pronoun: accusative	magát
REFL_PRON_ACC	reflexive pronoun: other case	magadra
REFL_PRON_CASE	reflexive pronoun: nominative	magad
REL_PRON_NOM	relative pronoun: nominative	aki
REL_PRON_ACC	relative pronoun: accusative	akit
REL_PRON_CASE	relative pronoun: other case	akire
ROM_NUMBER	Roman numeral	IV
SENT	sentence final punctuation	., ;
SPEC	special string (URL, email)	www.xzymn.com
SUFF	suffix	-re
TRUNC	compound part	asztal-
VERB	verb	csinál

A.10. Italian POS Tags

Tag	Description	Example
ADJEX	proclitic noun modifier *ex*	ex
ADJPL	plural adjective	belle
ADJSG	singular adjective	buono, narcisistico
ADV	adverb	lentamente, già, poco
CLIT	clitic pronoun or adverb	vi, ne, mi, ci
CM	comma	,
CONJ	conjunction	e, ed, e/o
CONNADV	adverbial connector	quando, dove, come
CONNCHE	relative pronoun or conjunction	ch', che
CONNCHI	relative or interrogative pronoun *chi*	chi
DEMPL	plural demonstrative	quelli
DEMSG	singular demonstrative	ciò
DETPL	plural determiner	tali, quei, questi
DETSG	singular determiner	uno, questo, il
DIG	digits	+5, iv, 23.05, 3,45, 1997
INTERJ	interjection	uhi, perdiana, eh
ITEM	list item marker	A.
LET	single letter	[di tipo] C
NPL	plural noun	case
NSG	singular noun	casa, balsamo
ORDPL	plural ordinal	terzi

Tag	**Description**	**Example**
ORDSG	singular ordinal	secondo
POSSPL	plural possessive	mie, vostri, loro
POSSSG	singular possessive	nostro, sua
PRECLIT	pre-clitic	me [lo dai], te [la rubo]
PRECONJ	pre-conjunction	dato [che]
PREDET	pre-determiner	tutto [il giorno], tutti [i problemi]
PREP	preposition	tra, di, con, su di
PREPARTPL	preposition + plural article	sulle, sugl', pegli
PREPARTSG	preposition + singular article	sullo, nella
PREPREP	pre-preposition	prima [di], rispetto [a]
PRON	pronoun (3rd person singular/plural) *sé*	[disgusto di] sé
PRONINDPL	plural indefinite pronoun	entrambi, molte
PRONINDSG	singular indefinite pronoun	troppa
PRONINTPL	plural interrrogative pronoun	quali, quanti
PRONINTSG	singular interrogative pronoun	cos'
PRONPL	plural personal pronoun	noi, loro
PRONREL	invariant relative pronoun	cui
PRONRELPL	plural relative pronoun	quali, quanti
PRONRELSG	singular relative pronoun	quale
PRONSG	singular personal pronoun	esso, io, tu, lei, lui
PROP	proper noun	Bernardo, Monte Isola
PUNCT	other punctuation	- ;
QUANT	invariant quantifier	qualunque, qualsivoglia
QUANTPL	plural quantifier, numbers	molti, troppe, tre
QUANTSG	singular quantifier	niuna, nessun
SENT	sentence final punctuation	. ! ? :
VAUXF	finite auxiliary *essere* or *avere*	è, sarò, saranno, avrete
VAUXGER	gerund auxiliary *essere* or *avere*	essendo, avendo
VAUXGER_CLIT	gerund auxiliary + clitic	essendogli
VAUXIMP	imperative auxiliary	sii, abbi
VAUXIMP_CLIT	imperative auxiliary + clitic	siatene, abbiatemi
VAUXINF	infinitive auxiliary *essere*/*avere*	esser, essere, aver, avere
VAUXINF_CLIT	infinitive auxiliary *essere*/*avere* + clitic	esserle, averle
VAUXPPPL	plural past participle auxiliary	stati/e, avuti/e
VAUXPPPL_CLIT	plural past part. auxiliary + clitic	statine, avutiti
VAUXPPSG	singular past participle auxiliary	stato/a, avuto/a
VAUXPPSG_CLIT	singular past part. auxiliary + clitic	statone, avutavela
VAUXPRPL	plural present participle auxiliary	essenti, aventi

Tag	Description	Example
VAUXPRPL_CLIT	plural present participle auxiliary + clitic	aventile
VAUXPRSG	singular present participle auxiliary	essente, avente
VAUXPRSG_CLIT	singular present participle auxiliary + clitic	aventela
VF	finite verb form	blatereremo, mangio
VF_CLIT	finite verb + clitic	trattansi, leggevansi
VGER	gerund	adducendo, intervistando
VGER_CLIT	gerund + clitic	saziandole, appurandolo
VIMP	imperative	pareggiamo, formulate
VIMP_CLIT	imperative + clitic	impastategli, accoppiatevele
VINF	verb infinitive	sciupare, trascinar
VINF_CLIT	verb infinitive + clitic	spulciarsi, risucchiarsi
VPPPL	plural past participle	riposti, offuscati
VPPPL_CLIT	plural past participle + clitic	assestatici, ripostine
VPPSG	singular past participle	sbudellata, chiesto
VPPSG_CLIT	singular past participle + clitic	commossosi, ingranditomi
VPRPL	plural present participle	meditanti, destreggianti
VPRPL_CLIT	plural present participle + clitic	epurantile, andantivi
VPRSG	singular present participle	meditante, destreggiante
VPRSG_CLIT	singular present participle + clitic	epurantelo, andantevi

A.11. Japanese POS Tags

Tag	Description	Example
AA	adnominal adjective	その[人]，この[日]，同じ
AJ	normal adjective	美しい，早い，面白い
AN	adjectival noun	きれい[だ]，静か[だ]，正確[だ]
D	adverb	じっと，じろっと，ふと
EOS	sentence-final punctuation	。．
FP	non-derivational prefix	両[選手]，現[首相]
FS	non-derivational suffix	[綺麗]な，[派手]だ
HP	honorific prefix	お[風呂]，ご[不在]，ご[意見]
HS	honorific suffix	[小泉]氏，[恵美]ちゃん，[伊藤]さん
I	interjection	こんにちは，ほら，どっこいしょ
J	conjunction	すなわち，なぜなら，そして
NC	common noun	公園，電気，デジタルカメラ
NE	noun before numerals	文禄[三年]

Tag	Description	Example
NN	numeral	3, 2, 五, 二百
NP	proper noun	北海道, 斉藤
NR	pronoun	私, あなた, これ
NU	classifier	[100]メートル, [3]リットル
O	others	BASIS
PL	particle	[雨]が[降る], [そこ]に[座る], [私]は[一人]
PUNCT	punctuation other than end of sentence	,「」();
UNKNOWN	unknown	デパ[地下], ヴェロ
V	verb	書く, 食べます, 来た
V1	vowel-stem verb	食べ[る], 集め[る], 起き[る]
V5	consonant-stem verb	気負[う], 知り合[う], 行き交[う]
VN	verbal noun	議論[する], ドライブ[する], 旅行[する]
VS	suru-verb	馳せ参[じる], 相半ば[する]
VX	irregular verb	移り行[く], トラブ[る]
WP	derivational prefix	チョー[綺麗], バカ[正直]
WS	derivational suffix	[東京]都, [大阪]府, [白]ずくめ

A.12. Korean POS Tags

Tag	Description	Example
B	adverb	혹은
D	determiner	이
EOS	sentence-final punctuation	.
FW	foreign word	TV, alphabet
j	josa (post-positional particle)	으로
N	noun	책거리
NN	number	6.25
PUNCT	other punctuation	,)
UNKNOWN	unknown part of speech	
V	verb/adjective	참고하시기, 바랍니다
L	exclamation	와
Q	symbol	・(HANGUL LETTER ARAEA)
e	Eomi (verb ending)	'새 국민의 정부'라는

A.13. Polish POS Tags

Tag	**Description**	**Example**
ADV	adverb: adjectival	szybko
	adverb: comparative adjectival	szybciej
	adverb: superlative adjectival	najszybciej
	adverb: non-adjectival	trochę, wczoraj
ADJ	adjective: attributive (postnominal)	[stopy] procentowe
	adjective: attributive (prenominal)	szybki [samochód]
	adjective: predicative	[on jest] ogromny
	adjective: comparative attributive	szybszy [samochód]
	adjective: comparative predicative	[on jest] szybszy
	adjective: superlative attributive	najszybszy [samochód]
	adjective: superlative predicative	[on jest] najszybszy
CJ/AUX	conjunction with auxiliary *być*	[robi wszystko,] żebyśmy [przyszli]
CM	comma	,
CMPND	compound part	[ośrodek] naukowo-[badawczy]
CONJ	conjunction	a, ale, gdy, i, lub
DATE	date expression	31.12.99
EXCL	interjection	aha, brawo, hej
FRGN	foreign material	cogito, numerus
NOUN	noun: common	reakcja, weksel
	noun: proper	Krzysztof, Francja
	noun: nominalized adjective	chory, [pośpieszny z] Krakowa
NUM	numeral (cardinal)	22; 10,25; 5-7; trzy
ORD	numeral (ordinal)	12. [maja], 2., 12go, 13go, 28go
PHRAS	phraseology	[po] polsku, [na] bosaka, [w] mig, fiku-miku
PPERS	personal pronoun	ja, ty, on, ona, my, wy, mnie, tobie, jej, jemu, nam, mi, ci, jego, go, nas, was
PR/AUX	pronoun with auxiliary *być*	[co] wyście [zrobili]
PREFL	reflexive pronoun	[nie może] sobie [przypomnieć], [zabierz to ze] sobą, [warto] sobie [zadać pytanie]
PREL	relative pronoun	który [problem], jaki [problem], co, który [on widzi], jakie [mamuzeum]
PREP	preposition	od [dzisiaj], na [rynku walutowym]

Tag	Description	Example
PRON	pronoun: demonstrative	[w] tym [czasie]
	pronoun: indefinite	wszystkie [stopy procentowe], jakieś [nienaturalne rozmiary]
	pronoun: "expletive"	to [(jest) ostateczna decyzja]
	pronoun: possessive	nasi [dwaj bracia]
	pronoun: interrogative	Jaki [masz samochód?]
PRTCL	particle	także, nie, tylko, już
PT/AUX	particle with auxiliary *być*	gdzieście [byli]
PUNCT	punctuation (other than CM or SENT)	: ( ) [ ] " " - "
QVRB	quasi-verb	brak, szkoda
SENT	sentence final punctuation	. ! ? ;
SYMB	symbol	@ §
TIME	time expression	11:00
VAUX	auxiliary	być, zostać
VFIN	finite verb form: present	[Agata] maluje [obraz]
	finite verb form: future	[Piotr będzie] malował [obraz]
VGER	gerund	[demonstrują] domagając [się zmian]
VINF	infinitive	odrzucić, stawić [się]
VMOD	modal	[wojna] może [trwać nawet rok]
VPRT	verb participle: predicative	[wynik jest] przesądzony
	verb participle: passive	[postępowanie zostanie] zawieszone
	verb participle: attributive	[zmiany] będące [wynikiem...]

A.14. Portuguese POS Tags

Tag	Description	Example
ADJ	invariant adjective	[duas saias] cor-de-rosa
ADJPL	plural adjective	[cidadãos] portugueses
ADJSG	singular adjective	[continente] europeu
ADV	adverb	directamente
ADVCOMP	comparison adverb *mais* and *menos*	[um país] mais [livre]
AUXBE	finite "be" (*ser* or *estar*)	é, são, estão
AUXBEINF	infinitive "be"	ser, estar
AUXBEINFPRON	infinitive "be" with clitic	sê-lo
AUXBEPRON	finite "be" with clitic	é-lhe
AUXHAV	finite "have"	tem, haverá
AUXHAVINF	infinitive "have" (*ter*, *haver*)	ter, haver
AUXHAVINFPRON	infinitive "have" with clitic	ter-se
AUXHAVPRON	finite "have" with clitic	tinham-se

Tag	**Description**	**Example**
CM	comma	,
CONJ	(coordinating) conjunction	[por fax] ou [correio]
CONJCOMP	comparison conjunction *do que*	[mais] do que [uma vez]
CONJSUB	subordination conjunction	para que, se, que
DEMPL	plural demonstrative	estas
DEMSG	singular demonstrative	aquele
DETINT	interrogative or exclamative *que*	[demostra a] que [ponto]
DETINTPL	plural interrogative determiner	quantas [vezes]
DETINTSG	singular interrogative determiner	qual [reação]
DETPL	plural definite article	os [maiores aplausos]
DETRELPL	plural relative determiner	..., cujas [presações]
DETRELSG	singular relative determiner	..., cuja [veia poética]
DETSG	singular definite article	o [service]
DIG	digit	123
GER	gerundive	examinando
GERPRON	gerundive with clitic	deixando-a
INF	verb infinitive	reunir, conservar
INFPRON	infinitive with clitic	datar-se
INTERJ	interjection	oh, aí, claro
ITEM	list item marker	A. [Introdução]
LETTER	isolated character	[da seleção] A
NEG	negation	não, nunca
NOUN	invariant common noun	caos
NPL	plural common noun	serviços
NPROP	proper noun	PS, Lisboa
NSG	singular common noun	[esta] rede
POSSPL	plural possessive	seus [investigadores]
POSSSG	singular possessive	sua [sobrinha]
PREP	preposition	para, de, com
PREPADV	preposition + adverb	[venho] daqui
PREPDEMPL	preposition + plural demonstrative	desses [recursos]
PREPDEMSG	preposition + singular demonstrative	nesta [placa]
PREPDETPL	preposition + plural determiner	dos [Grandes Bancos]
PREPDETSG	preposition + singular determiner	na [construção]
PREPPRON	preposition + pronoun	[atrás] dela
PREPQUANTPL	preposition + plural quantifier	nuns [terrenos]
PREPQUANTSG	preposition + singular quantifier	numa [nuvem]
PREPREL	preposition + invariant relative pronoun	[nesta praia] aonde

Tag	Description	Example
PREPRELPL	preposition + plural relative pronoun	[alunos] aos quais
PREPRELSG	preposition + singular relative pronoun	[área] através do qual
PRON	invariant pronoun	se, si
PRONPL	plural pronoun	as, eles, os
PRONSG	singular pronoun	a, ele, ninguém
PRONREL	invariant relative pronoun	[um ortopedista] que
PRONRELPL	plural relative pronoun	[as instalações] as quais
PRONRELSG	singular relative pronoun	[o ensaio] o qual
PUNCT	other punctuation	: ( ) ;
QUANTPL	plural quantifier	quinze, alguns, tantos
QUANTSG	singular quantifier	um, algum, qualquer
SENT	sentence final punctuation	. ! ?
SYM	symbols	@ %
VERBF	finite verb form	corresponde
VERBFPRON	finite verb form with clitic	deu-lhe
VPP	past participle (also adjectival use)	penetrado, referida

A.15. Russian POS Tags

Tag	Description	Example
ADJ	adjective	красивая, зеленый, удобный, темный
ADJ_CMP	adjective: comparative	красивее, зеленее, удобнее, темнее
ADV	adverb	быстро, просто, легко, правильно
ADV_CMP	adverb: comparative	быстрее, проще, легче, правильнее
AMOUNT	currency + cardinal, percentages	$20.000, 10%
CM	comma	,
CONJ	conjunction	что,или и, а
DET	determiner	какой, некоторым [из вас], который[час]
DIG	numerals (digits)	1, 2000, 346
FRGN	foreign word	бутерброд, армия, сопрано
IREL	relative/interrogative pronoun	кто [сделает это?] каков [результат?], сколько [стоит?], чей
ITJ	interjection	увы, ура
MISC	(miscellaneous)	АЛ345, чат, N8

Tag	Description	Example
NOUN	common noun: nominative case	страна
	common noun: accusative case	[любить] страну
	common noun: dative case	[посвятить] стране
	common noun: genitive case	[история] страны
	common noun: instrumental case	[гордиться] страной
	common noun: prepositional case	гороворить о стране
NUM	numerals (spelled out)	шестьсот, десять, два
ORD	ordinal	12., 1.2.1., IX.
PERS	personal pronoun	я, ты, они, мы
PREP	preposition	в, на, из-под [земли], с [горы]
PRONADV	pronominal adverb	как, там, зачем, никогда, когда-нибудь
PRON	pronoun	все, тем, этим, себя
PROP	proper noun	Россия, Арктика, Ивановых, Александра
PTCL	particle	[но все] же,[постой]-ка [ну]-ка,
PTCL_INT	introduction particle	вот [она], вон [там], пускай, неужели, ну
PTCL_MOOD	mood marker	[если] бы, [что] ли,[так] бы [и сделали]
PTCL_SENT	stand-alone particle	впрочем, однако
PUNCT	punctuation (other than CM or SENT)	: ; " " ( )
SENT	sentence final punctuation	. ? !
SYMB	symbol	*, ~
VAUX	auxiliary verb	быть,[у меня] есть
VFIN	finite verb	ходили, любила, сидит,
VGER	verb gerund	бывая, думая, засыпая
VINF	verb infinitive	ходить, любить, сидеть,
VPRT	verp participle	зависящий [от родителей], сидящего [на стуле]

A.16. Spanish POS Tags

Tag	Description	Example
ADJ	invariant adjective	beige, mini
ADJPL	plural adjective	bonitos, nacionales
ADJSG	singular adjective	bonito, nacional
ADV	adverb	siempre, directamente
ADVADJ	adverb, modifying an adjective	muy [importante]
ADVINT	interrogative adverb	adónde, cómo, cuándo

Tag	**Description**	**Example**
ADVNEG	negation *no*	no
ADVREL	relative adverb	cuanta, cuantos
AUX	finite auxiliary *ser* or *estar*	es, fui, estaba
AUXINF	infinitive *ser*, *estar*	estar, ser
AUXINFCL	infinitive *ser*, *estar* with clitic	serme, estarlo
CM	comma	,
COMO	reserved for word *como*	como
CONADV	adverbial conjunction	adonde, cuando
CONJ	conjunction	y, o, si, porque,sin que
DETPL	plural determiner	los, las, estas, tus
DETQUANT	invariant quantifier	demás, más, menos
DETQUANTPL	plural quantifier	unas, ambos, muchas
DETQUANTSG	singular quantifier	un, una, ningún, poca
DETSG	singular determiner	el, la, este, mi
DIG	numerals (digits)	123, XX
HAB	finite auxiliary *haber*	han, hubo, hay
HABINF	infinitive *haber*	haber
HABINFCL	infinitive *haber* with clitic	haberle, habérseme
INTERJ	interjection	ah, bravo, olé
ITEM	list item marker	a)
NOUN	invariant noun	bragazas, fénix
NOUNPL	plural noun	aguas, vestidos
NOUNSG	singular noun	agua, vestido
NUM	numerals (spelled out)	once, tres, cuatrocientos
PAPPL	past participle, plural	contenidos, hechas
PAPSG	past participle, singular	privado, fundada
PREDETPL	plural pre-determiner	todas [las], todos [los]
PREDETSG	singular pre-determiner	toda [la], todo [el]
PREP	preposition	en, de, con, para, dentro de
PREPDET	preposition + determiner	al, del, dentro del
PRON	pronoun	ellos, todos, nadie, yo
PRONCLIT	clitic pronoun	le, la, te, me, os, nos
PRONDEM	demonstrative pronoun	eso, esto, aquello
PRONINT	interrogative pronoun	qué, quién, cuánto
PRONPOS	possessive pronoun	(el) mío, (las) vuestras
PRONREL	relative pronoun	(lo) cual, quien, cuyo
PROP	proper noun	Pablo, Beralfier
PUNCT	punctuation (other than CM or SENT)	' ¡ ¿ : {

Tag	**Description**	**Example**
QUE	reserved for word *que*	que
SE	reserved for word *se*	se
SENT	sentence final punctuation	. ? ; !
VERBFIN	finite verb form	tiene, pueda, dicte
VERBIMP	verb imperative	dejad, oye
VERBIMPCL	imperative with clitic	déjame, sígueme
VERBINF	verb infinitive	evitar, tener, conducir
VERBINFCL	infinitive with clitic	hacerse, suprimirlas
VERBPRP	present participle	siendo, tocando
VERBPRPCL	present participle with clitic	haciéndoles, tomándolas

Appendix B. Morphological and Special Tags

When creating a user dictionary for For German, Dutch, Hungarian, English, French, Italian, Portuguese, and Spanish, you can include morphological tags and/or special tags in individual dictionary entries.

Morphological Tags. The Named Entity Extractor [156] uses the Morphological tags listed in the following sections to help it identify named entities.

Special Tags. For German, Dutch, and Hungarian, the Base Linguistics Language Analyzer (BL1) uses special tags to divide compounds into their components.

For German, Dutch, Hungarian, English, French, Italian, and Portuguese, BL1 uses special tags to indicate the boundaries in multi-word baseforms, contractions and elisions, and words with clitics. BL1 displays these boundaries as spaces in the STEM [87] results.

B.1. German Morphological Tags

Tag	**Description**	**Examples**
+Bus	business name	Basis Technology, Daimler
+Continent	continent	Europa
+Country	country, nation	Deutschland
+First	first name	Hans, Maria
+Lake	lake	Bodensee
+Last	family name	Schiller, Maier
+Org	organization	Bundestag, UNO
+Prop	proper noun	Hans, Maier, Basis Technology
+Region	region, mountain,...	Zugspitze, Sahara
+River	river	Rhein, Donau
+Sea	sea, ocean	Nordsee, Mittelmeer
+State	State, Bundesland, Kanton, ...	Hessen, Uri, Arizona
+Title	title (mainly for abbr.)	Herr, Dr. [Maier]
+Town	town, city	Stuttgart
+TownCountry	town and/or country	Monaco
+TownState	town and/or state	Bremen

B.2. German Special Tags

BL1 uses the composition boundary and compound linking tags to handle compounds. It replaces ^& and ^_ with a space when it returns a STEM.

Tag	**Description**	**Example: Token (STEM)**
^#	composition boundary	Kindergarten (Kind^#Garten)
^-	composition boundary (hyphen)	US-weit (US^-weit)
^/	compound linking element	Arbeitszeit (Arbeit^/s^#Zeit)

Tag	Description	Example: Token (STEM)
^&	token boundary for contracted word form	gib's (geben^&es)
^_	space in multi-word token	ein wenig (ein^_wenig)

B.3. English Morphological Tags

These tags apply to English (en) and upper-case English (en_uc).

Tag	Description	Example
+Bus	business name	Basis Technology
+City	city name	London
+Continent	continent name	Europe
+Country	country name	Scotland
+Fam	family name	Clinton
+Fem	proper noun feminine gender	Mary
+Masc	proper noun masculine gender	Peter
+Misc	miscellaneous place name	Thames
+Place	place name	Everest
+Prop	poper name	Peter
+Title	title	Mister
+Usastate	a state in the USA	California

B.4. English Special Tags

These tags apply to English (en) and upper-case English (en_uc). BL1 replaces each of these tags with a space when it returns a STEM.

Tag	Description	Example: Token (STEM)
^=	separator for contracted forms	don't (do^=not)
^_	space in multi-word baseforms	New York (New^_York)

B.5. Spanish Morphological Tags

Tag	Description	Example
+Continent	continent name	Europa
+Lugar	place	Madrid
+Pay	country	España
+Soc	company name	ABC
+Titl	title	Señora
+Usastate	a state in USA	Texas

B.6. Spanish Special Tags

Tag	Description	Example: Token (STEM)
^=	separator for contracted forms	al (a^=el)
^_	space in multi-word baseforms	al uso (al^_uso)
^|	derivation boundary	duramente (duro^|)

B.7. French Morphological Tags

Tag	Description	Examples
+Location	noun indicating location; may behave as an adverb	rue, impasse, avenue
+PreN	first name	Nicolas, Marie
+Proper	proper noun	Paris, Jean
+Tit	title	Mme, Prof.

B.8. French Special Tag

BL1 replaces this tag with a space when it returns a STEM.

Tag	Description	Example: token (STEM)
^=	separator for contracted forms	aux (à^=le)

B.9. Hungarian Special Tags

BL1 uses the compound entry tag to handle compounds. It replaces ^| with a space when it returns a STEM.

Tag	Description	Example: Token (STEM)
^CB+	compound boundary	hírnévre (hír^CB+név)
^|	segment boundary (lexicalized)	nemcsak (nem^|csak)

B.10. Italian Morphological Tags

Tag	Description	Example
+Cit	name of a city	Roma
+Continent	continent (proper noun)	Europa
+Fam	family name	Romano
+Giv	given name	Anna
+Lake	name of lake	Maggiore
+Mount	name of mountain	Vesuvio
+Org	name of organization	ONU
+Pay	name of a state or country	Italia
+Place	place name	Romagna

Tag	Description	Example
+Prop	proper noun	Paolo
+River	name of river	Po
+Sea	name of sea or ocean	Adriatico
+Title	title	signora
+Usastate	a state in the USA	Texas

B.11. Italian Special Tags

BL1 replaces each of these tags with a space when it returns a STEM.

Tag	Description	Example: Token (STEM)
^=	separator for contractions and elisions	allo (a^=lo)
^_	space in multi-word baseforms	a meno de (a^_meno^_di)
^|	derivation boundary or separator for clitics	farti :: fare^|tu

B.12. Dutch Morphological Tags

Tag	Description	Example
+Abbr	abbreviation	km
+City	city, town name	Amsterdam
+Continent	name of continent	Europa
+Country	name of country	Holland
+Fam	family name	Bakker
+Giv	given name	Anna
+Org	organization	Basis Technology
+Place	place	Emmastraat
+Prop	proper noun	Anna
+Region	region	Brabant
+Title	title	prof.
+Usastate	a state in USA	Utah

B.13. Dutch Special Tags

BL1 uses the compound tags to handle compounds. It replaces ^_ with a space when it returns a STEM.

Tag	Description	Example: Token (STEM)
^#	compound boundary	prijscompensatie (prijs^#compensatie)
^\ \	boundary for compound linking element	herhalingsprik (herhaling^\\s^#prik)
^_	space in multi-word token	nu en dan (nu^_en^_dan)

B.14. Portuguese Special Tags

BL1 replaces each of these tags with a space when it returns a STEM

Tag	Description	Example: Token (STEM)
^=	separator for contractions and elisions	lhas (ela^=elas)
^_	space in multi-word baseforms	por causa de (por^_causa^_de)
^|	derivation boundary or separator for clitics	amá-lo (amar^|ele)

Appendix C. Tcl Regular Expression Syntax

The Regular Expression [163] processor uses the *Tcl* regular expression engine to identify named entities in input text. To see the named entity types that the Regular Expression processor with the shipped regular expressions file returns, see Named Entities [45] . For background information about adding your own regular expressions, see Creating Regular Expressions [56] .

This appendix contains information extracted from the Tcl re_syntax Manual Page [http://www.tcl.tk/man/tcl/TclCmd/re_syntax.htm]. It also contains the Tcl Software License [248] .

C.1. Name

re_syntax - Syntax of Tcl regular expressions.

C.2. Description

A *regular expression* describes strings of characters. It's a pattern that matches certain strings and doesn't match others.

C.3. Different Flavors of REs

Regular expressions ("RE"s), as defined by POSIX, come in two flavors: *extended* REs ("EREs") and *basic* REs ("BREs"). EREs are roughly those of the traditional *egrep*, while BREs are roughly those of the traditional *ed*. This implementation adds a third flavor, *advanced* REs ("AREs"), basically EREs with some significant extensions.

This manual page primarily describes AREs. BREs mostly exist for backward compatibility in some old programs; they will be discussed at the end. POSIX EREs are almost an exact subset of AREs. Features of AREs that are not present in EREs will be indicated.

C.4. Regular Expression Syntax

Tcl regular expressions are implemented using the package written by Henry Spencer, based on the 1003.2 spec and some (not quite all) of the Perl5 extensions (thanks, Henry!). Much of the description of regular expressions below is copied verbatim from his manual entry.

An ARE is one or more *branches*, separated by '|', matching anything that matches any of the branches.

A branch is zero or more *constraints* or *quantified atoms*, concatenated. It matches a match for the first, followed by a match for the second, etc; an empty branch matches the empty string.

A quantified atom is an *atom* possibly followed by a single *quantifier*. Without a quantifier, it matches a match for the atom. The quantifiers, and what a so-quantified atom matches, are:

*****
: a sequence of 0 or more matches of the atom

+
: a sequence of 1 or more matches of the atom

?
: a sequence of 0 or 1 matches of the atom

{ *m* }
: a sequence of exactly *m* matches of the atom

{ *m* ,}
: a sequence of *m* or more matches of the atom

{ *m* , *n* }
: a sequence of *m* through *n* (inclusive) matches of the atom; *m* may not exceed *n*

***? +? ?? { *m* }? { *m* ,}? { *m* , *n* }?**
: *non-greedy* quantifiers, which match the same possibilities, but prefer the smallest number rather than the largest number of matches (see Matching)

The forms using **{** and **}** are known as *bound*s. The numbers *m* and *n* are unsigned decimal integers with permissible values from 0 to 255 inclusive.

An atom is one of:

(*re*)
: *re* is any regular expression) matches a match for *re*, with the match noted for possible reporting

(?: *re*)
: as previous, but does no reporting (a "non-capturing" set of parentheses)

()
: matches an empty string, noted for possible reporting

(?:)
: matches an empty string, without reporting

[*chars*]
: a *bracket expression*, matching any one of the *chars* (see BRACKET EXPRESSIONS for more detail)

.
: matches any single character

\ *k*
: (where *k* is a non-alphanumeric character) matches that character taken as an ordinary character, e.g. \\ matches a backslash character

\ *c*
: where *c* is alphanumeric (possibly followed by other characters), an *escape* (AREs only), see Escapes below

{
: when followed by a character other than a digit, matches the left-brace character '**{**'; when followed by a digit, it is the beginning of a *bound* (see above)

x
: where *x* is a single character with no other significance, matches that character.

A *constraint* matches an empty string when specific conditions are met. A constraint may not be followed by a quantifier. The simple constraints are as follows; some more constraints are described later, under Escapes.

^
: matches at the beginning of a line

$
: matches at the end of a line

(?= *re*)
: *positive lookahead* (AREs only), matches at any point where a substring matching *re* begins

(?! *re*)
: *negative lookahead* (AREs only), matches at any point where no substring matching *re* begins

The lookahead constraints may not contain back references (see later), and all parentheses within them are considered non-capturing.

An RE may not end with '\'.

C.5. Bracket Expressions

A *bracket expression* is a list of characters enclosed in '**[ ]**'. It normally matches any single character from the list (but see below). If the list begins with '^', it matches any single character (but see below) *not* from the rest of the list.

If two characters in the list are separated by '**-**', this is shorthand for the full *range* of characters between those two (inclusive) in the collating sequence, e.g. **[0-9]** in ASCII matches any decimal digit. Two ranges may not share an endpoint, so e.g. **a-c-e** is illegal. Ranges are very collating-sequence-dependent, and portable programs should avoid relying on them.

To include a literal**]** or **-** in the list, the simplest method is to enclose it in **[.** and **.]** to make it a collating element (see below). Alternatively, make it the first character (following a possible '^'), or (AREs only) precede it with '\'. Alternatively, for '**-**', make it the last character, or the second endpoint of a range. To use a literal **-** is the first endpoint of a range, make it a collating element or (AREs only) precede it with '\'. With the exception of these, some combinations using **[** and escapes, all other special characters lose their special significance within a bracket expression.

Within a bracket expression, a collating element (a character, a multi-character sequence that collates as if it were a single character, or a collating-sequence name for either) enclosed in **[.** and **.]** stands for the sequence of characters of that collating element. The sequence is a single element of the bracket expression's list. A bracket expression in a locale that has multi-character collating elements can thus match more than one character. So (insidiously), a bracket expression that starts with ^ can match multi-character collating elements even if none of them appear in the bracket expression! (*Note:* Tcl currently has no multi-character collating elements. This information is only for illustration.)

For example, assume the collating sequence includes a **ch** multi-character collating element. Then the RE **[[.ch.]]*c** (zero or more **ch**'s followed by **c**) matches the first five characters of '**chchcc**'. Also, the RE **[^c]b** matches all of '**chb**' (because **[^c]** matches the multi-character **ch**).

Within a bracket expression, a collating element enclosed in **[=** and **=]** is an equivalence class, standing for the sequences of characters of all collating elements equivalent to that one, including itself. (If there are no other equivalent collating elements, the treatment is as if the enclosing delimiters were '**[.**'and '**.]**'.) For example, if **o** and **ô** are the members of an equivalence class, then '**[[=o=]]**', '**[[=ô=]]**', and '**[oô]**' are all synonymous. An equivalence class may not be an endpoint of a range. (*Note:* Tcl currently implements only the Unicode locale. It doesn't define any equivalence classes. The examples above are just illustrations.)

Within a bracket expression, the name of a *character class* enclosed in **[:** and **:]** stands for the list of all characters (not all collating elements!) belonging to that class. Standard character classes are:

alpha A letter.
upper An upper-case letter.
lower A lower-case letter.
digit A decimal digit.
xdigit A hexadecimal digit.
alnum An alphanumeric (letter or digit).
print An alphanumeric (same as alnum).
blank A space or tab character.
space A character producing white space in displayed text.
punct A punctuation character.
graph A character with a visible representation.
cntrl A control character.

A locale may provide others. See Character Classes [57] (Note that the current Tcl implementation has only one locale: the Unicode locale.) A character class may not be used as an endpoint of a range.

There are two special cases of bracket expressions: the bracket expressions **[[:<:]]** and **[[:>:]]** are constraints, matching empty strings at the beginning and end of a word respectively. A word is defined as a sequence of word characters that is neither preceded nor followed by word characters. A word character is an *alnum* character or an underscore (_). These special bracket expressions are deprecated; users of AREs should use constraint escapes instead (see below).

C.6. Escapes

Escapes (AREs only), which begin with a \ followed by an alphanumeric character, come in several varieties: character entry, class shorthands, constraint escapes, and back references. A \ followed by an alphanumeric character but not constituting a valid escape is illegal in AREs. In EREs, there are no escapes: outside a bracket expression, a \ followed by an alphanumeric character merely stands for that character as an ordinary character, and inside a bracket expression, \ is an ordinary character. The latter is the one actual incompatibility between EREs and AREs.

Character-entry escapes (AREs only) exist to make it easier to specify non-printing and otherwise inconvenient characters in REs:

\a
alert (bell) character, as in C

\b
backspace, as in C

\B
synonym for \ to help reduce backslash doubling in some applications where there are multiple levels of backslash processing

\c *X*
: (where X is any character) the character whose low-order 5 bits are the same as those of *X*, and whose other bits are all zero

\e
: the character whose collating-sequence name is '**ESC**', or failing that, the character with octal value 033

\f
: formfeed, as in C

\n
: newline, as in C

\r
: carriage return, as in C

\t
: horizontal tab, as in C

\u *wxyz*
: (where *wxyz* is exactly four hexadecimal digits) the Unicode character **U+** *wxyz* in the local byte ordering

\U *stuvwxyz*
: (where *stuvwxyz* is exactly eight hexadecimal digits) reserved for a somewhat-hypothetical Unicode extension to 32 bits

\v
: vertical tab, as in C are all available.

\x *hhh*
: (where *hhh* is any sequence of hexadecimal digits) the character whose hexadecimal value is **0x** *hhh* (a single character no matter how many hexadecimal digits are used).

\0
: the character whose value is **0**

\ *xy*
: (where *xy* is exactly two octal digits, and is not a *back reference* (see below)) the character whose octal value is **0** *xy*

\ *xyz*
: (where *xyz* is exactly three octal digits, and is not a back reference (see below)) the character whose octal value is **0** *xyz*

Hexadecimal digits are '**0**'-'**9**', '**a**'-'**f**', and '**A**'-'**F**'. Octal digits are '**0**'-'**7**'.

The character-entry escapes are always taken as ordinary characters. For example, **\135** is **]** in ASCII, but **\135** does not terminate a bracket expression. Beware, however, that some applications (e.g., C compilers) interpret such sequences themselves before the regular-expression package gets to see them, which may require doubling (quadrupling, etc.) the '\'.

Class-shorthand escapes (AREs only) provide shorthands for certain commonly-used character classes:

\d
: **[[:digit:]]**

\s
: **[[:space:]]**

\w
: **[[:alnum:]_]** (note underscore)

\D
: **[^[:digit:]]**

\S
: **[^[:space:]]**

\W
: **[^[:alnum:]_]** (note underscore)

Within bracket expressions, '**\d**', '**\s**', and '**\w**' lose their outer brackets, and '**\D**', '**\S**', and '**\W**' are illegal. (So, for example, **[a-c\d]** is equivalent to **[a-c[:digit:]]**. Also, **[a-c\D]**, which is equivalent to **[a-c^[:digit:]]**, is illegal.)

A constraint escape (AREs only) is a constraint, matching the empty string if specific conditions are met, written as an escape:

\A
: matches only at the beginning of the string (see Matching, below, for how this differs from '^')

\m
: matches only at the beginning of a word

\M
: matches only at the end of a word

\y
: matches only at the beginning or end of a word

\Y
: matches only at a point that is not the beginning or end of a word

\Z
: matches only at the end of the string (see Matching, below, for how this differs from '**$**')

\ *m*
: (where *m* is a nonzero digit) a *back reference*, see below

\ *mnn*
: (where *m* is a nonzero digit, and *nn* is some more digits, and the decimal value *mnn* is not greater than the number of closing capturing parentheses seen so far) a *back reference*, see below

A word is defined as in the specification of **[[:<:]]** and **[[:>:]]** above. Constraint escapes are illegal within bracket expressions.

A back reference (AREs only) matches the same string matched by the parenthesized subexpression specified by the number, so that (e.g.) **([bc])\1** matches **bb** or **cc** but not '**bc**'. The subexpression must entirely precede the back reference in the RE. Subexpressions are numbered in the order of their leading parentheses. Non-capturing parentheses do not define subexpressions.

There is an inherent historical ambiguity between octal character-entry escapes and back references, which is resolved by heuristics, as hinted at above. A leading zero always indicates an octal escape. A single nonzero digit, not followed by another digit, is always taken as a back reference. A multi-digit sequence not starting with a zero is taken as a back reference if it comes after a suitable subexpression (i.e. the number is in the legal range for a back reference), and otherwise is taken as octal.

C.7. Metasyntax

In addition to the main syntax described above, there are some special forms and miscellaneous syntactic facilities available.

Normally the flavor of RE being used is specified by application-dependent means. However, this can be overridden by a *director*. If an RE of any flavor begins with '*****:**', the rest of the RE is an ARE. If an RE of any flavor begins with '*****=**', the rest of the RE is taken to be a literal string, with all characters considered ordinary characters.

An ARE may begin with *embedded options*: a sequence **(?** *xyz* **)** (where *xyz* is one or more alphabetic characters) specifies options affecting the rest of the RE. These supplement, and can override, any options specified by the application. The available option letters are:

b	rest of RE is a BRE
c	case-sensitive matching (usual default)
e	rest of RE is an ERE
i	case-insensitive matching (see Matching, below)
m	historical synonym for **n**
n	newline-sensitive matching (see Matching, below)
p	partial newline-sensitive matching (see Matching, below)
q	rest of RE is a literal ("quoted") string, all ordinary characters
s	non-newline-sensitive matching (usual default)
t	tight syntax (usual default; see below)
w	inverse partial newline-sensitive ("weird") matching (see Matching, below)
x	expanded syntax (see below)

Embedded options take effect at the **)** terminating the sequence. They are available only at the start of an ARE, and may not be used later within it.

In addition to the usual (*tight*) RE syntax, in which all characters are significant, there is an *expanded* syntax, available in all flavors of RE with the **-expanded** switch, or in AREs with the embedded x option. In the expanded syntax, white-space characters are ignored and all characters between a **#** and the following newline (or the end of the RE) are ignored, permitting paragraphing and commenting a complex RE. There are three exceptions to that basic rule:

a white-space character or '**#**' preceded by '\' is retained
white space or '**#**' within a bracket expression is retained
white space and comments are illegal within multi-character symbols like the ARE '**(?:**' or the BRE '**\(**'

Expanded-syntax white-space characters are blank, tab, newline, and any character that belongs to the *space* character class.

Finally, in an ARE, outside bracket expressions, the sequence '**(?#** *ttt* **)**' (where *ttt* is any text not containing a '**)**') is a comment, completely ignored. Again, this is not allowed between the characters of multi-character symbols like '**(?:**'. Such comments are more a historical artifact than a useful facility, and their use is deprecated; use the expanded syntax instead.

None of these metasyntax extensions is available if the application (or an initial *****=** director) has specified that the user's input be treated as a literal string rather than as an RE.

C.8. Matching

In the event that an RE could match more than one substring of a given string, the RE matches the one starting earliest in the string. If the RE could match more than one substring starting at that point, its choice is determined by its *preference*: either the longest substring, or the shortest.

Most atoms, and all constraints, have no preference. A parenthesized RE has the same preference (possibly none) as the RE. A quantified atom with quantifier **{** *m* **}** or **{** *m* **}?** has the same preference (possibly none) as the atom itself. A quantified atom with other normal quantifiers (including **{** *m* **,** *n* **}** with *m* equal to *n*) prefers longest match. A quantified atom with other non-greedy quantifiers (including **{** *m* **,** *n* **}?** with *m* equal to *n*) prefers shortest match. A branch has the same preference as the first quantified atom in it which has a preference. An RE consisting of two or more branches connected by the **|** operator prefers longest match.

Subject to the constraints imposed by the rules for matching the whole RE, subexpressions also match the longest or shortest possible substrings, based on their preferences, with subexpressions starting earlier in the RE taking priority over ones starting later. Note that outer subexpressions thus take priority over their component subexpressions.

Note that the quantifiers **{1,1}** and **{1,1}?** can be used to force longest and shortest preference, respectively, on a subexpression or a whole RE.

Match lengths are measured in characters, not collating elements. An empty string is considered longer than no match at all. For example, **bb*** matches the three middle characters of '**abbbc**', **(week|wee)(night|knights)** matches all ten characters of '**weeknights**', when **(.*).*** is matched against **abc** the parenthesized subexpression matches all three characters, and when **(a*)*** is matched against **bc** both the whole RE and the parenthesized subexpression match an empty string.

If case-independent matching is specified, the effect is much as if all case distinctions had vanished from the alphabet. When an alphabetic that exists in multiple cases appears as an ordinary character outside a bracket expression, it is effectively transformed into a bracket expression containing both cases, so that **x** becomes '**[xX]**'. When it appears inside a bracket expression, all case counterparts of it are added to the bracket expression, so that **[x]** becomes **[xX]** and **[^x]** becomes '**[^xX]**'.

If newline-sensitive matching is specified, **.** and bracket expressions using ^ will never match the newline character (so that matches will never cross newlines unless the RE explicitly arranges it) and ^ and **$** will match the empty string after and before a newline respectively, in addition to matching at beginning and end of string respectively. ARE **\A** and **\Z** continue to match beginning or end of string *only*.

If partial newline-sensitive matching is specified, this affects **.** and bracket expressions as with newline-sensitive matching, but not ^ and '**$**'.

If inverse partial newline-sensitive matching is specified, this affects ^ and **$** as with newline-sensitive matching, but not **.** and bracket expressions. This isn't very useful but is provided for symmetry.

C.9. Limits and Compatibility

No particular limit is imposed on the length of REs. Programs intended to be highly portable should not employ REs longer than 256 bytes, as a POSIX-compliant implementation can refuse to accept such REs.

The only feature of AREs that is actually incompatible with POSIX EREs is that \ does not lose its special significance inside bracket expressions. All other ARE features use syntax which is illegal or has undefined or unspecified effects in POSIX EREs; the ******* syntax of directors likewise is outside the POSIX syntax for both BREs and EREs.

Many of the ARE extensions are borrowed from Perl, but some have been changed to clean them up, and a few Perl extensions are not present. Incompatibilities of note include '**\b**', '**\B**', the lack of special treatment for a trailing newline, the addition of complemented bracket expressions to the things affected by newline-sensitive matching, the restrictions on parentheses and back references in lookahead constraints, and the longest/shortest-match (rather than first-match) matching semantics.

Henry Spencer's original 1986 *regexp* package, still in widespread use (e.g., in pre-8.1 releases of Tcl), implemented an early version of today's EREs. There are four incompatibilities between *regexp*'s near-EREs ('RREs' for short) and AREs. In roughly increasing order of significance:

In AREs, \ followed by an alphanumeric character is either an escape or an error, while in RREs, it was just another way of writing the alphanumeric. This should not be a problem because there was no reason to write such a sequence in RREs.
{ followed by a digit in an ARE is the beginning of a bound, while in RREs, **{** was always an ordinary character. Such sequences should be rare, and will often result in an error because following characters will not look like a valid bound.
In AREs, \ remains a special character within '**[ ]**', so a literal \ within **[ ]** must be written '\\'. \\ also gives a literal \ within **[ ]** in RREs, but only truly paranoid programmers routinely doubled the backslash.
AREs report the longest/shortest match for the RE, rather than the first found in a specified search order. This may affect some RREs which were written in the expectation that the first match would be reported. The careful crafting of RREs to optimize the search order for fast matching is obsolete (AREs examine all possible matches in parallel, and their performance is largely insensitive to their complexity) but cases where the search order was exploited to deliberately find a match which was *not* the longest/shortest will need rewriting.

C.10. Basic Regular Expressions

BREs differ from EREs in several respects. '|', '+', and **?** are ordinary characters and there is no equivalent for their functionality. The delimiters for bounds are **{** and '**}**', with **{** and **}** by themselves ordinary characters. The parentheses for nested subexpressions are **(** and '**)**', with **(** and **)** by themselves ordinary characters. ^ is an ordinary character except at the beginning of the RE or the beginning of a parenthesized subexpression, **$** is an ordinary character except at the end of the RE or the end of a parenthesized subexpression, and ***** is an ordinary character if it appears at the beginning of the RE or the beginning of a parenthesized subexpression (after a possible leading '^'). Finally, single-digit back references are available, and \< and \> are synonyms for **[[:<:]]** and **[[:>:]]** respectively; no other escapes are available.

C.11. Tcl License

Appendix D. Error Codes

RLP APIs and logs may return the error codes described below. Positive error codes indicate success `BT_OK` or a non-error condition, either `BT_NO_MORE_DATA` or `BT_WANT_MORE_DATA`. Negative error codes fall into the following categories:

Error Code Ranges

-1 to -49	Internal errors.
-50 to -99	Argument, class, or state validity errors.
-100 to -149	System (memory, file, etc.) errors.
-150 to -174	License Key library (`btkey`)
-10000 or higher	Codes specific to RLP

Table D.1. Error Codes

Error #	Hex #	Error Name	This error code is returned when...
1	1	`BT_OK`	The function completed successfully.
2	2	`BT_NO_MORE_DATA`	There is no more data. This is an informational message and does not necessarily mean that an error occurred during processing.
3	3	`BT_WANT_MORE_DATA`	The function cannot continue until the caller passes the object more data. This is an informational message and does not necessarily mean that an error occurred during processing.
-1	-1	`BT_ERR_UNSPECIFIED`	The function encounters an unspecified error.
-2	-2	`BT_ERR_BUFFER_TRUNCATED`	The data written into a `BT_Char16Buf` was truncated. This is an informational message and does not necessarily mean that an error occurred during processing.
-3	-3	`BT_ERR_INTERNAL`	The function encounters an internal error. Contact Basis Technology through the appropriate email address for the product you are using. In your email, carefully describe the scenario in which the error occurred.
-4	-4	`BT_ERR_UNIMPLEMENTED`	The function does not implement the functionality requested. Typically, this happens when a platform-specific feature cannot be used because it doesn't exist on the present platform.
-50	-32	`BT_ERR_INVALID_INSTANCE`	A handle passed to the function (generally a C function) is invalid.

Error #	Hex #	Error Name	This error code is returned when...
-51	-33	BT_ERR_INVALID_ARGUMENT	An argument passed to the function is invalid.
-52	-34	BT_ERR_INVALID_FILE_FORMAT	A file whose name is passed to the function has an invalid format.
-100	-64	BT_ERR_SYSTEM_ERROR	The function encounters a system-level error. The caller can check the value of `errno` to determine the exact error.
-101	-65	BT_ERR_OUT_OF_MEMORY	The function could not allocate new memory, or if there is insufficient memory for a required operation.
-102	-66	BT_ERR_FILE_NOT_FOUND	A file whose name is passed to the function could not be found.
-103	-67	BT_ERR_FILE_PERMISSION_DENIED	A file whose name is passed to the function could not be accessed because of a permissions restriction.
-150	-96	BT_ERR_LICENSE_INVALID	At initialization, a license file is present but the required key is not valid. The key may be missing a field, or a field value may not be valid.
-151	-97	BT_ERR_LICENSE_EXPIRED	At initialization, a license file is present but has passed its expiration date.
-152	-98	BT_ERR_LICENSE_WRONG_PLATFORM	At initialization, a license file is present but the platform in the license key is not valid.
-153	-99	BT_ERR_LICENSE_NOT_AVAILABLE	A license file is not available. This happens when attempting to run a language processor for which there is no license.
-154	-9A	BT_ERR_LICENSE_FILE_NOT_FOUND	At initialization, a license file could not be found or could not be processed.
-155	-9B	BT_ERR_DATA_NOT_LICENCED	An attempt is made to create a dictionary from a data file that is not licensed.
-156	-9C	BT_ERR_DATUM_NOT_FOUND	A datum, such as a record or table entry, which has been requested, cannot be found in the corresponding resource.
-157	-9D	BT_ERR_DATA_GENERATION_INCOMPATIBLE	An attempt is made to initiate use of a database, table, dictionary, or other data collection which is not of a compatible generation, i.e., was not built with, intended to be used with, or operable with, the versions of other comparable collections already in use by the application or subsystem.
-10000	-2710	BT_RLP_ERR_NO_LANGUAGE_PROCESSORS	No language processors have been defined (or could be loaded) within a context. A context must have at least one language processor to be valid.

Error #	Hex #	Error Name	This error code is returned when...
-10002	-2712	`BT_RLP_ERR_INVALID_PROCESSOR_VERSION`	A processor's internal API version does not match the version required by the core RLP library. This can occur if the processor is newer than the version of RLP being used.
-10003	-2713	`BT_RLP_ERR_NO_LICENSES_AVAILABLE`	There are no license keys defined. This can occur when the named license file exists but does not contain valid key values.
-10004	-2714	`BT_RLP_ERR_LANGUAGE_NOT_SUPPORTED`	A processor does not support the language requested. This is not necessarily an error; a language processor can be called for a language it doesn't support, in which case it will do nothing, returning this value.
-10005	-2715	`BT_RLP_ERR_REQUIRED_DATA_MISSING`	A language processor required data that is not available. For example, if named entity extraction requires that the input be POS-tagged and it is not, the language processor may return this value.
-10006	-2716	`BT_RLP_ERR_CHARSET_NOT_SUPPORTED`	The application passes a mime charset to \texttt{ProcessBuffer} or \texttt{ProcessFile} and either the charset is not acceptable to the processor or is invalid or undefined. Note that some processors ignore the charset or treat it as a hint, not a firm declaration.
-10007	-2717	`BT_RLP_ERR_INVALID_INPUT_DATA`	A processor detects data that it cannot process. For example, a processor for a specific file format returns this error when the data is not in the specified file format.
-10008	-2718	`BT_RLP_ERR_NO_ROOT_DIRECTORY`	The application has not established a Basis root directory.
-10010	-271A	`BT_RLP_ERR_UNINITIALIZED_ENVIRONMENT`	The application attempts to create a context before initializing an environment.

Appendix E. Guidelines for Reporting Bugs

When you encounter a bug in our software, we need to reproduce the bug before we can take steps to fix it. This document provides guidelines that we ask you to follow so we can address the bug as quickly as possible.

E.1. Background Information

Before reporting a bug to productsupport@basistech.com, please check that you are providing the following information:

- product name
- version
- platform

We do keep records of versions shipped to customers. However, since many customers use multiple versions and platforms, we ask that you tell us specifically which version and platform you are actually using.

The easiest way to provide this information is to tell us the name of the SDK package you are using, such as **rlp-6.5.2-sdk-amd64-glibc23-gcc34.tar.gz** or **rlp-6.5.2-sdk-ia32-w32-msvc80.zip**. The package filename identifies product, version, and platform.

E.1.1. Platform

If you do not have the original package, the name of the subdirectory under **rlp/bin** is the platform name. For example: **amd64-glibc23-gcc34** or **ia32-w32-msvc80**.

E.1.2. Version

You can find the version by running the **rlp** command-line utility with the `-v` flag.

For information on running **rlp**, see Using the Command-line Utility [10] .

APIs for obtaining version information also exist:

C++

```
BT_RLP_Library::VersionNumber()
BT_RLP_Library::VersionString()
```

C

```
BT_RLP_CLibrary_VersionNumber()
BT_RLP_CLibrary_VersionString()
```

Java

```
RLPEnvironment::versionString()
```

E.2. Reproducing the Bug with the rlp Command-line Utility

You may have found the bug in your own application. In many cases, it is not feasible for Basis to run or debug your application, so we ask that you try to reproduce the bug with the **rlp** command-line tool. With some effort, most bugs can be reproduced in this manner. The **rlp[.exe]** command-line utility is often sufficient to reproduce bugs that you encounter using the C++, C, or Java APIs.

For more information, see Using the Command-line Utility to Process Your Own Text [10] .

A small percentage of bugs may be hard to reproduce with the **rlp** command-line utility. For example, the bug may depend on multiple threads or a specific order of document processing. But please make every effort to isolate a simple command-line test case before contacting Basis Technology.

If you can reproduce the bug with the command-line utility, please provide us the following:

- The command line you ran
- The RLP Environment configuration (usually **rlp/etc/rlp-global.xml**)
- The RLP Context configuration (such as **rlp/samples/etc/rlp-context.xml**)
- The input file you used
- The license you used (usually **rlp/rlp/licenses/rlp-license.xml**)

Example on Unix. Assume you have installed the RLP SDK to **~/btroot** and that your platform is `amd64-glibc23-gcc34`. You are using the default environment and context files. The input file is **~/btroot/in.txt**. For maximum output to the console, set the BT_RLP_LOG_LEVEL environment variable to "all" before running the command.

```
export BT_RLP_LOG_LEVEL=all
export LD_LIBRARY_PATH= ~/btroot/rlp/lib/amd64-glibc23-gcc34
cd ~/btroot
rlp/bin/amd64-glibc23-gcc34/rlp -root .  -lang en rlp/etc/rlp-global.xml rlp/samples/etc/rlp-context.xml \
in.txt
```

Example on Windows. Assume you have installed the RLP SDK to **\btroot** and that your platform is `ia32-w32-msvc80`. You are using the default environment and context files. The input file is **\btroot\in.txt**. For maximum output to the console, set the BT_RLP_LOG_LEVEL environment variable to "all" before running the command.

```
cd \btroot
set BT_RLP_LOG_LEVEL=all
rlp\bin\ia32-w32-msvc80\rlp.exe -root . -lang en rlp\etc\rlp-global.xml rlp\samples\etc\rlp-context.xml
in.txt
```

If you use the default **rlp-context.xml** file, REXML [165] output is written to **rlp-output.xml** in the current directory.

E.3. Reproducing in a Sample Application

If the **rlp** command-line tool is not sufficient to reproduce the bug (for example, you are using an RLP API not called by the tool), you may write a small sample application tailored to provoke the bug. As a starting point, you can use the core sample applications shipped with RLP:

C++
: **rlp/samples/cplusplus/rlp_sample.cpp** (see also Sample C++ Application [28])

C
: **rlp/samples/capi/rlp_sample_c.c** (see also Sample C Application [34])

Java
: **rlp/samples/java/RLPSample.java** (see also Java Sample Application [59])

Having created a small application that reproduces the bug, you can provide us with the application so that we can also reproduce the bug, determine the cause, and work on a fix.

Appendix F. The Rosette Demo

The Rosette Demo Package

The Rosette Demo is available as a separate package. It is no longer included in the RLP SDK package for Windows platforms. For information about obtaining the Rosette Demo Package, please contact `ProductSupport@basistech.com` .

F.1. Launching the GUI Demo

Be Sure the License is In Place. To use RLP functionality, you must copy the RLP license you obtain from Basis Technology (**rlp-license.xml**) to **RLP Demo\rlp\rlp\licenses**, where **RLP Demo** is the directory where you install the Demo.

Select **All Programs** → **Basis Technology** → **Rosette Demo** from the Windows Desktop **start** menu to launch the demo.

The following text is also found in the demo's online help.

F.2. What the Demo Does

The **Rosette Demo** performs the following operations on input text in many languages. (Some operations may be unavailable if they are not authorized by your RLP license .)

- For multilingual text, identify individual language regions so the operations described below can be applied as appropriate to each language region.
- Identify the language, encoding, and MIME type.
- Tokenize the text, tag parts of speech, derive stems, analyze compounds, and provide phonetic transcriptions.
- Extract base noun phrases.
- Extract named entities. You can also create and edit named entity definitions [263] .
- Convert text in Simplified Chinese script to Traditional Chinese script and vice versa.

To see the list of languages these operations support, see RLP Key Features [1] .

F.3. How to Run the RLP Demo

F.3.1. Input the text to be analyzed

If the text includes more than one language, select the **Demo** → **Enable Multilingual Document Analysis** option. When this option is selected, **RLP Demo** applies each operation to the individual language regions as appropriate for each language rather than homogeneously to the entire body of the text.

- Load a text file by selecting **File** → **Open** (**Ctrl-O**) or 📂 to open a text file in any language and encoding. For information about the file, select **File** → **Properties**. [1]
- Paste in text by selecting **Edit** → **Paste** (**Ctrl-V**) to paste copied text from an application into the Text Window [261] .
- Type in your own text by selecting **File** → **New** (**Ctrl-N**) or 🗋.

[1]RLP accepts file formats including plain text, HTML, XML, PDF, Word, Excel, PowerPoint, and Access.

F.3.2. Edit or load the text

To edit text already in the **Text Window**, select **Edit** → **Edit Text** (**Alt-W**) or .

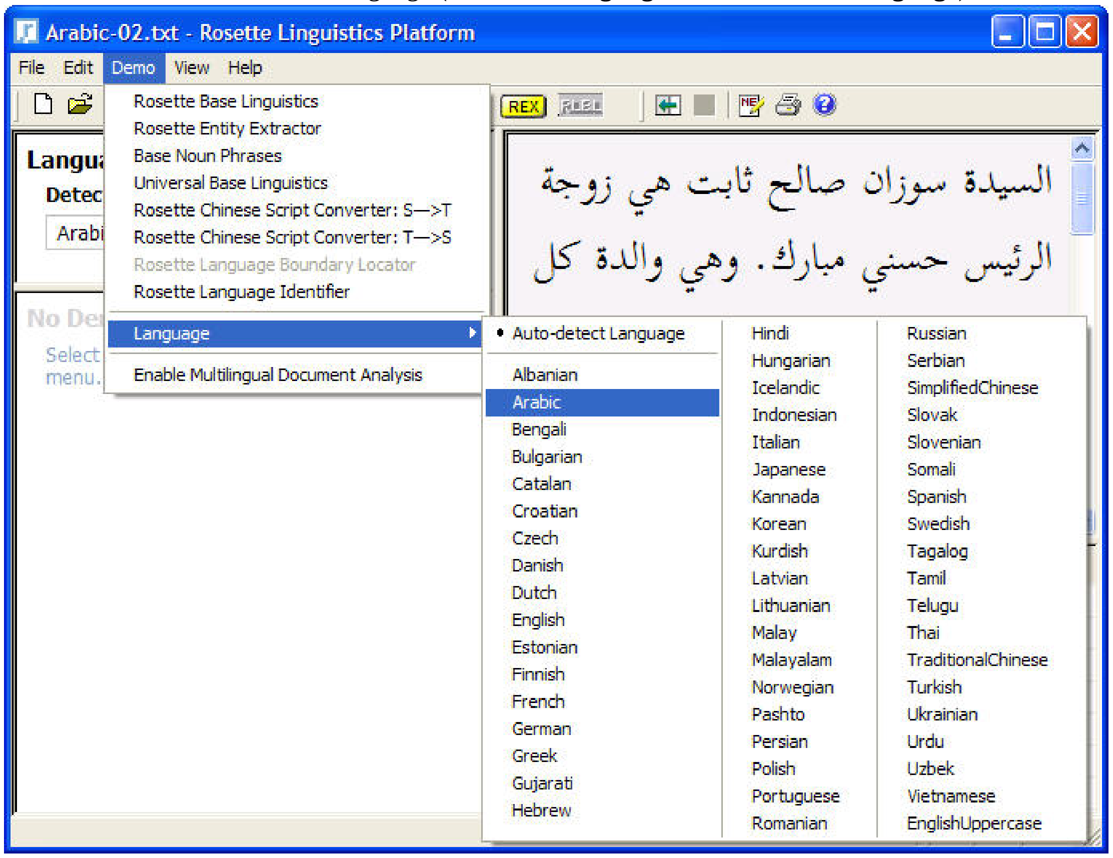
If you manually entered text into the input field, select to load the text; **RLP Demo** will display the text's language.

F.3.3. Specify the language (optional)

By default, RLP uses the Rosette Language Identifier (RLI) to auto-detect the language. If the text is multilingual and you have enabled multilingual processing on the **Demo** menu, RLP uses the Rosette Language Boundary Locator (RLBL) to detect language regions.

You can manually select a language from the **Demo** → **Language** submenu. You must do this if your RLP license does not contain RLI. [2] If you select a new language, any text in the **Text Window** is reprocessed immediately.

The manually selected language setting remains in place until you choose a different language, or switch the demo back to Auto-detect Language (**Demo** → **Language** → **Auto-detect Language**).



السيدة سوزان صالح ثابت هي زوجة
الرئيس حسني مبارك. وهي والدة كل

F.3.4. Apply a Demo Process

Select a process from the **Demo** menu, or click one of the shortcut buttons on the toolbar, and **RLP Demo** color codes the processed results to show linguistic features of the input text.

[2]If your license does not contain RLI, any plain text files you open for processing must be in in a Unicode encoding (UTF-8, UTF-16, or UTF-32).

Not all functions may be applicable or available for all languages. See the full list of functions available for each language [1] .

Rosette Language Identifier RLI**.** Uses RLI to identify the language, script, MIME type, encoding, and length.

Rosette Base Linguistics RBL**.** Tokenizes the text and tags parts of speech. For applicable languages [1] , it may also derive stems, analyze compounds, and generate phonetic transcriptions.

Base Noun Phrases. Identifies base noun phrases in the input text.

Rosette Entity Extractor REX**.** Uses statistical analysis, gazetteers, and regular expressions to identify named entities

- **Built-in entity types:** locations, organizations, persons, geo-political entities, facilities, religions, nationalities, titles, dates, and identifiers (credit card numbers, email addresses, latitudes and longitudes, money, personal ID numbers, phone numbers, distances, URLs, and Univeral Transverse Mercator coordinates).
- **Adding entities:** Once you have created a gazetteer [264] for a specific entity type, you can select words in the Text Window and right click to add them to that named entity type. In general, use the Named Entities Editor [263] to modify named entity definitions, create new named entity types, and extend the set of languages that are analyzed for named entities.
- **RLP Demo** automatically reruns REX after you make an addition while using the REX.

Rosette Language Boundary Locator RLBL**.** Uses the Rosette Language Boundary Locator (RLBL) to identify language regions and their size. This option is available if you have selected **Enable MultiLingual Document Analysis** from the **Demo** menu.

Universal Base Linguistics. Use this process to tokenize text in any language.

Rosette Chinese Script Converter. Converts text in Simplified Chinese script to Traditional Chinese script and vice versa.

F.3.5. View the analysis results

See Layout of the Demo Display [259] for details.

F.3.6. Save the analysis results (optional)

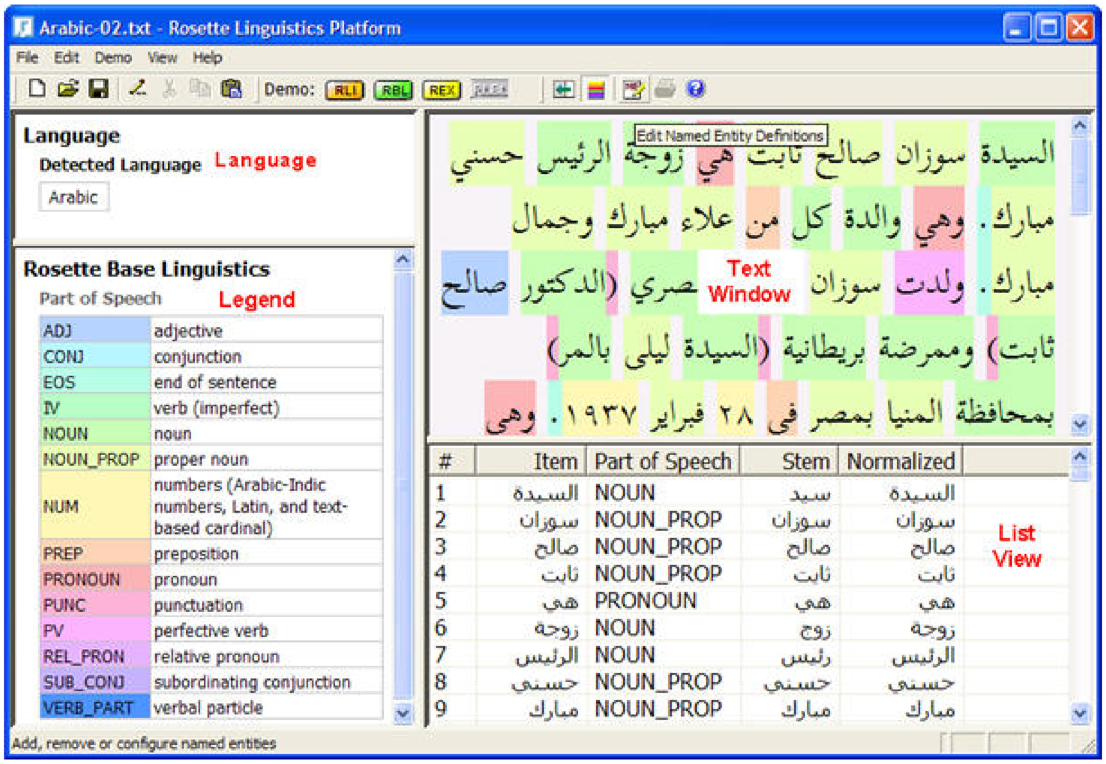
Save the list of tokens and their analysis as an XML file or a Comma-Separated Values (CSV) file with **File** → **Save Item List As**. For more information, see Saving Analysis Results [261] .

Send the list of tokens and their analysis to an **Excel** worksheet with **File** → **Send Item List to Excel**. For more information, see Sending Analysis Results to Excel [261] .

You can also save the input text [262] to a file to process again later.

F.4. Layout of the Demo Display

Here is an image of the display after Base Linguistics has been applied to some Arabic text.



Language

Detected Language

Arabic

Rosette Base Linguistics

Part of Speech

ADJ	adjective
CONJ	conjunction
EOS	end of sentence
IV	verb (imperfect)
NOUN	noun
NOUN_PROP	proper noun
NUM	numbers (Arabic-Indic numbers, Latin, and text-based cardinal)
PREP	preposition
PRONOUN	pronoun
PUNC	punctuation
PV	perfective verb
REL_PRON	relative pronoun
SUB_CONJ	subordinating conjunction
VERB_PART	verbal particle

السيدة سوزان صالح ثابت هي زوجة الرئيس حسني
مبارك. وهي والدة كل من علاء مبارك وجمال
مبارك. ولدت سوزان صري (الدكتور صالح
ثابت) وممرضة بريطانية (السيدة ليلى بالمر)
بمحافظة المنيا بمصر فى ٢٨ فبراير ١٩٣٧. وهى

#	Item	Part of Speech	Stem	Normalized
1	السيدة	NOUN	سيد	السيدة
2	سوزان	NOUN_PROP	سوزان	سوزان
3	صالح	NOUN_PROP	صالح	صالح
4	ثابت	NOUN_PROP	ثابت	ثابت
5	هي	PRONOUN	هي	هي
6	زوجة	NOUN	زوج	زوجة
7	الرئيس	NOUN	رئيس	الرئيس
8	حسني	NOUN_PROP	حسني	حسني
9	مبارك	NOUN_PROP	مبارك	مبارك

Add, remove or configure named entities

The display is divided into four panes. You can use the mouse to drag the horizontal and vertical divider bars between the panes. See Customizing the Display [262] .

Text Window Pane. Displays the input text. Select ✎ to switch between Edit mode and Read-Only mode. In Edit mode (the background is white), you can edit the text. In Read-Only mode (the background is grey), RLP has loaded the text, converting it to Unicode UTF-16. After you select a Demo process (from the **Demo** menu), words or phrases may be highlighted in color. See Using the Text Window [261]

Language Pane. Reports the language or languages of the input text. The **Language Pane** should display this information before you select a process to run.

When you are viewing multilingual text, click one of the language names to display the annotations for the text in that language.

Legend Pane. Shows the Demo process applied and, for some processes, displays the key to the color-coded parts of speech or named entity types in the input text. Click a part-of-speech (POS) tag, a named entity type, or a language region in the **Legend**, to highlight all corresponding elements in the **Text Window**.

When viewing multilingual text, click a language region to specify the text for which you want to see the annotations.

List View Pane. Displays the processed results. The first column numbers each token. The other columns shown depend on the process you have applied and the input language. See Using the List View [260] for details.

F.5. Using the List View

The **List View** displays the processed results in a table.

The first column numbers each token. The other columns vary depending upon the process applied and the input language.

Navigation and Selection. Click on the **List View** and you can use the arrow keys to navigate up and down between rows. When you select a row, the **Text Window** highlights the corresponding tokens in a contrasting color.

Copying a Token. To copy the token associated with the selected row to the clipboard, select **Edit** → **Copy** (**Ctrl-C**).

Saving Analysis Results. To save the entire contents of the **List View** as a UTF-8 encoded **XML** or **CSV** (comma-separated values) file, select **File** → **Save Item List As**. You are prompted to name the file and select a file format from a dropdown menu.

In the XML file, [3] field names are the **List View** column headers in lower case with spaces removed. In the CSV file the column names are upper cased. The # column is named `<index>` or `INDEX`.

Sending Analysis Results to Excel. To send the entire contents of the **List View** to an **Excel** worksheet, select **File** → **Send Item List to Excel**.

The item list appears in a new **Excel** worksheet. AutoFilter is turned on. You can use the Autofilter arrows that appear to the right of each column label to sort the display and to filter the display by selection. To turn AutoFilter off (or back on), select **Data** → **Filter** → **AutoFilter**.

F.5.1. Column order, size, sorting

To resize the columns, use the mouse to drag the dividing lines. Default column widths are restored each time you launch a process.

To reorder columns, drag and drop column headers, or bring up the **Select Columns** dialog box by right clicking any column header or using **View** → **Select Columns**. The dialog also lets you hide any column except the # column.

To sort rows by a column's values, click the column header. Click again to reverse the sort order.

The column settings are specific to the current combination of input language and Demo process. See Customizing and Saving Display Settings [262] to save these settings.

F.5.2. Font Selection

To change the List View font, select **View** → **Set View Font** → **List View**. This font change applies only to the current session and only to the specific language of the current input text. See Customizing and Saving Display Settings [262] to save these settings.

F.6. Using the Text Window

Navigation. Click on the **Text Window** and you can **Tab** through token by token (or go backwards using **Shift-Tab**).

Token highlighting. Navigating or clicking a highlighted token colors the corresponding item in the List View [260] . Toggle the highlighting on/off by selecting **View** → **Mark Entities** or ▤.

[3]Note that the XML format is *not* REXML, RLP's native XML format. To generate REXML, you must include the REXML processor in the context configuration file [267] .

Font. To change the **Text Window** font, select **View** → **Set View Font** → **Full Text View**. This font change only applies to text in the language of the current document (shown in **Text Profile**) for the current session. See Customizing and Saving Display Settings [262] to save the font selected.

Copying Input Text. Highlight the text to copy using the mouse or **Shift** and the arrow keys. Then select **Edit** → **Copy** (**Ctrl-C**) and paste the text into other applications. You cannot drag and drop text from the **Text Window**.

Saving Input Text. Save the contents of the **Text Window** to a file with **File** → **Save Text As** (**Ctrl-S**) or 🖫. You can save the text in any encoding supported by the Rosette Core Library for Unicode (RCLU).

F.7. Customizing and Saving Display Settings

You can save the following visual settings to an XML configuration file:

- Size and position of the overall Demo window
- Position of horizontal and vertical divider bars between the four panes [259]
- Layout of the **List View** columns [261] for each combination of Demo process and input language.
- List View font [261] and Text Window font [262] for each input language.

You can also edit **.htm** files that control some Legend display features [262] .

F.7.1. Saving and Loading Settings

Setting changes you have made are *not* automatically saved when the application is closed.

To save your settings, select **File** → **Save Configuration**.

To make these settings the default, save the file as **RLPDemoAppConfig.xml**, the default filename, in the **Basis Technology** subfolder of your **Local Settings/Application Data** profile. Otherwise, save the file to a different name and manually choose the file when you launch the Demo by selecting **File** → **Load Configuration**.

Alternatively, load the settings when you launch the demo with the `/config` switch from the command line.

```
btrlpdemo.exe /config my-customizations.xml
```

loads the previously saved customization file named **my-customization.xml** in the **Basis Technology** subfolder of your **Local Settings/Application Data** profile.

Use the `/config` switch with no argument to load the default configuration file, **RLPDemoAppConfig.xml**.

```
btrlpdemo.exe /config
```

F.7.2. Customizing the Legend

You can alter a limited subset of **Legend** display features by editing the **demo-xxx-legend.htm** files in **rlp\samples\w32demo**. See each file's comments for details.

For the Named Entities legend, text and coloring is automatic.

For the Part of Speech (POS) legends, the text is supplied in **demo-pos-legend.xml** and the coloring is automatic.

F.8. The Named Entities Editor

RLP uses statistical analysis (the Named Entity Extractor), regular expressions, and gazetteers to identify named entities. A number of entity types are pre-defined builtin types. You may add your own named entities as well as entity types and subtypes.

For a more detailed explanation about customizing named entities in RLP, consult Customizing Named Entities [52] .

The Named Entities Editor lets you:

- Create user-defined entity types and subtypes.
- Specify weights the Named Entity Redactor uses to determine which source to favor (gazetteers, regular expressions, or statistical analysis) when more than one source finds a named entity in the same or overlapping text.
- Edit and add gazetteers.
- Edit and add regular expressions.
- Set the normalization options used with gazetteers.

If you are viewing the results of the **Named Entities Extraction** process, **RLP Demo** automatically reruns the process after you make edits and close the editor.

F.8.1. Files the Named Entities Editor Modifies

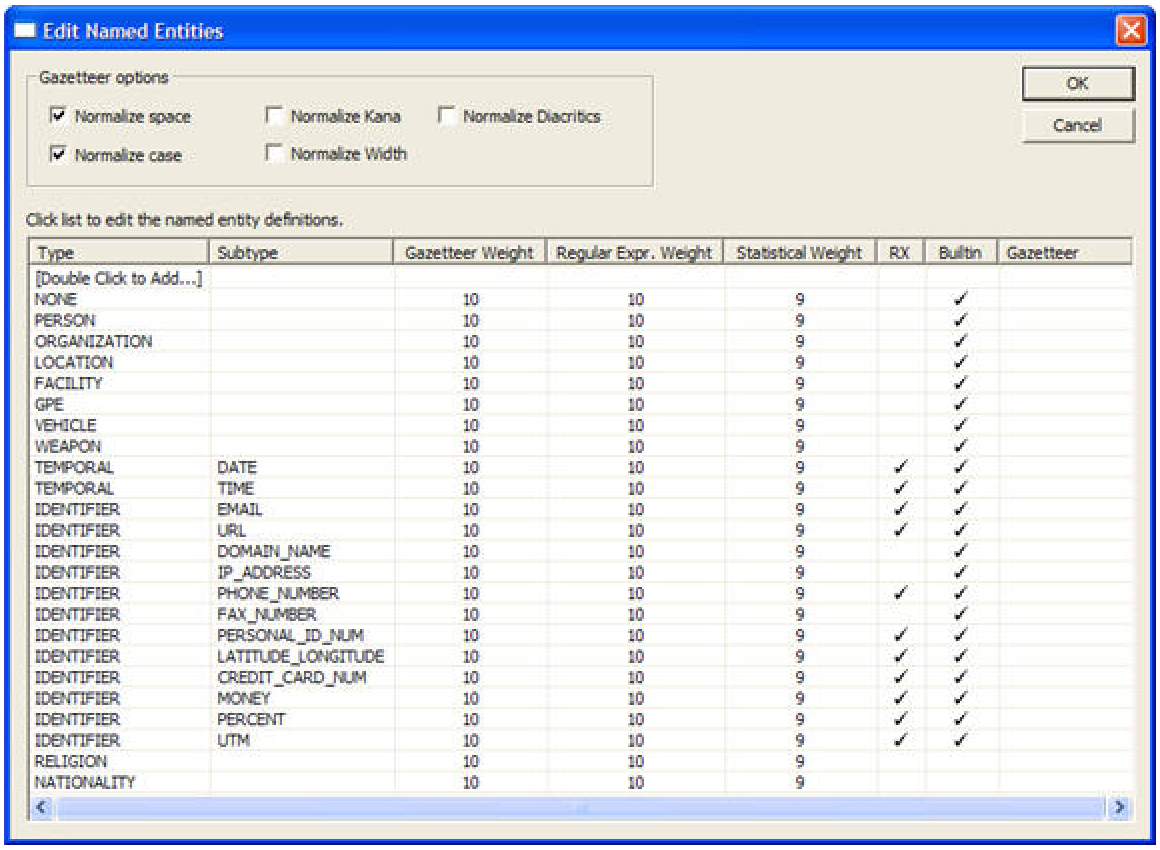
The Named Entities Editor edits the following files which are also used by both the demo and the RLP SDK.

- **rlp\etc\ne-types.xml** Specifies types, subtypes, and weights.
- **rlp\etc\regex-config.xml** Defines the regular expressions. Each regular expression is associated with a type and, optionally, with a subtype and a language.
- **rlp\etc\gazetteer-options.xml** Specifies the gazetteers to use and normalization settings.
- **Gazetteers.** Each gazetteer is associated with a type and, optionally with a subtype and a language.

Note: To view or edit any XML file with the Named Entities Editor, the corresponding DTD file must be in the folder with the XML file. For known file locations, **RLP Demo** copies the required DTD file from the **rlp/config/DTDs**. Make sure that the XML files and the folders containing the XML files you want to edit are not write protected.

F.8.2. Opening the Named Entitities Editor

Select **Edit** → **Named Entities** or to open the editor. The **Edit Named Entities** dialog displays Gazetteer Options and a list of entity types.



Gazetteer options

[✓] Normalize space
[✓] Normalize case
[ ] Normalize Kana
[ ] Normalize Width
[ ] Normalize Diacritics

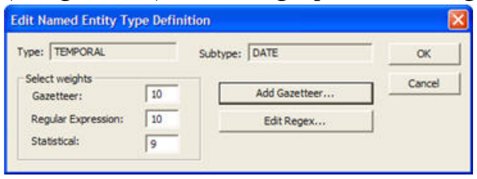
Click list to edit the named entity definitions.

Type	Subtype	Gazetteer Weight	Regular Expr. Weight	Statistical Weight	RX	Builtin	Gazetteer
[Double Click to Add...]							
NONE		10	10	9		✓	
PERSON		10	10	9		✓	
ORGANIZATION		10	10	9		✓	
LOCATION		10	10	9		✓	
FACILITY		10	10	9		✓	
GPE		10	10	9		✓	
VEHICLE		10	10	9		✓	
WEAPON		10	10	9		✓	
TEMPORAL	DATE	10	10	9	✓	✓	
TEMPORAL	TIME	10	10	9	✓	✓	
IDENTIFIER	EMAIL	10	10	9	✓	✓	
IDENTIFIER	URL	10	10	9	✓	✓	
IDENTIFIER	DOMAIN_NAME	10	10	9		✓	
IDENTIFIER	IP_ADDRESS	10	10	9		✓	
IDENTIFIER	PHONE_NUMBER	10	10	9	✓	✓	
IDENTIFIER	FAX_NUMBER	10	10	9		✓	
IDENTIFIER	PERSONAL_ID_NUM	10	10	9	✓	✓	
IDENTIFIER	LATITUDE_LONGITUDE	10	10	9	✓	✓	
IDENTIFIER	CREDIT_CARD_NUM	10	10	9	✓	✓	
IDENTIFIER	MONEY	10	10	9	✓	✓	
IDENTIFIER	PERCENT	10	10	9	✓	✓	
IDENTIFIER	UTM	10	10	9	✓	✓	
RELIGION		10	10	9			
NATIONALITY		10	10	9			

In the Editor list display, you can edit the names for user-defined types and subtypes, and the weights associated with the three sources for each entity type.

F.8.3. Adding Named Entities with Gazetteers

In the Editor list, double click the type/subtype for which you want to define new entities by listing them (in a gazetteer) or defining a pattern as a regular expression.



Type: TEMPORAL

Subtype: DATE

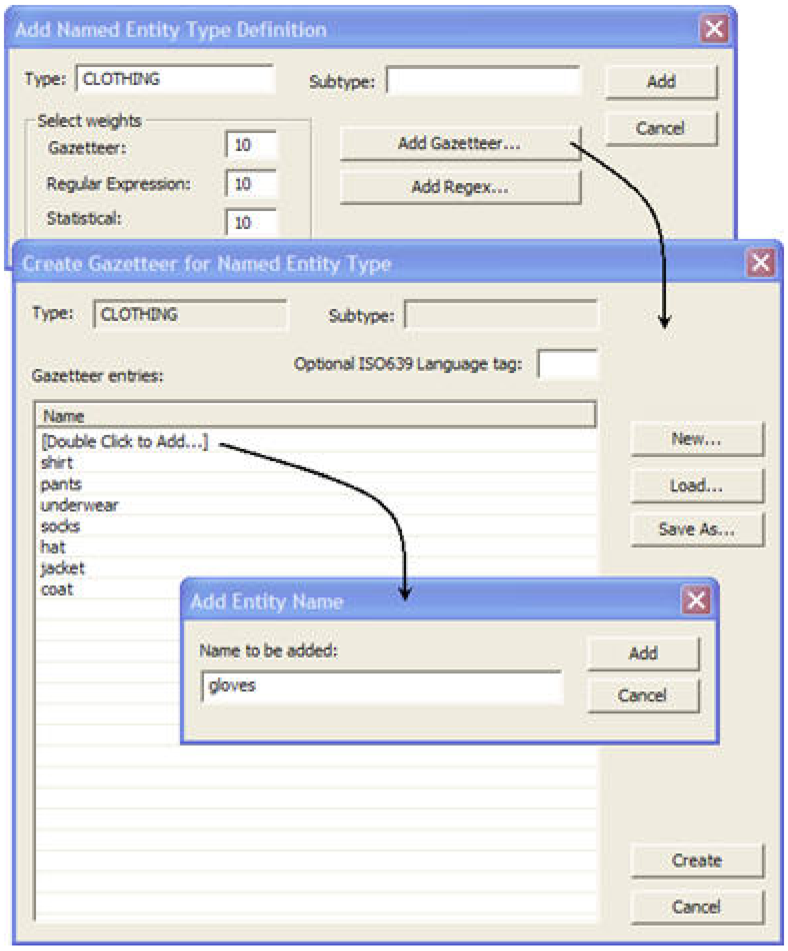
Select weights

Gazetteer: 10

Regular Expression: 10

Statistical: 9

Double click **[Double Click to Add...]** to add an entry to the gazetteer or double click an entry to edit it.



In the example above, the user has created a gazetteer for the CLOTHING type, and is adding another entry.

Click **Load** to open a gazetteer you have already created for this type.

Click **New** to flush the contents of the list and start a new gazetteer.

F.8.4. Gazetteer Options

At the **Edit Named Entities** dialog, check the types of normalization to apply when matching input to gazetteer entires.

Normalize space. Normalize consecutive whitespaces to a single space.

Normalize case. Normalize text to lower case.

Normalize Kana. Convert Japanese Hiragana characters to Katakana.

Normalize Width. Normalize half-width and full-width characters to generic-width.

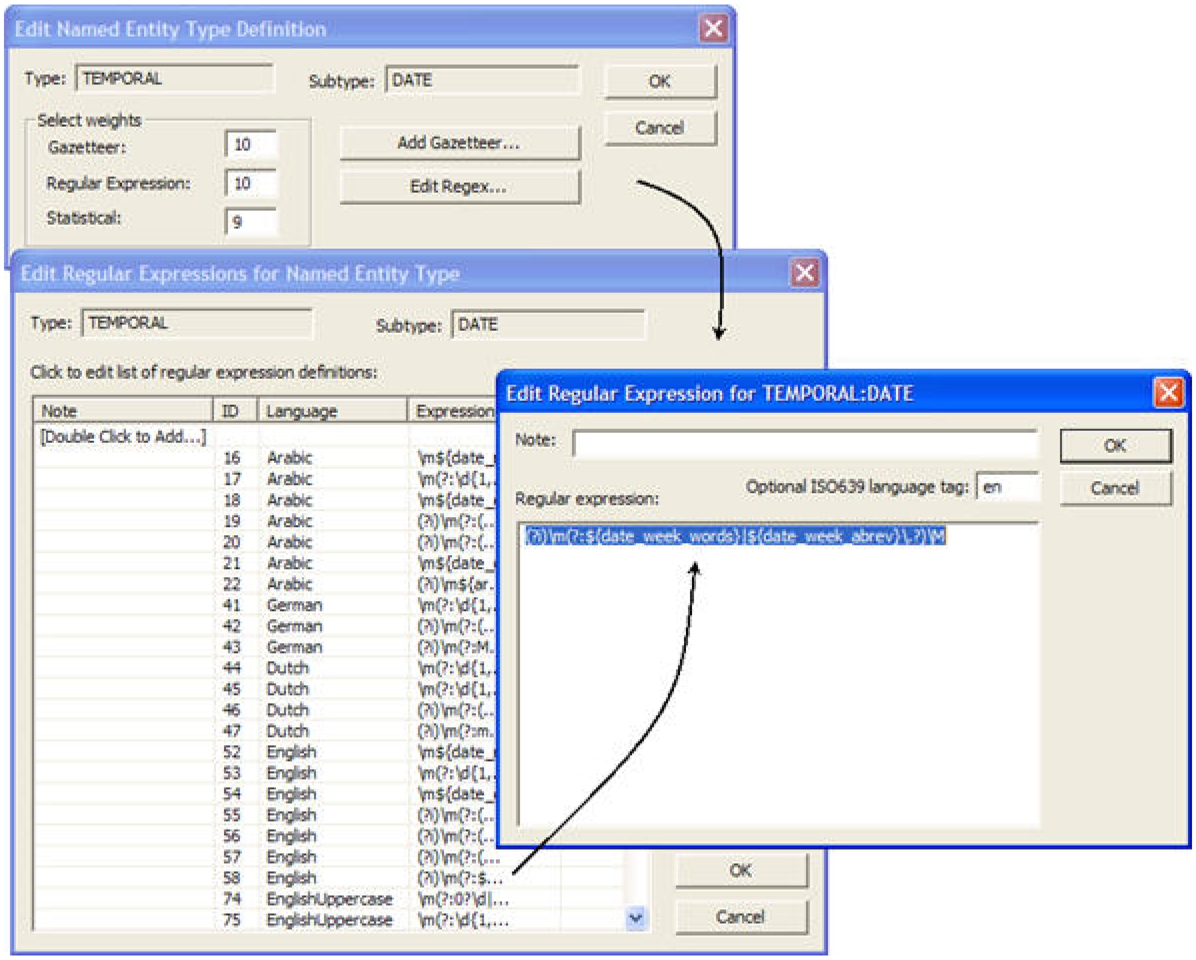
Normalize Diacritics. Strip diacritics and accent marks.

F.8.5. Adding Named Entities with Regular Expressions

Add or edit regular expressions for any entity type by double clicking on a type/subtype.

RLP supports Tcl regular expressions. See TclRegular Expression Syntax [239] .

With the Named Entities Editor, you cannot view or edit any of the `define` *elements, such as* `${time_ampm}` *specified in* ***regex-config.xml*** *and used in regular expressions.*


Edit Named Entity Type Definition

Type: TEMPORAL   Subtype: DATE

Select weights
Gazetteer: 10
Regular Expression: 10
Statistical: 9

Edit Regular Expressions for Named Entity Type

Type: TEMPORAL   Subtype: DATE

Click to edit list of regular expression definitions:

Note	ID	Language	Expression
[Double Click to Add...]			
	16	Arabic	\m${date_
	17	Arabic	\m(?:\d{1,
	18	Arabic	\m${date_
	19	Arabic	(?i)\m(?:(..
	20	Arabic	(?i)\m(?:(..
	21	Arabic	\m${date_
	22	Arabic	(?i)\m${ar..
	41	German	\m(?:\d{1,
	42	German	(?i)\m(?:(..
	43	German	(?i)\m(?:M..
	44	Dutch	\m(?:\d{1,
	45	Dutch	\m(?:\d{1,
	46	Dutch	(?i)\m(?:(..
	47	Dutch	(?i)\m(?:m..
	52	English	\m${date_
	53	English	\m(?:\d{1,
	54	English	\m${date_
	55	English	(?i)\m(?:(..
	56	English	(?i)\m(?:(..
	57	English	(?i)\m(?:(...
	58	English	(?i)\m(?:$...
	74	EnglishUppercase	\m(?:0?\d|...
	75	EnglishUppercase	\m(?:\d{1,...

Edit Regular Expression for TEMPORAL:DATE

Note:

Optional ISO639 language tag: en

Regular expression:

```
(?i)\m(?:${date_week_words}|${date_week_abrev}\.?)\M
```

Optionally, enter a note to track the details of a given regular expression. Notes are saved in **regex-config.xml**.

To indicate that a regular expression is language-specific, enter the ISO639 language tag:

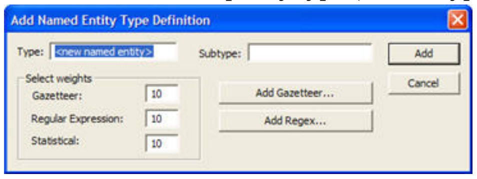
Language	**Tag**
Arabic	ar
Chinese - Simplified	zh_sc
Chinese - Traditional	zh_tc
Czech	cs
Dutch	nl
English	en
Upper-Case English[a]	en_uc
Farsi (Persian)	fa
French	fr

Language	**Tag**
German	de
Greek	el
Hungarian	hu
Italian	it
Japanese	ja
Korean	ko
Polish	pl
Portuguese	pt
Russian	ru
Spanish	es
Urdu	ur

[a]For more accurate processing of English text that is entirely upper case, use the `en_uc` language tag.

F.8.6. Adding New Named Entity Types

To define a new type, double click the first entry in the **Edit Named Entities** dialog's list: **[Double click to Add...]** and then specify type (and subtype), and create a gazetteer or add regular expressions.



Type: <new named entity>
Subtype:
Select weights
Gazetteer: 10
Regular Expression: 10
Statistical: 10

F.8.7. Deleting a User-Defined Named Entity Type

To delete a user-defined type, right click its entry in the list, and select **Delete** from the menu. **Note:** Builtin types (indicated by a check in the **Builtin** column) cannot be deleted or renamed.

F.9. Troubleshooting: the RLP Log

RLP maintains a trace log of significant events and diagnostics. To view the RLP log, select **View** → **Log**. The log may help you find and fix any problems you experience.

You can also *contact Basis Technology* for help.

F.10. Process Context Files

Each Demo process is defined by an RLP context file in **rlp\samples\w32demo**. A context file contains a list of RLP processors in the order in which they are to be applied to the input text. It may also include property settings.

Process	**Context File**
Loading the input text	**demo-minimal-context.xml**
Rosette Base Linguistics	**demo-bl-context.xml**
Base Noun Phrases	**demo-bnp-context.xml**

Process	**Context File**
Rosette Entity Extractor	**demo-ne-context.xml**
Rosette Chinese Script Converter (Simplified to Standard)	**demo-csc-s2t-context.xml**
Rosette Chinese Script Converter (Standard to Simplified)	**demo-csc-t2s-context.xml**

These files can be modified. See Defining an RLP Context [18] .

Glossary

In the glossary below, terms that have a specific meaning for RLP are capitalized, e.g., "Context." Non-capitalized terms are general linguistic or computing terms.

A

abbreviation	An abbreviation is a shortened way to write a long word or phrase. For example, the word "miscellaneous" is frequently replaced with the abbreviation "misc."
adjectival	An adjective that describes a word functioning as an adjective.
adjectival noun	An adjective functioning as a nominal. See Also nominal.
adjective	An adjective is a word that modifies a noun to denote quantity, extent, quality, or to specify the noun as distinct from something else. Consider the sentence, "The smart woman has a great job." The words "smart" and "great" are adjectives, which describe "the woman" and the "job," respectively.
adverb	An adverb is a word that modifies a verb to describe how the verb was performed. Consider the sentence, "The racer drove quickly." The word "quickly" is an adverb that describes the driving of the racer. Adverbs may also modify whole sentences. Consider the sentence, "Frankly, I don't care." The word "frankly" is an adverb that describes the rest of the sentence.
affix	One or more sounds or characters attached to the beginning, middle, or end of a word or base to create a derived word or inflectional form.
alternative readings	Alternative readings are returned in Japanese when the recognized word has more than one valid pronunciation.
auxiliary verb	A verb that is used in forming certain tenses, aspects and moods of other verbs. Consider the sentence, "The man is running." The verb "is" is an auxiliary verb to the main verb "run." The main difference in English between "runs" and "is running" is aspectual (habitual vs. progressive). Also, the difference between "saw" and "was seen," with the auxiliary verb "was," is mood (active vs. passive).

B

bopomofo	A method for transcribing Chinese text using Chinese character elements for their reading value. For more information, see Ken Lunde’s *CJKV Information Processing*, O’Reilly 1999. Also called zhuyin fuhao.
broken plurals	Refer to a class of nouns whose plural is formed in an irregular way. Use is specific to Semitic languages such as Arabic.
bound morpheme	A bound morpheme is a morpheme that cannot stand alone and must be combined with some other morpheme. An affix is an example of a bound morpheme. See Also morpheme.

BT_BUILD	`BT_BUILD` designates the platform on which **RLP** runs. It is embedded in the name of the RLP SDK package that you use to install **RLP**, and it is used to name platform-specific subdirectories, such as ***BT_ROOT*/rlp/bin** and ***BT_ROOT*/rlp/lib**. See Supported Platforms and BT_BUILD Values [13] .
BT_ROOT	`BT_ROOT` designates the Basis root directory, the directory where the **RLP SDK** is installed. During initialization, an RLP application must set the path to `BT_ROOT`.

C

choseong	Leading consonants or syllabic initial jamo in written Korean.
compound analysis	Dissects a compound word (a word composed of many words combined) into its constituent pieces. For example, in German, the word '"Bibliothekskatalogen" (library catalog) can be decompounded into "Bibliothek" and "Katalog".
conjunction	A conjunction is word that links two phrases, such as the words "and" and "or" in English.
consonant-stem verb (五段動詞)	A category of Japanese verbs whose stems end in consonants. Some examples of these verbs are *hakob(u)* 運ぶ, *kak(u)* 書く, and *shir(u)* 知る. An easy way to tell if a Japanese verb is a consonant-stem is to look at the Romanization of the direct-style negative form of the verb. If the letter preceding the suffix *-anai* is a consonant, it is a consonant-stem verb.

Lemma			Negative Direct-Style			Stem
hakob(u)	運ぶ	"carry"	*hakob(anai)*	運ばない	"don't carry"	*hakob-*
kak(u)	書ó	"write"	*kak(anai)*	書かない	"don't write"	*kak-*
shir(u)	知る	"know"	*shir(anai)*	知らない	"don't know"	*shir-*

Context	RLP context is an XML document that defines a sequence of language processors to apply to the input text. It may include property settings to customize the processing.
copula verb	A copula is a form of the verb "to be" used for equating two phrases. In Japanese, copula refers to the word *da* and its various forms, which is used with some Japanese nouns and adjectives.
count noun	A count noun is a noun that has a plural form. Most nouns in English are count nouns. See Also unit noun, mass noun.

D

decomposing	Decomposing refers to taking tokens and further breaking them down into smaller constituent parts where possible. See Also segmentation.
decompounding	See decomposing.

determiner	A determiner is a word which specifies a particular noun phrase. For example, in English, an article such as "the" ("the book").
direct-style	Refers to a politeness level in Japanese, which shows intimacy between the speaker and listener. Direct-style is the opposite of distal style. See Also distal-style.
distal-style	Refers to a politeness level in Japanese, which shows distance (and thus politeness and deference to the listener) between the speaker and listener. The distal-style is marked by verb endings containing *-mas-*, and use of the copula *desu* instead of *da*. Distal-style is the opposite of direct-style. See Also direct-style.

E

Environment	The RLP environment is an XML document that represents the global state of RLP. The environment is responsible for loading and maintaining the various language processors authorized by the software license.
eojeol	Eojeol are written syllables in Korean composed of jamo.
eumjeol	Eumjeol are a space delimited sequence of eojeol in Korean Hangul writing. See Also Hangul, eojeol.

F

FACILITY	A Named Entity type. See Named Entity Definitions [49] .
full-width	When describing a character, "full-width" refers to characters in the ASCII character set which appear "wider" than their ASCII versions. The ASCII versions are often called half-width. The table below shows a few full-width and half-width characters for comparison.

half-width	ABC 123 %&*
full-width	ＡＢＣ　１２３　％＆＊

In Japanese, there are full-width (zenkaku) and half-width (hankaku) characters in Katakana. However, full-width Katakana is considered "normal" whereas full-width ASCII is "not-normal." The table below shows a few zenkaku and hankaku characters for comparison.

zenkaku Katakana	インターネット
hankaku Katakana	ｲﾝﾀｰﾈｯﾄ

fully-productive derivational suffix	Fully-productive means that one need not remember which words can be used with the suffix — anyplace the suffix is used is considered valid. The best example of a fully-productive derivation suffix in English is *-ness*, which can be used with any noun.
furigana	In Japanese, *furigana* refers to the Katakana or Hiragana characters used to show the pronunciation of a Japanese word containing Kanji. Also known as *yomigana* or *rubi*.

	See Also Kanji.

G

Gazetteer	A list of words or phrases that share a certain property. Gazetteers are used to extend support for named entities. For example, a gazetteer could contain a list of companies that provide a particular service. See Gazetteer [144] language processor and Gazetteer source files [53] .
glyph	A glyph is a graphical representation of a character. In a language such as Arabic, a character's appearance varies depending on whether it is shown alone (isolated) or in a given position within a word (initial, middle or final). Thus multiple glyphs are mapped to the codepoint for any given Arabic character.
GPE.	Geo-Political Entity. A Named Entity type. See Named Entity Definitions [49] .

H

half-width	Half-width refers to characters which would appear as they do in the ASCII encoding. The opposite of "half-width" is "full-width." See Also full-width.
Han characters	Han characters refers generally to the Chinese ideographic characters which are used in Chinese, Japanese, and Korean. ós an example of two Han characters.
Hangul	Hangul is the natively created phonetic Korean script.
hanja	Hanja is the Korean word referring to Chinese ideographic characters used in Korean.
hankaku	See half-width.
hanyu pinyin	This system of pinyin uses Roman letters to express the pronunciation of Chinese words. This system is used widely in Mainland China. See Also pinyin.
Hanzi	Hanzi is the Chinese word referring to Chinese ideographic characters used in Chinese.
harakat	Vowel markers in Arabic script.
Hiragana	The Japanese phonetic alphabet used to write native Japanese words (as opposed to Katakana, the alphabet used for borrowed foreign words).
honorific prefix	A set of characters or sounds added to the beginning of a word to make it into an honorific word. For example, in Japanese, the prefix *o-*/*go-* ós added to the beginning of some nouns to make them honorific.
honorific suffix	A set of characters or sounds added to the end of a word make the word honorific. For example, in Japanese, the suffix *-sama* ós added to the personal name when addressing a person.

I

IDENTIFIER:UTM	Univeral Transverse Mercator. A Named Entity type. See Named Entity Definitions [49] .
interjection	An interjection is an exclamation usually expressing an emotion. The English interjection "Ow!" may express pain.
irregular verb	An irregular verb is one which does not follow the regular conjugation patterns of the other verbs in the language.
Iterator	Used to access the results generated by the processors in the context.

J

jamo	Jamo are the basic phonetic building blocks for writing Korean in Hangul. Jamo consists of 10 basic vowels and 14 consonants. The combination of jamo forms a syllable called an eojeol. See Also eojeol.
jongseong	Trailing consonants or syllabic final jamo in written Korean.
jungseong	Vowels or "syllabic peak" jamo in written Korean.

K

kana	Kana refers to the Japanese alphabets Hiragana and Katakana collectively. See Also Hiragana, Katakana.
Kanji	Chinese ideographs used in the Japanese language are called Kanji. An example of Kanji: 日本語.
kashida	A kashida is a typographical embellishment in Arabic used to justify text by elongating certain characters at specified points. Also called "tatweel."
Katakana	The Japanese phonetic alphabet used to write borrowed foreign words. See Also Hiragana.

L

Language Processor	Processes input text given to RLP and generates analytical results for use by other language processors and the RLP application.
lemma	A lemma is the form of the word a person would look for when looking it up in a dictionary, i.e., the "dictionary form" of a word. In English, it is the singular of nouns ("book," not "books") and the "base form" of verbs ("define," not "defines," "defining," or "defined"). Other languages have somewhat different notions of what the "dictionary form" or the "base form" is, but whatever form it takes, it is the lemma.
lemmatization	The act of finding the lemma of a word.

lexeme	A lexeme is an item in the vocabulary of a language, frequently called a word.
lexicalized	A word that has entered the language as a single word, where the meaning of the whole word no longer relates to the meaning of the constituent parts of the word, is said to be lexicalized.
lexicon	A listing of all the words in a language with information about its form, meaning, and part of speech.
LOCATION	A Named Entity type. See Named Entity Definitions [49] .

M

Main Dictionary	The primary lexicon shipped with a language analyzer.
mass noun	A mass noun is a type of noun that does not have a plural form. In English, "water" is a typical mass noun; instead of making it plural, one must use a measure word (e.g. "two glasses of water"). In Japanese and Chinese, every noun is a mass noun. See Also unit noun, count noun.
measure	A measure is a word expressing a unit of measurement, such as "meters" or "kilograms."
morpheme	A morpheme is the smallest unit for building words (composed of characters or sounds). That is, morphemes cannot be broken down any further into meaningful parts. Note: All morphemes are lexemes, but not all lexemes are morphemes. E.g., book + s = books. "book" and "s" are morphemes. See Also lexeme, bound morpheme.
Multilingual Language Identifier	See Rosette Language Boundary Locator.

N

named entities	The names of persons, places, times or things: "Bill Gates", "New York City", "11/22/63" and "Harvard University" are all named entities.
named entity extraction	Named entity extraction is the act of extracting known types of entities such as personal names, organization corporate names, and geographical names from a stream of text.
NATIONALITY	A Named Entity type. See Named Entity Definitions [49] .
nominal	A word in Japanese that functions as a noun.
normalization	The act of removing variations from words due to spelling or writing conventions that are not significant from the point of view of information retrieval.
noun	A noun is a word expressing a person, place or thing. See Also adjectival noun, temporal noun, unit noun, verbal noun (サ変動詞).
numeric	Numeric describes a token or word containing a number either in Arabic numerals (0, 1, 2, 3, ...), Arabic-Indic numerals (used in Arabic script), or characters that represent numbers.

O

onomatope	A word that expresses a sound. For example, in English "glug, glug" is the onomatope for the sound of someone drinking.
ordinal numeric	An ordinal numeric is a number that designates a place in a sequence, such as "first," "second," etc.
ORGANIZATION	A Named Entity type. See Named Entity Definitions [49] .

P

part-of-speech	Each language has several dozen parts of speech; each part of speech (POS) describes how a word can combine with other words in that language. A part of speech tag (POS tag) can be assigned to each token.
particle	A particle is a word that indicates the role of a preceding or following word in a sentence. For example, in Japanese, the particle *wa* (ϕ́)appears after the subject of a sentence.
PERSON	A Named Entity type. See Named Entity Definitions [49] .
pinyin	Pinyin is a system that uses the Roman alphabet used to show the pronunciation of Chinese words. There are several types of pinyin. See Also hanyu pinyin.
prefix	A bound morpheme that attaches to the beginning of a word or base to create a derivational word or inflectional form. See Also bound morpheme, morpheme.
preposition	A preposition is a word that appears before a noun phrase and combines to form a phrase. It expresses the relation that the noun phrase has with the rest of the sentence. For example, the phrase "under the boardwalk" contains the preposition "under".
postposition	A postposition is the same as a preposition, except that it appears after a noun phrase instead of before it.
pronoun	A pronoun is a word that refers to a noun. Some English pronouns are "he," "she," and "it."
proper noun	A proper noun is the name of a person, place, or entity. For example, "London", "Elizabeth", and "Basis Technology" are proper nouns. In English, proper nouns are always capitalized.

R

regular expression	A way of writing a pattern that a string can match or not. Regular expressions are written in Tcl syntax, which is based on the 1003.2 spec and some (not all) of the Perl5 extensions. For example, "ab*a" matches a string that starts and ends with an "a", with zero or more occurrences of "b" in the middle. For the details, see Tcl Regular Expression Syntax [239] .

RELIGION	A Named Entity type. See Named Entity Definitions [49] .
Romanization	Transcription or transliteration of text from another alphabet or writing script to the Latin alphabet. See Also transcription, transliteration.
Rosette Language Boundary Locator	Formerly known as the Multilingual Language identifier (MLI), RLBL detects boundaries between areas of different languages in text containing multiple languages and between areas of different scripts.

S

script	A script is a system of writing that is associated with a language. Hiragana, Katakana, and Kanji are examples of written scripts used in the Japanese language, but which are not, by themselves, languages.
segmentation	The act of dividing a stream of text into segments or tokens. Broadly speaking, RLP uses a "longest match" principle to segment text. See Also decomposing, token.
sentence boundary detection	Uses the language-based rules of punctuation ito determine the end of sentences. In many languages (such as Chinese and Japanese), end-of-sentence punctuation is largely unambiguous, and so sentence boundary detection is relatively easy. In other languages, such as English and most European languages, sentence-ending punctuation can also be used for other purposes. For example, periods can be used in English to end sentence, to mark abbreviations, or in some cases do both at the same time. A sentence boundary detector for English, then, has to distinguish among all the uses of sentence-ending punctuation and decide when it is actually ending a sentence.
stem	A stem is the hypothetical form from which all the forms of a word are created. Frequently, the stem is not an independent word that can stand alone. For example, the English verb "decode" has the following observed forms: "decode," "decoding," "decodes," and "decoded." The stem is "decod"; it clearly is a string from which the observed words are built, but the stem itself is never seen as a real word.
stemming	The process of removing prefixes and suffixes from a word until only a "stem" remains. Note that a stem is not necessarily the same as the lemma (dictionary form) of the word. See Also lemma, stem.
stopword	In the context of information retrieval, a stopword is a word that appears so frequently in a language that it is statistically insignificant when performing a search. For example, in English, the words "a" and "the" would appear in virtually any document, and thus generally speaking would not aid in finding a specific category of documents. Some information retrieval systems will choose to ignore stopwords when performing a search.
suffix	A bound morpheme that attaches to the beginning of a word or base to create a derived word or inflectional form. See Also fully-productive derivational suffix.
suru-verb (サ変動詞)	A suru-verb is a Japanese verb that is formed of a one character plus the verb *suru*(する) or other related forms such as *siru*(しる), which means "to do." Usually

	the first character does not stand alone as a word, although it may in some cases. For example, *ai(suru)* (愛する), *kan(suru)* (関する), and *syou(jiru)* (生じる) are suru-verbs.

T

tatweel	See kashida.
temporal noun	A noun expressing a time, such as "February."
token	A string (i.e., a sequence of characters) that corresponds to an irreducible lexical or typographical element in a language.
Tokenizer	Language processor that parses a document and generates a sequence of tokens.
transcription	The representation of the sound of the words in a language using another alphabet or set of symbols created for that purpose. Transcription is not concerned with representing characters; it strives to give a phonetically (or phonologically) accurate representation of the word. This may differ, depending on the language or set of symbols into which the word is being transcribed.
transliteration	The spelling of the words in one language and writing script with characters from another writing script. Ideally, transliteration is a character-for-character replacement so the reverse transliteration into the original script is possible. Transliteration is not concerned with representing the phonetics of the original; it only strives to accurately represent the characters.

U

unit noun	A unit noun is what is also called sometimes called a "counter," "classifier," or "measure word." In some languages, objects cannot be counted directly in the way English never counts bread except by a unit (e.g., a loaf). In English, one would never say "two breads" but rather "two loaves of bread." In that case, "loaf/loaves" is a unit noun. See Also mass noun.
User Dictionary	A language dictionary created and maintained by the user for use by a language analyzer in conjunction with the standard dictionary or dictionaries that RLP provides for that language. Depending on the language analyzer, the dictionary may provide segmentation or morphological information. Currently supported for Chinese [133] , Japanese [147] , Korean [151] , and Base Linguistics Analyzer [129] (European languages). See Creating User Dictionaries [193] .

V

verbal noun (サ変動詞)	A verbal noun is a verb composed of a noun with two or more characters and the verb *suru* (する) ("to do"). For example, *benkyou suru* (勉強する) means "to study" and is composed of the word for "study" with the verb *suru*. Because the noun can stand alone, it may be segmented from the verb *suru*. See Also suru-verb (サ変動詞).

vowel-stem verb (一段動詞)

This is a category of Japanese nouns whose stems end in a vowel. Some examples of these are *tabe(ru)* (ɸ), *i(ru)* (ɸ), and *age(ru)* (ɸ). An easy way to tell if a Japanese verb is a vowel-stem, is to look at the romanization of the direct-style negative form of the verb. If the letter preceding the suffix *-nai* is a vowel, then it is a vowel-stem verb.

Lemma			**Negative Direct-Style**			**Stem**
tabe(ru)	食べる	"eat"	*tabe(nai)*	食べない	"don't eat"	*tabe-*
i(ru)	居る	"be (animate being)"	*i(nai)*	居ない	"don't be"	*i-*
age(ru)	上げる	"give"	*age(nai)*	上げない	"don't give"	*age-*

Y

yomigana

See furigana.

Z

zenkaku

See half-width.

zhuyin fuhao

See bopomofo.

Index