Rosette Base Linguistics (RBL) Application Developer's Guide Java Edition

7.41.1.c65.0

Publication date 2021-07-15

About Basis Technology

Verifying identity, understanding customers, anticipating world events, uncovering crime. For over twenty years, Basis Technology has provided the underlying analytical components enabling businesses and governments to tackle some of their toughest problems. Our Rosette™ text analytics platform employs a hybrid of classical machine learning and deep neural nets to extract meaningful information from unstructured data. Autopsy, our digital forensics platform, and Cyber Triage, our incident response tool, serve the needs of law enforcement, national security, and legal technologists with over 5,000 downloads every week. KonaSearch delivers natural language search of every field, object, and file in Salesforce all from the same index.

Company headquarters are in Somerville, Massachusetts, with branch offices in Washington, London, Tel Aviv, and Tokyo. For more information, visit www.basistech.com.

Basis Technology Corp.
1060 Broadway
Somerville, MA 02140
T 617.386.2000
F 617.386.2020
E support@rosette.com
http://support.rosette.com

Table of Contents

1. Introduction

Rosette Base Linguistics (RBL) provides a set of linguistic tools to prepare your data for analysis. Language-specific modules provide base forms (lemmas) of words, parts-of-speech tagging, compound components, normalized tokens, stems, and roots. RBL also includes a Chinese Script Converter [43] (CSC) which converts tokens in Traditional Chinese text to Simplified Chinese and vice versa.

1.1. Using RBL

You can use RBL in your own JVM application, use its Apache Lucene compatible API in a Lucene application, or integrate it directly with either Apache Solr or Elasticsearch. [1]

- **JVM Applications**
  To integrate base linguistics functionality in your applications, the JVM includes two sets of Java classes and interfaces:
  - ADM API: A collection of Java classes and interfaces that generate and represent Rosette's linguistic analysis as a set of annotations. This collection is called the Annotated Data Model (ADM) and is used in other Rosette tools, such as Rosette Language Identifier and Rosette Entity Extractor, as well as RBL. There are some advanced features which are only supported in the ADM API and not the classic API. When using the ADM API, you create an annotator which includes both tokenizer and analyzer functions.
  - Classic API: A collection of Java classes and interfaces that generate and represent Rosette's linguistic analysis that is analogous to the ADM API, except it is not compatible with any other Rosette products. It also supports streaming: a user can start processing a document before the entire document is available and it can produce results for pieces of a document without storing the results for the entire document in memory at once.
    When using the classic API, you create tokenizers and analyzers.
- **Lucene**
  In an Apache Lucene application, you use a Lucene analyzer which incorporates a Base Linguistics tokenizer and token filter to produce an enriched token stream for indexing documents and for queries.
- **Solr**
  With the Solr plugin, an Apache Solr search server uses RBL for both indexing documents and for queries.
- **Elasticsearch**
  Install the Elasticsearch plugin to use RBL for analysis, indexing, and queries.

NOTE
The Lucene, Solr, and Elasticsearch plugins use APIs based on the classic API. All options that are in the Enum classes `TokenizerOption` or `AnalyzerOption` are available along with some additional plugin specific options.

[1] Apache Lucene™, Lucene™, Apache Solr™, and Solr™ are trademarks of the Apache Software Foundation. Elasticsearch™ is a trademark of Elasticsearch BV.

1.2. Tokenizers and Analyzers

Most of the linguistic work is performed by tokenizers and analyzers. Tokenizers identify the words in text. Analyzers determine their linguistic attributes such as lemmas and parts of speech.

Depending on the language, each analysis object may contain one or more of the following:

- Lemma
- Part of Speech
- Normalized Token
- Compound Components
- Readings
- Stem
- Semitic Root

For some languages, the analyzer can disambiguate [23] between multiple analysis objects and return the disambiguated analysis object.

In the ADM API, the `BaseLinguisticsFactory` sets the linguistic options and instantiates an `Annotator` which annotates the input text with the linguistic objects.

In the classic API, the `TokenizerFactory` and `AnalyzerFactory` configure and create these objects.

1.3. Annotator - ADM API

Basis Technology has created a collection of Java classes that generate and represent Rosette's linguistic analyses as a set of annotations. This collection of Java classes is called the Annotated Data Model (ADM) and may be used in RLI and REX as well as in RBL.

The ADM provides advanced functionality not available with the classic API.

When using the ADM API, you use `BaseLinguisticsFactory` to set the options and instantiate an `Annotator`. Depending on the analysis and the language, you may get information about sentences, layout regions, tokens, compounds, and readings.

The ADM API supports more options than the classic API. Whenever possible, we recommend using this API.

For complete API documentation of the ADM, consult the Javadoc for the package:

- `com.basistech.rosette.dm`

1.3.1. ADM Usage Pattern

The standard procedure for using the ADM is as follows:

- Create an Annotator [3].
- Use the Annotator to annotate the input text.
- Get the analytical data you want from the AnnotatedText [3].

Create an Annotator

Use `BaseLinguisticsFactory` to set the `BaseLinguisticsOptions` and to instantiate an `Annotator`. Options may be set on the factory itself or passed in to the `create` method.

1. At the minimum, you should set options for `rootDirectory` and `language`.

TIP
If the license is not the default directory (`${rootDirectory}/licenses)`, you need to pass in the `licensePath`.

```
BaseLinguisticsFactory factory = new BaseLinguisticsFactory();
factory.setOption(BaseLinguisticsOption.rootDirectory, rootDirectory);
File rootPath = new File(rootDirectory);
factory.setOption(BaseLinguisticsOption.licensePath,
        new File(rootPath, "licenses/rlp-license.xml").getAbsolutePath());
```

2. Then use the `BaseLinguisticsFactory` to create the `Annotator`. This sample sets the language to English (`eng`).

```
Annotator annotator;
EnumMap<BaseLinguisticsOption, String> options = new EnumMap<>(BaseLinguisticsOption.class);
options.put(BaseLinguisticsOption.language, "eng");
annotator = factory.createSingleLanguageAnnotator(options);
```

Annotate

Use `Annotator` to annotate the input text, which returns a `AnnotatedText` object.

The `AnnotatedText` object provides an API for gathering data from the linguistic analysis that RBL performs during the annotation process. Depending on the analysis and the language, you may get information about sentences, layout regions, tokens, compounds, and readings.

`getTokens()` returns a list of tokens, each of which contains a list of morphological analyses.

```
AnnotatedText results = annotator.annotate(getInput(inputFilePathname));
int index = 0;
for (Token token : results.getTokens()) {
    outputData.format("token %d:\t%s%n", index, token.getText());
    int aindex = 0;
    List<MorphoAnalysis> analyses = token.getAnalyses();
    if (null != analyses) {
        outputData.format("\tindex\tlemma\tpart-of-speech%n");
        for (MorphoAnalysis ma : analyses) {
            outputData.format("\t%d    %s\t%s%n", aindex, ma.getLemma(),
                    ma.getPartOfSpeech());
             aindex++;
        }
    }
    index++;
}
```

1.4. Classic API

When using the classic API, you instantiate separate factories for tokenizers and analyzers.

- The `TokenizerFactory` produces a language-specific tokenizer that processes documents, producing a sequence of tokens.
- The `AnalyzerFactory` produces a language-specific analyzer that uses dictionaries and statistical analysis to add analysis objects to tokens.

If your application requires streaming, use this API. The Lucene, Solr, and Elasticsearch integrations use these factories.

For the complete API documentation, consult the Javadoc for these classes.

- `TokenizerFactory` [4]
- `AnalyzerFactory` [4]

1.4.1. TokenizerFactory

Use the `TokenizerFactory` to create a language-specific tokenizer that extracts tokens from a plain text source. Prior to using the factory to create a tokenizer, you use the factory with `TokenizerOption` to define the root of your RBL installation, as illustrated in the following sample. See the Javadoc for other options you may set.

TIP

If the license is not the default directory (`${rootDirectory}/licenses)`, you need to pass in the `licensePath`.

The `Tokenizer` uses a word breaker to establish token boundaries and detect sentences. For each token, it also provides offset information, length of the token and a tag. Some tokenizers calculate morphological analysis information as part of the tokenization process, filling in appropriate analysis entries in the token object that they return. For other languages, you use the analyzer described below to return analysis objects for each token.

Create a tokenizer factory

```
TokenizerFactory factory = new TokenizerFactory();
factory.setOption(TokenizerOption.rootDirectory, rootDirectory);
File rootPath = new File(rootDirectory);
factory.setOption(TokenizerOption.licensePath,
        new File(rootPath, "licenses/rlp-license.xml").getAbsolutePath());
```

Set tokenization options

```
factory.setOption(TokenizerOption.nfkcNormalize, "true");
```

1.4.2. AnalyzerFactory

Use the `AnalyzerFactory` to create a language-specific analyzer. Prior to creating the analyzer, use the factory and `AnalyzerOption` to define RBL root, as illustrated in the sample below. See the Javadoc for other options you may set.

TIP

If the license is not the default directory (`${rootDirectory}/licenses)`, you need to pass in the `licensePath`.

```
AnalyzerFactory factory = new AnalyzerFactory();
factory.setOption(AnalyzerOption.rootDirectory, rootDirectory);
File rootPath = new File(rootDirectory);
factory.setOption(AnalyzerOption.licensePath,
        new File(rootPath, "licenses/rlp-license.xml").getAbsolutePath());
```

Use the `Analyzer` to return an array of `Analysis` objects for each token.

1.5. Multithreading

1.5.1. Annotator Management for Multithreaded Applications

For a multithreaded application it is recommended that a pool of annotators be constructed. Annotators must be used on a per-thread basis, and it is desirable to avoid the overhead of creating annotators each time one is required. The settings of annotators cannot be changed after they are built, so if multiple configurations are required they should be stored separately.

1.5.2. Classic API Threading

RBL integrates easily with multi-threaded architectures, but, to avoid performance penalties, it does not make promiscuous use of locks. Instead, most objects and interfaces in RBL are either read-only, reentrant objects or read-write, per-thread objects. `TokenizerFactory` and `AnalyzerFactory` objects are hybrids as they have both thread-safe and per-thread methods. The `create` methods of these factories are thread-safe because they do not alter any data within the factories, but the `setOption`, `addDynamicUserDictionary`, and `addUserDefinedDictionary` methods are not thread-safe because they do alter data within the factory. The tokenizers and analyzers created by these factories are always meant to be used on a per-thread basis because they are not reentrant and do alter data within the objects. You can use a factory across multiple threads to create objects as long as calls to the factory methods for setting options or adding user dictionaries are synchronized appropriately. The objects created by the factory must each be created and used by only one thread, which need not be the thread initializing the factory.

2. Getting Started

2.1. Minimum System Requirements

RBL is a Java SDK and works on any system with Java, including OpenJDK, installed.

- Java Runtime Environment 8, 9, or 11
- Ant 1.7.1 or later (optional - required to run sample applications with Ant build scripts)

2.2. Memory

The minimum Java heap required to run RBL smoothly is 1.5 GB. It has been observed that a `-Xmx` setting of 8 GB allows for optimal performance in significantly multithreaded environments. This memory setting accounts for the simultaneous use of multiple languages.

We also recommend that you set `-Xms` equal to your `-Xmx` setting. This prevents the JVM from having to grow the heap, which is time-consuming.

2.3. System Requirements for TensorFlow

RBL uses TensorFlow, a native library, for disambiguating Hebrew [30] when using a neural network for the disambiguator. Ubuntu 14.04+, Windows 7+, and macOS 10.11+ are fully supported, but you should be able to run the disambiguator successfully on other modern Linux flavors as well.

The TensorFlow library for Linux requires a system with:

- `libc.so.6` (`GLIBC`) 2.17 or newer
- `libstdc++.so.6` (`GLIBCXX`) 3.4.19 or newer
- `libgcc_s.so.1` (`GCC`) 3.0 or newer.

The version of TensorFlow included with RBL is configured to work on a broad range of systems, which requires it to avoid some features not available everywhere. Compiling TensorFlow for your specific system and using it on the classpath instead of the default `libtensorflow_jni-<tensorflowversion>.jar` will likely improve performance. A JAR with TensorFlow compiled with GPU support for Linux is available for download.

The neural network is used for Hebrew disambiguation when the `disambiguatorType` is set to `DNN`.

2.4. Installing RBL

Your installation of RBL will include the following files:

1. The SDK package: `rbl-je-<version>.zip`, where <version> is the version of RBL you are installing, e.g. `rbl-je-7.36.0.c62.2.zip`. When you unzip the SDK package, the root directory is `rbl-je-<version>`. It contains text files with license and copyright information, along with the following subdirectories:

   dicts — RBL binary dictionaries.

   lib — The JAR files that the SDK uses.
   The core SDK .jar files are `btrbl-je-<version>.jar` and `btcommon-api-<btcommonversion>.jar`. The SDK also uses the Simple Logging Facade for Java (SLF4J), `slf4j-api-<slf4jversion>.jar`. You should add these .jar files to your classpath.
   If you are using Lucene or Solr, you should also add the appropriate JAR file to your classpath. The versions [7] are dependent on the version of Lucene/Solr you are using.

   licenses — Default location for placing your license file, `rlp-license.xml`. The samples that accompany this SDK require `rlp-license.xml` to be in this directory.

   models — RBL binary models.

samples RBL sample files.
Sample text and query files in all supported languages are in `samples/data`. These files are used with the code samples.
See Running the Core Samples [11] and JapaneseAnalyzerSample [49].

tools Tools for generating user dictionaries and the RBLCmd line utility.

2. The Documentation: `rbl-je-<version>-doc.zip`
When you unzip the documentation package to the same location where you have unzipped the SDK package, the root directory contains a `doc` subdirectory containing:
   - Rosette Base Linguistics Application Developer's Guide (this document, `rbl-je-<version>-appdev-guide.pdf`)
   - Release Notes (`rbl-je-<version>-release-notes.pdf`)
   - Java API documentation (`apidocs/index.html`)
3. The license file: `rlp-license.xml`.
The samples expect to find it in `rbl-je-<version>/licenses`.

2.5. Lucene/Solr Versions

RBL contains version-specific files for the Lucene and Solr integrations. Many of these files support multiple versions of Lucene/Solr, as indicated in the table below.

Lucene/Solr Version	RBL File name version	Solr lib JAR file[a]	Lucene Sample Directory
4.3–4.8	4_3	`btrbl-je-lucene-solr-4_3-<version>.jar`	`rbl-je-<version>/samples/lucene-4_3`
4.9	4_9	`btrbl-je-lucene-solr-4_9-<version>.jar`	`rbl-je-<version>/samples/lucene-4_9`
4.10	4_10	`btrbl-je-lucene-solr-4_10-<version>.jar`	`rbl-je-<version>/samples/lucene-4_10`
5.0–5.5	5_0	`btrbl-je-lucene-solr-5_0-<version>.jar`	`rbl-je-<version>/samples/lucene-5_0`
6.0–6.1	6_0	`btrbl-je-lucene-solr-6_0-<version>.jar`	`rbl-je-<version>/samples/lucene-6_0`
6.2–8.9	6_2	`btrbl-je-lucene-solr-6_2-<version>.jar`	`rbl-je-<version>/samples/lucene-6_2`

[a] <version> is RBL version

2.6. Note on Logging

RBL uses the Logging Facade for Java (SLF4J) to log activities. The SLF4J API JAR is in `lib/`.

SFL4J is a façade for various logging APIs. Using SFL4J, the developer or an administrator can determine which one of many popular logging systems to use at runtime. In the `tools/lib` directory, we include the SLF4J binding JARs `log4j-1.2.17.jar` and `slf4j-log4j12-1.7.5.jar`. These JARs are used by our samples and `RBLCmd`. When you place all three of these JARs on your classpath, the logging façade is bound to the implementation, and RBL logging is turned on. As defined in the file `etc/log4j.properties`, by default, `INFO` messages are output to the console (`System.err`). As the Javadoc for `org.slf4j.impl.SimpleLogger` explains, you can use system properties or this properties file to output to a file and control other logging parameters.

If you want to use SLF4J with a different implementation, put the appropriate binding JAR files and properties file on your classpath.

2.7. Removing Unnecessary Files

Depending on the scope of your application, you may wish to remove unnecessary files to reduce the size of your application.

2.7.1. Tool Files

The tools directory contains files for:

- Building user dictionaries [34]
- Building CSC user dictionaries [46]
- The RBL command line utility [12]

You can delete some or all of the directories, as needed. For example, you may decide to delete the user dictionary directories, but keep the RBLCmd utility. If you don't need to build these dictionaries or use the RBL command line utility, you may freely delete the entire `tools` directory.

2.7.2. Language-Specific Model and Dictionary Files

You can remove files that represent languages your application does not need to support. Some languages require files for other language codes, either because they are canonicalized to the other language, or because there is some internal RBL requirement. Some languages require the files of more than one language.

When removing language files, be sure to check the deletion rules for the language and keep the files for all required language codes.

Dependent Languages

Language	Language Code	Required Language Code(s)
Chinese (Simplified)	`zhs`	`zho`
Chinese (Traditional)	`zht`	
Norwegian	`nor`	`nob`
Norwegian Bokmål	`nob`	
Afghan Persian	`prs`	`fas`
Western Persian	`pes`	
Russian	`rus`	`rus`, `eng`
Korean	`kor`	`kor`, `eng`

Additionally, if your distribution platform is of a particular endianness, you can remove the models of the opposite endianness. When applicable, the endianness of a file is given at the end of the file name; for example, the file `root/dicts/ara/dictLemmas-LE.bin` is a little-endian binary storing the Arabic lemma dictionary, whereas `root/dicts/ara/dictLemmas-BE.bin` is the same but stored in big-endian format.

- `root/dicts`: Any directory named after a language code (or “csc” for the Chinese Script Converter) and all its contents are used only for that language, and any file with “BE” or “LE” in its name is only used on big- or little-endian systems, respectively.

- `root/models`: Any directory named after a language code and all its contents are used only for that language.
- `root/contractions`: Any `.yaml` file whose name ends with a language code is used only for that language.
- `root/upt-16`: Any `.yaml` file whose name ends with a language code is used only for that language.

The files required for Japanese and Chinese depend on the value of the `tokenizerType` option, as shown in the table below.

tokenizerType	Files required
SPACELESS_LEXICAL	root/dicts/jpn/jla/*
	root/dicts/zho/cla/*
SPACELESS_STATISTICAL	root/models/jpn-model*.bin
	root/models/jpn.model
	root/models/zho.model

3. A Quick Look at RBL: Running a Sample Program

After you install RBL and the license file, try running a sample application. RBL contains a samples directory `rbl-je-<version>/samples`.

1. At a command prompt, navigate to `rbl-je-<version>/samples/<sampleName>`.
2. Use the Ant build script to compile and run the samples.

```
ant run
```

3.1. Initial and Path Options

If the option `rootDirectory` is specified, then the string `${rootDirectory}` takes that value in the `dictionaryDirectory`, `modelDirectory`, and `licensePath` options.

Initial and Path Options

Option	Description	Type (Default) (Default)	Supported Languages
`dictionaryDirectory`	The path of the lemma and compound dictionary, if it exists.	Path ${rootDirectory}/dicts	All
`language`	The language to process by analyzers or tokenizers created by the factory.	Language code	All
`licensePath`	The path of the RBL license file.	Path ${rootDirectory}/licenses/rlp-license.xml	All
`licenseString`	The XML license content, overrides `licensePath`.	String	All
`modelDirectory`	The directory containing the model files.	Path ${rootDirectory}/models	All

Option	Description	Type (Default) (Default)	Supported Languages
`rootDirectory`	Set the root directory. Also sets default values for other required options (`dictionaryDirectory`, `licensePath`, `licenseString`, and `modelDirectory`).	Path	All

Enum Classes:

- `AnalyzerOption`
- `BaseLinguisticsOption`
- `CSCAnalyzerOption` (except for `modelDirectory`)
- `TokenizerOption`

3.2. ADM Sample Application

A sample application that illustrates the use of ADM is in `rbl-je-<version>/samples/annotator-tokenize`.

1. In a Bash shell (Unix) or command prompt (Windows), navigate to `rbl-je-<version>/samples/annotator-tokenize`.
2. Use the Ant build script to compile and run the sample.

```
ant run
```

Your license (`rlp-license.xml`) must be in the `licenses` subdirectory of the RBL installation.

`AnnotatorTokenize` tokenizes the English string and provides one or more analyses with lemma and part-of-speech for each token.

The output appears in `annotator-tokenize.txt`.

```
length: 29
------
Some members spoke yesterday.

------
token 0: Some
index lemma part-of-speech
0 some QUANT
token 1: members
index lemma part-of-speech
0 member NOUN
token 2: spoke
index lemma part-of-speech
0 speak VPAST
1 spoke VI
2 spoke VPRES
3 spoke NOUN
token 3: yesterday
index lemma part-of-speech
0 yesterday ADV
1 yesterday NOUN
token 4: .
index lemma part-of-speech
0 . SENT
```

3.3. Classic API Sample Application

A sample application illustrating the use of the classic API is in `rbl-je-<version>/samples/tokenize-analyze`.

1. In a Bash shell (Unix) or command prompt (Windows), navigate to `rbl-je-<version>/samples/tokenize-analyze`.
2. Use the Ant build script to compile and run the sample.

   ```
   ant run
   ```

Your license (`rlp-license.xml`) must be in the `licenses` subdirectory of the RBL installation.

`TokenizeAnalyze` tokenizes the sample German document and provides a disambiguated analysis of each token.

The output appears in two files: `deu-tokenized.txt` and `deu-analyzed.txt`. The first file contains a token, **Tab**, `<ALNUM>` tag on each line, with a blank line following the end of a sentence.

The second file contains the token, lemma, part of speech, and compound components (where relevant) on each line. For those languages for which disambiguation is not supported,[2] there may be multiple rows for each token (the token appearing in the first column), one for each analysis. Here is a fragment with a sentence from `deu-analyzed.txt`:

[2]Or for Japanese, where disambiguation is turned off by default.

```
TOKEN                LEMMA                POS       COMPOUNDS
-----                -----                ---       ---------
3.11.06              3.11.06              CARD
-                    -                    PUNCT
Not                  Not                  NOUN
und                  und                  COORD
Elend                Elend                NOUN
in                   in                   PREP
ihren                ihr                  POSDET
Heimatländern        Heimatland           NOUN      [Heimat, Land]
lassen               lassen               VVFIN
immer                immer                ADV
mehr                 mehr                 INDADJ
Afrikaner            Afrikaner            NOUN
die                  der                  ART
Reise                Reise                NOUN
nach                 nach                 PREP
Europa               Europa               NOUN
antreten             antreten             VVINF
.                    .                    SENT
```

To run the samples with sample text in a different language, set the `test.language` parameter with the language code. For example to tokenize and analyze the Spanish sample, call

```
ant -Dtest.language=spa run
```

3.4. RBL Command Line Utility

`RBLCmd` is a general-purpose command line utility for RBL. It provides a simple way to produce RBL output without writing code. It is also useful for ad hoc speed and thread testing.

A Bash shell script (`RBLCmd`) and Windows script (`RBLCmd.bat`) for running this utility are in `rbl-je-<version>/tools/bin`. For more information, see `RBLCmd`'s on-line help, `RBLCmd -h`.

The command:

```
echo 'Hola' | ./tools/bin/RBLCmd -outputJson --language spa --rootDirectory . | jq
```

produces the following output:

```
{
  "version": "1.1.0",
  "data": "Hola\n",
  "attributes": {
    "sentence": {
      "type": "list",
      "itemType": "sentence",
      "items": [
        {
          "startOffset": 0,
          "endOffset": 5
        }
      ]
    },
    "scriptRegion": {
      "type": "list",
      "itemType": "scriptRegion",
      "items": [
        {
          "startOffset": 0,
          "endOffset": 5,
          "script": "Latn"
        }
      ]
    },
    "layoutRegion": {
      "type": "list",
      "itemType": "layoutRegion",
      "items": [
        {
          "startOffset": 0,
          "endOffset": 5,
          "layout": "STRUCTURED"
        }
      ]
    },
    "token": {
      "type": "list",
      "itemType": "token",
      "items": [
        {
          "startOffset": 0,
          "endOffset": 4,
          "text": "Hola",
          "analyses": [
            {
              "partOfSpeech": "INTERJ",
              "lemma": "hola",
              "raw": "hola[+INTERJ]",
              "tagSet": "BT_SPANISH"
            }
          ]
        }
      ]
    }
  },
  "documentMetadata": {}
}
```

4. Tokenizers

The tokenizer is a language-specific processor that evaluates documents and identifies the tokens. RBL supports tokenization and sentence boundaries for all languages. For many languages, you can choose the tokenizer by setting `tokenizerType`.

Tokenizer Types

TokenizerType	Description	Supported Languages
`ICU`	Uses the ICU tokenizer	All, except for Chinese and Japanese
`FST`	Uses the FST tokenizer	Czech, Dutch, English, French, German, Greek, Hungarian, Italian, Polish, Portuguese, Romanian, Russian, Spanish
`SPACELESS_LEXICAL`	Uses a lexicon and rules to tokenize input without spaces. Uses the Chinese Language Analyzer (CLA) or Japanese Language Analyzer (JLA).	Chinese, Japanese
`SPACELESS_STATISTICAL`	Uses statistical approach to tokenize input without spaces.	Chinese, Japanese, Korean, Thai
`DEFAULT`	Selects the default tokenizer for each language. The default is `SPACELESS_STATISTICAL` for Chinese, Japanese, and Thai, and `ICU` for all other languages.	All

NOTE
When creating Tokenizers and Analyzers, the `tokenizerType` must be the same for both.

Set `TokenizerOption#tokenizerType` and `AnalyzerOption#tokenizerType` to the same value.

For most languages the default tokenizer is referred to as the ICU tokenizer. It implements standard Unicode guidelines for determining boundaries between sentences and for breaking each sentence into individual tokens. Many languages have an alternate tokenizer, the FST tokenizer, enabled by setting the `tokenizerType` to `FST`. The FST tokenizer provides somewhat different sentence and token boundaries. For example, the FST tokenizer keeps hyphenated tokens together, while the ICU tokenizer breaks them into separate tokens. For applications that don't want tokens or lemmas that contains spaces, the ICU tokenizer provides the best accuracy. To determine which tokenizer is best for your use case, we recommend running each of them against a test dataset and reviewing the output.

For Chinese, Japanese, and Thai, the default tokenizer determines sentence boundaries, and then uses statistical models to segment each sentence into individual tokens. If Latin-script or other non-Chinese, non-Japanese, or non-Thai fragments greater than a certain length (defined by `minNonPrimaryScriptRegionLength`) are embedded in the Chinese, Japanese, or Thai text, then the tokenizer applies default Unicode tokenization to those fragments. If a non-primary script region is less than this length, and adjacent to a primary script region, it is appended to the primary script region.

To use the Chinese Language Analyzer (CLA) or Japanese Language Analyzer (JLA) [27] tokenization algorithm, set the `tokenizerType` to `SPACELESS_LEXICAL`. This disables post-tokenization analysis; an analyzer created with this option will leave its input tokens unchanged.

For all languages, the RBL tokenizer can apply Normalization Form KC (NFKC) as specified in Unicode Standard Annex #15 to normalize the tokens. This normalization includes a normalizing a fullwidth numeral to a halfwidth numeral, a fullwidth Latin letter to a halfwidth Latin letter, and a halfwidth Katakana character to a fullwidth Katakana character. NFKC normalization is turned off by default. Use the `nfkcNormalize` option to turn it on and use `tokenizerType` of `ICU`. To apply NKFC for Chinese and Japanese, `tokenizerType` must be `SPACELESS_STATISTICAL` or `DEFAULT`.

General Tokenizer Options

Option	Description	Type (Default)	Supported Languages
`caseSensitive`	Indicates whether tokenizers produced by the factory are case sensitive. If false, they ignore case distinctions.	Boolean (true)	Czech, Danish, Dutch, English, French, German, Greek, Hebrew, Hungarian, Indonesian, Italian, Malay (Standard), Norwegian, Polish, Portuguese, Russian, Spanish, Swedish, Tagalog
`defaultTokenizationLanguage`	Specify language to use for script regions, other than the script of the overall language.	Language code (`xxx`)	Chinese, Japanese, Thai
`minNonPrimaryScriptRegionLength`	Minimum length of sequential characters that are not in the primary script. If a non-primary script region is less than this length and adjacent to a primary script region, it is appended to the primary script region.	Integer (10)	Chinese, Japanese, Thai
`nfkcNormalize`	Turns on Unicode NFKC normalization before tokenization. `tokenizerType` must not be `FST` or `SPACELESS_LEXICAL`.	Boolean (false)	All
`query`	Indicates the input will be queries, likely incomplete sentences. If true, tokenizers may change their behavior.	Boolean (false)	All
`tokenizeForScript`	Indicates whether to use a different word-breaker for each script. If false, uses script-specific breaker for primary script and default breaker for other scripts.	Boolean (false)	Chinese, Japanese, Thai

Option	Description	Type (Default)	Supported Languages
`tokenizerType`	Selects the tokenizer to use	`TokenizerType` (`SPACELESS_STATISTICAL` for Chinese, Japanese, Thai; `ICU` for all other languages)	All

Enum Classes:

- `BaseLinguisticsOption`
- `TokenizerOption`

4.1. Structured Text

A document may contain tables and lists in addition to regular sentences. Structured text is composed of fragments, such as list items, table cells, and short lines of text. The tokenizer emits sentence offsets for each fragment it encounters.

One way fragments are identified is by detecting fragment delimiters. A delimiter is restricted to one character; the default delimiters are U+0009 (tab), U+000B (vertical tab), and U+000C (form feed). To modify the set of recognized delimiters, pass a string containing all desired delimiter values to the `fragmentBoundaryDelimiters` option. The string must include any default values you want to keep. Non-whitespace delimiters within a token will be ignored.

The following rules determine where fragments are identified, in descending priority:

- Each line in a list, where a list is defined as 3 or more lines containing the same punctuation mark within the first 5 characters of the line, are fragments
- A delimiter or three or more consecutive whitespace characters breaks a line into fragments
- A short line is a fragment if it is preceded by another short line, preceded by a fragment, or if it's the first line of text. The length of a short line is configurable with the `maxTokensForShortLine` option; the default is 6 or fewer tokens.

Fragments always include trailing whitespace.

Example:

```
BaseLinguisticsFactory factory = new BaseLinguisticsFactory();
factory.setOption(BaseLinguisticsOption.rootDirectory, rootDirectory)
factory.setOption(BaseLinguisticsOption.language, "eng");
EnumMap<BaseLinguisticsOption, String> options = Maps.newEnumMap(BaseLinguisticsOption.class);
options.put(BaseLinguisticsOption.fragmentBoundaryDelimiters, "|~");
options.put(BaseLinguisticsOption.maxTokensForShortLine, "5");
factory.createSingleLanguageAnnotator(options);
```

By default, fragment detection is enabled. Use the `fragmentBoundaryDetection` option to disable it.

Structured Text Options

Option	Description	Type (Default)	Supported Languages
`fragmentBoundaryDetection`	Turn on fragment boundary detection.	Boolean (true)	All
`fragmentBoundaryDelimiters`	Specify the fragment boundary delimiters.	String ("\u0009\u000B\u000C")	All
`maxTokensForShortLine`	The maximum length of a short line.	Integer (6)	All

Enum Classes:

- `BaseLinguisticsOption`
- `TokenizerOption`

4.2. Social Media Tokens: Emoji & Emoticons, Hashtags, @Mentions, Email Addresses, URLs

RBL supports POS-tagging of emoji, emoticons, @mentions, email addresses, hashtags, and URLs in all supported languages.

Tokenization of emoji is always enabled. The other options are disabled by default but can be enabled through the options listed. When tokenization is disabled, the characters may be split into multiple tokens.

Social Media Token Options

Option	Description	Default	Supported Languages
n/a[a]	Enables emoji tokenization	`true`	All
`emoticons`	Enables emoticon tokenization	`false`	All
`atMentions`	Enables atMention tokenziation	`false`	All
`hashtags`	Enables hashtag tokenization	`false`	All
`emailAddresses`	Enables emailAdress tokenization	`false`	All
`urls`	Enables url tokenization	`false`	All

[a]Emoji tokenization and POS-tagging is always enabled and cannot be disabled.

Enum Classes:

- `BaseLinguisticsOption`
- `TokenizerOption`

4.2.1. Sample input and output

Type	Input	Tokenization (when option is enabled)	Tokenization (when option is disabled)	POS tag
emoji	😀	😀	Tokenization of emoji cannot be disabled.	EMO
emoticon	:)	:)	:)	EMO
@mention	@basistechnology	@basistechnology	@ basistechnology	ATMENTION

Type	Input	Tokenization (when option is enabled)	Tokenization (when option is disabled)	POS tag
hashtag	#basistechnology	#basistechnology	# basistechnology	HASHTAG
email address	info@basistech.com	info@basistech.com	info @ basistech.com	EMAIL
URL	http:// www.basistech.com	http://www.basistech.com	http : / / www.basistech.com	URL

The tokenization when the option is disabled depend on the options `language` and `tokenizerType`. The samples provided here are for when `language=eng` and `tokenizerType=ICU`.

4.2.2. Emoji & Emoticon Recognition

Emoji are defined by Unicode Technical Standard #51. In tokenizing emoji, RBL recognizes the emoji presentation selector (VS16; U+FE0F) and text presentation selector (VS15; U+FE0E), which indicate if the preceding character should be treated as emoji or text.

Although RBL detects sideways, Western-style emoticons, it does not currently support Japanese-style emoticons called *kaomoji* such as `(o^ ^o)`.

4.2.3. Emoji Normalization & Lemmatization

RBL normalizes emoji, placing the result into the `lemma` field. The simplest example is when an emoji presentation selector follows a character that is already an emoji. In this case, RBL will simply remove the emoji presentation selector.

Lemmatization applies to an emoji character in multiple ways.

Emoji that depict people or body parts may be followed by an emoji modifier indicating skin tone. Lemmatization simply removes the emoji modifier skin tone from the emoji character. The reasoning is that the skin tone is of secondary importance to the meaning of the emoji.

Surface form	Lemmatized form
 (Boy + Medium Skin Tone)	

Emoji depicting people may be followed by an emoji component indicating hair color or style. Lemmatizing removes the hair component from the emoji character.

Surface form	Lemmatized form
 (Adult + ZWJ + Red Hair)	

Where a gender symbol has been added to create a gendered occupation emoji, lemmatization removes the gender symbol.

Surface form	Lemmatized form
 (Police Officer + ZWJ + ♀ Female Sign + VS16)	

Finally, RBL can normalize non-fully-qualified emoji ZWJ sequences to fully-qualified emoji ZWJ sequences. In the above example, it is possible to omit the VS16 (though discouraged by Unicode): since Police Officer is an emoji, anything joined to it by a ZWJ is implicitly an emoji too. RBL adds the missing VS16.

4.3. Customizing the ICU Tokenizer

The ICU tokenizer is the default tokenizer used for European languages. It works based on behavior defined in a rule file. If the default behavior is not exactly what is desired, RBL allows custom rule files to be supplied that will determine the behavior of the tokenizer. How to make these customizations is briefly outlined here. Be careful with any changes you make to the tokenizer behavior; Basis does not support customizations made by the user.

`BaseLinguisticsFactory` and `TokenizerFactory` both have a method `addCustomTokenizerRules` which can be used to specify a custom rule file. RBLCmd also has the `-ctr` option to specify a path on the command line. All of these methods accept a case-sensitivity value (for `-ctr`, `cs` and `ci` mean case-sensitive and case-insensitive), which is important because only when `BaseLinguisticsOption.caseSensitive` is the same as the value for a rule file will it be selected. Custom rule files are not cumulative, i.e. only one set of rules may be used at a time for any one combination of case sensitivity and language.

NOTE
Basis reserves the right to change the version of ICU used in RBL. Thus any rule file provided by Basis for a particular version of RBL may or may not work with newer versions.

4.3.1. Tokenization Rule File Format

A tokenization rule file is a ICU break rule file encoded in UTF-8. A custom file replaces Basis's tokenization rules, so a custom rule should include all the rules for basic tokenization as well as the new custom rules. The default rule files that RBL uses can be obtained by contacting Basis support, or you can copy the rule file from ICU.

RBL also provides the ability to pass in a subrule file if desired. This is for splitting tokens produced according to rules in the main file. The subrule file is a list of subrules, each of which is a number and a regex separated by a tab character. This number corresponds to the "rule status"[3] of the main rule whose tokens the subrule splits. Each capturing group in the subrule regex corresponds to a token that will be produced by the tokenizer.

The rule file and the subrule file can be placed anywhere. In particular, they need not be placed anywhere within your RBL installation directory.

There is one Basis-specific extension, `!!btinclude <filename>`. This command tells the preprocessor to replace the `!!btinclude` line with the contents of the specified file. Relative paths are relative to the location of the file containing the `!!btinclude` line. Recursive inclusion is allowed.

4.3.2. Example

The ICU tokenizer does not normally tokenize with an eye to emoticons, but perhaps that is important to your use case. You could make a copy of the default rule file and add the following.

[3]Essentially an ID number; see the ICU break rule documentation

```
...
$Smiley = [\:=][)}\]];

!!forward;
$Smiley;
...
```

For the input:

```
=)
```

instead of the output:

```
=
)
```

of two tokens with the Basis default rules you would get back one token:

```
=)
```

4.4. Unknown Language Tokenization

RBL provides basic tokenization support when the language is "Unknown" (`xxx`). The tokenizer uses generic rules to tokenize, such as whitespace and punctuation delimitation.

Supported Features when language is unknown (`xxx`):

- Tokenization
- Sentence breaking
- Identification of some common acronyms and abbreviations
- Segmentation user dictionaries

Using the language code of `xxx` will provide basic tokenization support for languages not supported by RBL.

5. Analyzers

The analyzer is a language-specific processor that uses dictionaries and statistical analysis to add analysis objects to tokens.

To extend the coverage that RBL provides for each supported language you can create User Dictionaries [34]. Segmentation user dictionaries are supported for all languages. Lemma user dictionaries are supported for Chinese, Czech, Danish, Dutch, English, French, German, Greek, Hebrew, Hungarian, Italian, Japanese, Korean, Norwegian, Polish, Portuguese, Russian, Spanish, Swedish, and Thai.

A stem is the substring of a word that remains after prefixes and suffixes are removed, while the lemma is the dictionary form of a word. RBL supports stems [30] for Arabic, Finnish, Persian, and Urdu.

Semitic roots are generated for Arabic and Hebrew [29].

The option name to set the analysis cache depends on the accepting factory. The option `analysisCacheSize` is a `BaseLinguisticsOption` while `cacheSize` is an option for both `AnalyzerOption` and `CSCAnalyzerOption`. They all perform the same function.

General Analyzer Options

Option	Description	Type (Default)	Supported Languages
`analysisCacheSize` `cacheSize`	Maximum number of entries in the analysis cache. Larger values increase throughput, but use extra memory. If zero, caching is off.	Integer (100.000)	All
`caseSensitive`	Indicates whether analyzers produced by the factory are case sensitive. If false, they ignore case distinctions.	Boolean (true)	Czech, Danish, Dutch, English, French, German, Greek, Hebrew, Hungarian, Indonesian, Italian, Malay (Standard), Norwegian, Polish, Portuguese, Russian, Spanish, Swedish, Tagalog
`deliverExtendedTags`	Indicates whether the analyzers should return extended tags with the raw analysis. If true, the extended tags are returned.	Boolean (false)	All
`normalizationDictionaryPaths`	A list of paths to user many-to-one normalization dictionaries, separated by semicolons or the OS-specific path separator.	List of paths	All
`query`	Indicates the input will be queries, likely incomplete sentences. If true, analyzers may change their behavior (e.g. disable disambiguation)	Boolean (false)	All
`tokenizerType`	Selects the tokenizer to use with this analyzer.	`TokenizerType` (`SPACELESS_STATISTICAL` for Chinese, Japanese, Thai; `ICU` for all other languages)	All

NOTE

When creating Tokenizers and Analyzers, the `tokenizerType` must be the same for both.

Set `TokenizerOption#tokenizerType` and `AnalyzerOption#tokenizerType` to the same value.

Enum Classes:

- `AnalyzerOption`
- `BaseLinguisticsOption`
- `CSCAnalyzerOption` (`cacheSize` only)

5.1. Lemma Lookup

For each token and normalized form in the token stream, the analyzer performs a dictionary lookup starting with any user dictionaries followed by the RBL dictionary. During lookup, RBL ignores the context in which the token or normalized form appears.

Once the analyzer has found one or more lemmas in a dictionary, it does not consult additional dictionaries. In other words, if two user dictionaries are specified, and the filter finds a lemma in the first dictionary, it does not consult the second user dictionary or the RBL dictionary.

Unless overridden by an analysis dictionary, the only lemmatization done in Chinese and Thai is number normalization. Other Chinese and Thai tokens' lemmas are equal to their surface forms.

There is no analysis dictionary available for Finnish, Pashto, or Urdu. All other languages are supported.

5.2. Guessing

No dictionary can ever be complete: new words get added to languages, languages change and borrow. So, in general, analysis for each language includes some sort of guessing capability. The job of a guesser is to take a word and to come up with some analysis of it. Whatever facts we generate for a language, those are all possible outputs of a guesser.

In European languages, guessers deliver lemmas and parts of speech. In Korean, guessers provide morphemes, morpheme tags, compound components, and parts of speech.

5.3. Whitespace in Lemmas

By default, the analyzer returns any lemma that contains whitespace as multiple lemmas (each with no whitespace). To allow lemmas with whitespace (such as `International Business Machines` as a lemma for the token `IBM`) to be placed as such in the token stream, you can create a user analysis dictionary [34] with an entry that defines the lemma. For example:

```
IBM    International[^_]Business[^_]Machines[+PROP]
```

5.4. Compounds

The analyzer decomposes Chinese, Danish, Dutch, German, Hungarian, Japanese, Korean, Norwegian, and Swedish compounds, returning the lemmas of each of the components.

The lemmas may differ from their surface form in the compound, such that the concatenation of the components is not the same as the original compound (or its lemma). Components are often connected by elements that are present only in the compound form.

For example, the German compound *Eingangstüren* (entry doors) is made up of two components, *Eingang* (entry) and *Tür* (door), and the connecting 's' is not present in the component list. For this input token, the RBL tokenizer and analyzer return the following entries:

- Original form: *Eingangstüren*

- Lemma for the compound: *Eingangstür*
- Component lemmas: *Eingang*, *Tür*

Other German examples include letter removal (*Rennrad* ⇒ *rennen* + *Rad*), vowel changes (*Mängelliste* ⇒ *Mangel* + *Liste*), and capitalization changes (*Blaugrünalge* ⇒ *blau* + *grün* + *Alge*).

Compound Options

Option	Description	Type (Default)	Supported Languages
`decomposeCompounds`	Indicates whether to decompose compounds. For Chinese and Japanese, `tokenizerType` must be `SPACELESS_LEXICAL`. If `koreanDecompounding` is enabled but `decomposeCompounds` is disabled, compounds will be decomposed.	Boolean (true)	Chinese, Danish, Dutch, German, Hungarian, Japanese, Korean, Norwegian (Bokmål, Nynorsk), Swedish
`compoundComponentSurfaceForms`	Indicates whether to return the surface forms of compound components. When this option is enabled and ADM results are returned, `getText` returns the surface form of a component `Token`, and its lemma can be retrieved using `Token#getAnalyses()` and `MorphoAnalysis#getLemma()`. When this option is enabled and the results are not in ADM format, `getCompoundComponentSurfaceForms` returns the surface forms of a compound word's `Analysis`, and its surface form is not available. This option has no effect when `decomposeCompounds` is set to `false`.	Boolean (false)	Dutch, German, Hungarian

Enum Classes:

- `AnalyzerOption`
- `BaseLinguisticsOption`

5.5. Disambiguation

For some languages, the analyzer can disambiguate between multiple analysis objects and return the disambiguated analysis object. The `disambiguate` option enables the disambiguator. When `true`, the disambiguator determines the best analysis for each word given the context in which it appears.

When using an annotator, the disambiguated result is at the head of all possible analyses. The remainder of the list is ordered randomly. When using a tokenizer/analyzer, use the method `getSelectedAnalysis` to return the disambiguated result.

For all languges except Japanese, disambiguation is enabled by default. For performance reasons, disambiguation is disabled by default for Japanese when using the statistical model.

Disambiguation Options

Option	Description	Type (Default)	Supported Languages
`disambiguate`	Indicates whether the analyzers should disambiguate the results.	Boolean (true)	Arabic, Chinese, Czech, Dutch, English, French, German, Greek, Hebrew, Hungarian, Italian, Japanese, Korean, Polish, Portuguese, Russian, Spanish
`alternativeEnglishDisambiguation`	Enables faster part of speech disambiguation for English.	Boolean (false)	English
`alternativeGreekDisambiguation`	Enables faster part of speech disambiguation for Greek	Boolean (false)	Greek
`alternativeSpanishDisambiguation`	Enables faster part of speech disambiguation for Spanish.	Boolean (false)	Spanish

Enum Classes:

- `AnalyzerOption`
- `BaseLinguisticsOption`

5.6. Part-of-Speech (POS) Tags

In RBL, each language has its own set of POS tags and a few languages have multiple tag sets. Each tag set is identified by an identifier, which is a value of the `TagSet` enum. When RBL outputs a POS tag, it also lists the identifier for the tag set it came from. Output from a single language may contain POS tags from multiple tag sets, including the language-neutral set.

POS tags are defined for Arabic, Chinese, Czech, Dutch, English, French, German, Greek, Hebrew, Hungarian, Italian, Japanese, Korean, Persian, Polish, Portuguese, Russian, Spanish, and Urdu.

5.6.1. Returning Universal Part-of-Speech (POS) Tags

The `universalPosTags` option converts Basis POS tags to universal POS tags, as defined by the Universal Dependencies project. The POS tag mappings are defined by POS tag map files. By default, the annotator uses the map in *`rootDirectory`*`/upt-16/upt-16-<language>.yaml`, where <language> is a language code [57]. `customPosTagsUri` allows you to specify custom POS tag mappings.

If you want to return universal part-of-speech tags in place of the language-specific tags that RBL ordinarily returns, set `universalPosTags` to `true`.

For an ADM sample that follows the same pattern as the preceding sample and returns universal POS tags for each token, see `rbl-je-<version>/samples/universal-pos-tags`.

Universal POS Tag Options

Option	Description	Type (Default)	Supported Languages
`universalPosTags`	Indicates if POS tags should be converted to unversal versions	Boolean (false)	POS tags are defined for Arabic, Chinese, Czech, Dutch, English, French, German, Greek, Hebrew, Hungarian, Italian, Japanese, Korean, Persian, Polish, Portuguese, Russian, Spanish, and Urdu.

Option	Description	Type (Default)	Supported Languages
`customPosTagsUri`	URI of a POS tag map	URI	POS tags are defined for Arabic, Chinese, Czech, Dutch, English, French, German, Greek, Hebrew, Hungarian, Italian, Japanese, Korean, Persian, Polish, Portuguese, Russian, Spanish, and Urdu.

Enum Classes:

- `BaseLinguisticsOption`

5.6.2. POS Tag Map File Format

A POS tag map file is a YAML file encoded in UTF-8. It is a sequence of mapping rules.

A mapping rule is a sequence of two elements: the POS tag to be mapped and a sequence of submappings. Rules are checked in the order they appear in the rule file. A token which matches a rule is not checked against any further rules.

A submapping is a mapping with the keys `m`, `s`, and `t`. `m` is a Java regular expression. `s` is a surface form. `m` and `s` are optional: they can be omitted or null. `t` specifies the output POS tag to use when the following criteria are met:

- The input token's POS tag equals the POS tag to be mapped.
- `m` (if any) matches a substring of the input token's raw analysis.
- `s` (if any) equals the input token's surface form, compared case-insensitively.

5.6.3. Example

```
-
  - NUM_VOC
  -
    - { m: \+Total, t: PRON }
    - { s: moc, t: DET }
    - { s: oba, t: DET }
    - { t: NUM }
```

This rule maps tokens with Basis's NUM_VOC POS tag. If the input token's raw analysis matches the regular expression `\+Total`, the token becomes a PRON. Otherwise, if the token's surface form is *moc* or *oba*, the token becomes a DET. Otherwise, the token becomes a NUM.

5.7. Splitting Contractions

You can split contractions and return analyses with tokens, lemmas, and POS tags for each constituent. For example, given the English contraction *can't*, RBL returns analyses for `can` and for `not`. To split contractions, set `tokenizeContractions` to `true`.

Contractions are defined by contraction rule files. By default, the tokenizer uses the rules in *`rootDirectory`*`/contractions/contraction-rules-<language>.yaml`, where <language> is the language code. RBL comes with contraction rules for English, German, and Portuguese. To add rules for these languages or add rules to support another language, edit the default files or create a custom rule file. The URI for the custom file is defined by `customTokenizeContractionRulesUri`.

Contraction Splitting Options

Option	Description	Type (Default)	Supported Languages
`tokenizeContractions`	Indicates whether to deliver contractions as multiple tokens. If `false`, they are delivered a a single token.	Boolean (false)	All
`customTokenizeContractionRulesUri`	URI of contraction rule file.	URI	All

Enum Classes:

- BaseLinguisticsOption

For a sample, see `rbl-je-<version>/samples/contractions`.

5.7.1. Contraction Splitting Rule File Format

A contraction rule file is a YAML file encoded in UTF-8. It must be a sequence of contraction rules.

A contraction rule is a sequence of two elements: a contraction key and a contraction replacement. Any token which matches the key is replaced with the replacement. Rules are checked in the order they appear in the rule file. A token which matches a rule is not checked against any further rules. A token which matches no rule is not rewritten.

A contraction key is a sequence of a surface form and a POS tag. A token matches a key if and only if its surface form and POS tag match the key's surface form and POS tag.

- A surface form is a string. Surface forms are compared case-insensitively.
- A POS tag is a string. POS tags are compared case-sensitively.

A contraction replacement is a sequence of replacement tokens.

- A replacement token is a sequence of a replacement surface form, POS tag, lemma, and raw analysis. All four are strings. The raw analysis can also be null.

5.7.2. Example

```
-
  - [ "ain't", "VBPRES" ]
  -
    - [ "am", "VBPRES", "be", null ]
    - [ "not", "NOT", "not", null ]
-
  - [ "amn't", "ADJ" ]
  -
    - [ "am", "VBPRES", "be", null ]
    - [ "not", "NOT", "not", null ]
-
  - [ "amn't", "NOUN" ]
  -
    - [ "am", "VBPRES", "be", null ]
    - [ "not", "NOT", "not", null ]
```

The first entry is for *ain't* with POS tag VBPRES. This splits into *am* and *not*. The next is for *amn't* as an ADJ, and the third is for *amn't* as a NOUN.

The replacement surface form uses the same capitalization format as the original surface form. Using the first entry of the above example, *ain't* becomes *am not*, *Ain't* becomes *Am not*, and *AIN'T* becomes *AM NOT*.

6. Chinese and Japanese Lexical Tokenization

For Chinese and Japanese, in addition to the statistical model described above, RBL includes Chinese Language Analyzer (CLA) and Japanese Language Analyzer (JLA) [41] modules [4] which are optimized for search. They are activated by setting `tokenizerType` to `SPACELESS_LEXICAL`.

Chinese and Japanese Lexical Options

Option	Description	Default value	Supported languages
`breakAtAlphaNumIntraWordPunct`	Indicates whether to consider punctuation between alphanumeric characters as a break. Has no effect when `consistentLatinSegmentation` is `true`.	`false`	Chinese
`consistentLatSegmentation`	Indicates whether to provide consistent segmentation of embedded text not in the primary script. If false, then the setting of `segmentNonJapanese` is ignored.	`true`	Chinese, Japanese
`decomposeCompounds`	Indicates whether to decompose compounds.	`true`	Chinese, Japanese
`deepCompoundDecomposition`	Indicates whether to recursively decompose each token into smaller tokens, if the token is marked in the dictionary as being decomposable. If deep decompounding is enabled, the decomposable tokens will be further decomposed into additional tokens. Has no effect when `decomposeCompounds` is `false`.	`false`	Chinese, Japanese
`favorUserDictionary`	Indicates whether to favor words in the user dictionary during segmentation.	`false`	Chinese, Japanese
`ignoreSeparators`	Indicates whether to ignore whitespace separators when segmenting input text. If `false`, whitespace separators will be treated as morpheme delimiters. Has no effect when `whitespaceTokenization` is `true`.	`true`	Japanese
`ignoreStopwords`	Indicates whether to filter stop words [29] out of the output.	`false`	Chinese, Japanese
`minLengthForScriptChange`	Sets the minimum length of non-native text to be considered for a script change. A script change indicates a boundary between tokens, so the length may influence how a mixed-script string is tokenized. Has no effect when `consistentLatinSegmentation` is `false`.	10	Chinese, Japanese
`pos`	Indicates whether to add parts of speech to morphological analyses.	`true`	Chinese, Japanese
`segmentNonJapanese`	Indicates whether to segment each run of numbers or Latin letters into its own token, without splitting on medial number/word joiners. Has no effect when `consistentLatinSegmentation` is `true`.	`true`	Japanese
`separateNumbersFromCounters`	Indicates whether to return numbers and counters as separate tokens.	`true`	Japanese
`separatePlaceNameFromSuffix`	Indicates whether to segment place names from their suffixes.	`true`	Japanese
`whiteSpaceIsNumberSep`	Indicates whether to treat whitespace as a number separator. Has no effect when `consistentLatinSegmentation` is `true`.	`true`	Chinese

[4] These analyzers are compatible with the Chinese and Japanese language processors found in the legacy Rosette (C++) products.

Option	Description	Default value	Supported languages
`whitespaceTokenization`	Indicates whether to treat whitespace as a morpheme delimiter.	`false`	Chinese, Japanese

Enum Classes:

- `BaseLinguisticsOption`
- `TokenizerOption`

6.1. Chinese and Japanese Readings

Chinese and Japanese Readings

Option	Description	Default value	Supported languages
`generateAll`	Indicates whether to return all the readings for a token. For characters with multiple readings, all the readings are returned in brackets and separated by semicolons. Has no effect when `readings` is `false`.	`false`	Chinese
`readingByCharacter`	Indicates whether to skip directly to the fallback behavior of `readings` without considering readings for whole words. Has no effect when `readings` is `false`.	`false`	Chinese, Japanese
`readings`	Indicates whether to add readings to morphological analyses. The annotator will try to add readings by whole words. If it cannot, it will concatenate the readings of individual characters.	`false`	Chinese, Japanese
`readingsSeparateSyllables`	Indicates whether to add a separator character between readings when concatenating readings by character. Has no effect when `readings` is `false`.	`false`	Chinese, Japanese
`readingType`	Sets the representation of Chinese readings. Possible values (case-insensitive) are: • `cjktex`: macros for the CJKTeX `pinyin.sty` style • `no_tones`: pinyin without tones • `tone_marks`: pinyin with diacritics over the appropriate vowels • `tone_numbers`: pinyin with a number from 1 to 4 suffixed to each syllable, or no number for neutral tone	`tone_marks`	Chinese
`useVForUDiaeresis`	Indicates whether to use 'v' instead of 'ü' in pinyin readings, a common substitution in environments that lack diacritics. The value is ignored when `readingType` is `cjktex` or `tone_marks`, which always use 'v' and 'ü' respectively. It is probably most useful when `readingType` is `tone_numbers`. Has no effect when `readings` is `false`.	`false`	Chinese

Enum Classes:

- `BaseLinguisticsOption`
- `TokenizerOption`

6.2. Editing the stop words list

The `ignoreStopwords` option uses a stop words list to define stop words. The path to the stop words list is language-dependent: Chinese uses `root/dicts/zho/cla/zh_stop.utf8` and Japanese uses `root/dicts/jpn/jla/JP_stop.utf8`.

You can add stop words to these files. When you edit one of these files, you must follow these rules:

- The file must be encoded in UTF-8.
- The file may include blank lines.
- Comment lines begin with `#`.
- Each non-blank non-comment line represents exactly one lexeme (stop word).

7. Japanese Lemma Normalization

In Japanese, foreign and borrowed words may vary in their phonetic transcription to Katakana, and some words may be expressed with an older or a modern Kanji form. The Japanese lemma dictionary maps Katakana variants to a standard form and old Kanji forms to their modern forms. Examples:

Katakana Spelling Variants	Normalized Form
ヴァイオリン	バイオリン
エクスポ	エキスポ

Older Kanji Form	Normalized Form
渡邊	渡辺
松濤	松涛
大學	大学

You can include orthographic normalization in lemma user dictionaries for Japanese. This information can be accessed at runtime from the `Analysis` or `MorphoAnalysis` object.

8. Hebrew Analyses

The following analyzer options are available for Hebrew.

Hebrew Options

Option	Description	Type (Default)
`guessHebrewPrefixes`	Splits prefixes off unknown Hebrew words	Boolean (false)
`includeHebrewRoots`	Indicates whether to generate Semitic root forms	Boolean (false)

Enum Classes:

- `AnalyzerOption`
- `BaseLinguisticsOption`

8.1. Hebrew Disambiguator Types

RBL includes multiple disambiguators for Hebrew. Set the value for the option `disambiguatorType` to select which type to use. The valid values for `DisambiguatorType` are:

- `PERCEPTRON`: a perceptron model
- `DICTIONARY`: a dictionary-based reranker
- `DNN`: a deep neural network.
  TensorFlow [6], which is not supported on all systems, much be installed. If `DNN` is selected and TensorFlow is not supported, RBL will throw a `RosetteRuntimeException`.

Hebrew Disambiguation Options

Option	Description	Type (Default)	Supported Languages
`disambiguatorType`	Selects which disambiguator to use for Hebrew.	DisambiguatorType (`PERCEPTRON`)	Hebrew

Enum Classes:

- `AnalyzerOption`
- `BaseLinguisticsOption`

9. Arabic, Persian, and Urdu Token Analysis

For Arabic, Persian (Western Persian and Afghan Persian), and Urdu, RBL may return multiple analyses for each token. Each analysis contains the normalized form of the token, a part-of-speech tag, and a stem. For Arabic, the analysis also includes a lemma and a Semitic root. For Persian, some analyses include a lemma.

This appendix provides information on token normalization and the generation of variant tokens. For Arabic, it also provides information on stems and Semitic roots.

Token normalization is performed in two stages:

1. Generic Arabic script normalization
2. Language-specific normalization

9.1. Generic Arabic Script Token Normalization

Generic Arabic script normalization includes the following:

- The following diacritics are removed: *dammatan*, *kasratan*, *fatha*, *damma*, *kasra*, *shadda*, *sukun*.
- The following characters are removed: kashida, *left-to-right marker*, *right-to-left marker*, *zero-width joiner*, *BOM*, *non-breaking space*, *soft hyphen*, *space*.
- *Alef maksura* is converted to *yeh* unless it is at the end of the word or followed by *hamza*.
- All numbers are converted to Arabic numbers: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
  [5]Thousand separators are removed, and the decimal separator is changed to a period (U+002E). The normalizer handles cases where ر (*reh*) is (incorrectly) used as the decimal separator.
- *Alef with hamza above*: ا ٔ (U+0675), أ (U+0672), or ا (U+0627) combined with *hamza above* (U+0654) is converted to أ (U+0623).
- *Alef with madda above*: ا (U+0627) combined with *madda above* (U+0653) is converted to آ (U+0622).
- *Alef with hamza below*: إ (U+0673) or ا (U+0627) combined with *hamza below* (U+0655) is converted to إ (U+0625).
- *Misra sign to ain*: (U+060F) is converted to ع (U+0639).
- *Swash kaf to kaf*: ک (U+06AA) is converted to ک (U+06A9).
- *Heh*: ہ (U+06D5) is converted to ه (U+0647).
- *Yeh with hamza above*: The following combinations are converted to ئ (U+0626).
  ی (U+06CC) combined with *hamza above* (U+0654)
  ى (U+0649) combined with *hamza above* (U+0654)
  ي (U+064A) combined with *hamza above* (U+0654)
- *Waw with hamza above*: و (U+0648) combined with *hamza above* (U+0654), ۥ (U+0677), or ۄ (U+0676) is converted to ؤ (U+0624).

9.2. Arabic Token Analysis

9.2.1. Token Normalization

For Arabic input, 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.
- Persian *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).
- Persian *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:

[5]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 Persian and Urdu Arabic script (۰, ۱, ۲, ۳, ۴, ۵, ۶, ۷, ۸, ۹).

- *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).

Token Variants

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 token variant is generated in 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*.

Stems and Semitic Roots

The stem returned is the normalized token with affixes (such as prepositions, conjunctions, the definite article, proclitic pronouns, and inflectional prefixes) removed.

In the process of stripping morphemes (affixes) from a token, the analyzer produces a stem, a lemma, and a Semitic root. Stems and lemmas result from stripping most of the inflectional morphemes, while Semitic 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 Semitic 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 Semitic root: كتب (ktb)

9.3. Persian Token Analysis

9.3.1. Persian Token Normalization

The following Persian-specific normalizations are performed on the output of the Arabic script normalization:

- *Fathatan* (U+064B) and *superscript alef* (U+0670) are removed.

- *Alef*أ (U+0623), إ (U+0625), or ٱ (U+0671) is converted to ا (U+0627).

- Arabic *kaf*ك (U+0643) is converted to Persian *kaf*ک (U+06A9).

- *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).
- Arabic *yeh*ي (U+064A) or ى (U+0649) is converted to Persian *yeh*ی (U+06CC).

Following morphological analysis:

- Zero-width non-joiner (U+200C) and superscript *alef* (U+0670) are removed.

9.3.2. Token Variants

The analyzer can generate a variant form for some tokens to account for the orthographic irregularity seen in contemporary written Persian. Each variation is generated with the normalized form.

- If a word contains *hamza on yeh* (U+0626), a variant is generated replacing the *hamza on yeh* with Persian *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).

9.3.3. Stems and Lemmas

The Persian analyzer produces both stems and lemmas. A stem is the substring of a word that remains after all prefixes and suffixes are removed. A lemma is the dictionary form of a word. The lemma may differ from the stem if a word is irregular, or if a word contains regular transformations. The distinction between stems and lemmas is especially important for Persian verbs. The typical verb inflection table for Persian includes a past stem and and a present stem that cannot be derived from each other.

Examples. The present subjunctive tense verb بگویم (*beguyam*, that I say) has the stem گوی (*guy*) . The past tense verb گفتم (*goftam*, I said) has the stem گفت (*goft*). These two have different stems, because the word-internal strings are different. They have the same lemma گفت (*goft*) because they are inflections of the same word.

9.4. Urdu Token Analysis

9.4.1. Token Normalization

The following Urdu-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* ۂ (U+06C0) is converted to ۀ (U+06C2).
- *Yeh*ي (U+064A) or ى (U+0649) is converted to ی (U+06CC).

9.4.2. Token Variants

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 with the normalized form.

- If a word contains *hamza on yeh* (U+0626), a variant is generated replacing the *hamza on yeh* with Persian *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 *heh doachashmee* (U+06BE), a variant is generated replacing the *heh doachashmee* 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).

10. User Dictionaries

User dictionaries are supplementary dictionaries that change the default linguistic analyses. These dictionaries can be static or dynamic.

- Static dictionaries are compiled ahead of time and passed to a factory.
- Dynamic dictionaries are created and configured at runtime. Dynamic dictionaries are held completely in memory and state is not saved on disk. When the JVM exits, or the factory becomes unused, the contents are lost.

In all dictionaries, the entries should be Form KC normalized. Japanese Katakana characters, for example, should be full width, and Latin characters, numerals, and punctuation should be half width. Analysis dictionaries can contain characters of any script, while for most consistent performance in Chinese, Japanese, and Thai, token dictionaries should only contain characters in the Hanzi (Kanji), Hiragana, Katakana, and Thai scripts.

In Chinese, Japanese, and Thai, text in foreign scripts (such as Latin script) in the input that equals or exceeds the length specified by `minNonPrimaryScriptRegionLength` (the default is 10) is passed to the standard Tokenizer and not seen by a user segmentation dictionary.

10.1. Types of User Dictionaries

- **Analysis Dictionary:** An analysis dictionary [38] allows users to modify the analysis or add new variations of a word. The analysis associated with a word includes the default lemma as well as part-of-speech tag and additional characteristics for some languages. Use of these dictionaries is not supported for Arabic, Persian, Romanian, Turkish, and Urdu.
- **Segmentation Dictionary:** A segmentation dictionary [38] allows users to specify strings that are to be segmented as tokens.
  Chinese and Japanese segmentation user dictionary entries may not contain the ideographic full stop.
- **Many-to-one Normalization Dictionaries:** Users can implement a many-to-one normalization dictionary [40] to map multiple spelling variants to a single normalized form.

- **CLA/JLA Dictionaries:** The Chinese Language Analyzer and Japanese Language Analyzer [41] both include the capability to create and use one or more segmentation (tokenization) user dictionaries for vocabulary specific to an industry or application. A common usage for both languages is to add new nouns like organizational and product names. These and existing nouns can have a compounding scheme specified if, for example, you wish to prevent an otherwise compound product name from being segmented as such. When the language is Japanese, you can also create user reading dictionaries with transcriptions rendered in Hiragana. The readings can override the readings returned from the JLA reading dictionary and override readings that are otherwise guessed from segmentation (tokenization) user dictionaries.
- **CSC Dictionary:** Users can specify conversion for use with the Chinese Script Converter (CSC).

10.2. Prioritization of User Dictionaries

All static and dynamic user dictionaries, except for many-to-one normalization dictionaries, are consulted in reverse order of addition. In cases of conflicting information, dictionaries added later take priority over those added earlier. Once a token is found in a user dictionary, RBL stops and will consult neither the remaining user dictionaries nor the RBL dictionary.

Many-to-one normalization dictionaries are consulted in the following order:

1. All dynamic user dictionaries, in reverse order of addition.
2. Static dictionaries in the order that they appear in the option list for `normalizationDictionaryPaths`.

Example of non-many-to-one user dictionary priority:

1. User adds dynamic dictionary named `dynDict1`
2. User adds static dictionary named `statDict2`
3. User adds static dictionary named `statDict3`
4. User adds dynamic dictionary named `dynDict4`

Dictionaries are prioritized in the following order:

`dynDict4`, `statDict3`, `statDict2`, `dynDict1`

Example of many-to-one normalization user dictionary priority:

1. User adds dynamic dictionary named `dynDict1`
2. User sets `normalizationDictionaryPaths = "statDict2;statDict3"`
3. User adds dynamic dictionary named `dynDict4`

Dictionaries are prioritized in the following order:

`dynDict4`, `dynDict1`, `statDict2`, `statDict3`

The Chinese and Japanese language analyzers load all dictionaries with the user dictionaries loaded at the end of the list. To prioritize the user dictionaries and put them at the front of the list, guaranteeing the matches in the user dictionaries will be used, set the option `favorUserDictionary` to `true`.

10.3. Preparing the Source

The following formatting rules apply to user dictionary source files.

- The source file is UTF-8 encoded.
- The file may begin with a byte order mark (BOM).
- Each entry is a single line.
- Empty lines are ignored.

Once complete, the source file is compiled into a binary format for use in RBL.

10.3.1. Dynamic User Dictionaries

A dynamic user dictionary allows users to add user dictionary values at runtime. Instead of creating and compiling the dictionary in advance, the values are added dynamically. Dynamic dictionaries are available for all types of user dictionaries, except the CSC dictionary.

The process for using dynamic dictionaries is the same for each dictionary type:

1. Create an empty dynamic dictionary for the dictionary type.
2. Use the appropriate add method to add entries to the dictionary.

Dynamic dictionaries use the same structure as the compiled user dictionaries, but instead of having a single tab-delimited string, they are composed of separate strings. As an example, let's look at a many-to-one normalization dictionary entry:

Static dictionary entry (values are separated by tabs):

```
norm1        var1        var2        var3
```

Dynamic dictionary entry:

```
dictionary.add("norm1", "var1", "var2", "var3");
```

CAUTION

Dynamic dictionaries are held completely in memory and state is not saved on disk. When the JVM exits, or the annotator, tokenizer, or analyzer becomes unused, the contents are lost.

10.3.2. Case

User dictionary lookups are case-sensitive. RBL provides an option, `caseSensitive`, to control whether the analysis phase is case-sensitive.

- If `caseSensitive` is `true`, (the default), then the token itself is used to query the dictionary.

- If `caseSensitive` is `false`, the token is lowercased before consulting the dictionary. If the analyses is intended to be case-insensitive then the words in the user dictionary must all be in lowercase.

If you are using the `BaseLinguisticsTokenFilterFactory`, then the value for `AnalyzerOption.caseSensitive` both turns on the corresponding analysis and associates the dictionary with that analysis.

For Danish, Norwegian, and Swedish, the provided dictionaries are lowercase and `caseSensitive` is automatically set to `false`.

10.3.3. Valid Characters for Chinese and Japanese User Dictionary Entries

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

0022..007E, 00A2, 00A3, 00A5, 00A6, 00A9, 00AC, 00AF, 00B7, 0370..04FF, 2010..206F, 2160..217B, 2190..2193, 2200..22FF, 2460..24FF, 2502, 25A0..26FF, 2985, 2986, 3001..3007, 300C, 300D, 3012, 3020, 3031..3037, 3041..3094, 3099..309E, 30A1..30FE, 3200..33FF, 4E00..9FFF, D800..FA2D, FF00, FF02..FFEF

10.4. Compiling a User Dictionary

In the `tools/bin` directory, RBL includes a shell script for Unix

```
rbl-build-user-dictionary
```

and a .bat file for Windows.

```
rbl-build-user-dictionary.bat
```

To compile a user dictionary, from the RBL root directory:

```
tools/bin/rbl-build-user-dictionary -type TYPE_ARGUMENT LANG INPUT_FILE OUTPUT_FILE
```

where TYPE_ARGUMENT is the dictionary type, LANG is the language code, INPUT_FILE is the pathname of the source file you have created, and OUTPUT_FILE is the pathname of the binary compiled dictionary the tool creates. For example:

```
tools/bin/rbl-build-user-dictionary -type analysis jpn jpn_lemmadict.txt jpn_lemmadict.bin
```

Type Arguments

Dictionary Type	TYPE_ARGUMENT
Analysis	`analysis`
Segmentation	`segmentation`
Many-to-one	`m1norm`
CLA or JLA segmentation	`cla` or `jla`
JLA reading	`jla-reading`

The script uses Java to compile the user dictionary. The operation is performed in memory, so you may require more than the default heap size. You can set heap size with the `JAVA_OPTS` environment variable. For example, to provide 8 GB of heap size, set `JAVA_OPTS` to `-Xmx8g`.

Unix shell:

```
export JAVA_OPTS=-Xmx8g
```

Windows command prompt:

```
set JAVA_OPTS=-Xmx8g
```

10.5. Segmentation Dictionaries

The format for a segmentation dictionary source file is very simple. Each word is written on its own line, and that word is guaranteed to be segmented as a single token when seen in the input text, regardless of context. Japanese example:

```
三菱 UFJ 銀行
酸素ボンベ
```

User Segmentation Dictionary API

Class	Method	Task
`BaseLinguisticsFactory`	`addUserSegDictionary`	Adds a user segmentation dictionary for a given language.
	`addDynamicSegDictionary`	Create and load new dynamic segmentation dictionary
`TokenizerFactory`	`addUserDefinedDictionary`	Adds a user segmentation dictionary
	`addDynamicUserDictionary`	Create and load new dynamic segmentation dictionary

10.6. Analysis Dictionaries

NOTE
Analysis dictionaries are not supported for Arabic, Persian, Romanian, Turkish, and Urdu.

Each entry is a word, followed by a tab and an analysis. The analysis must end with a lemma and a part-of-speech (POS) tag.

```
word    lemma[+POS]
```

For those languages for which RBL does not return POS tags, use `DUMMY`.

Variations. You can provide more than one *analysis* for a *word* or more than one version of a *word* for an *analysis*.

The following example includes two analyses for "telephone" (noun and verb), and two renditions of "dog" for the same analysis (noun).

```
telephone    telephone[+NOUN]
telephone    telephone[+VI]
dog    dog[+NOUN]
Dog    dog[+NOUN]
```

For some languages, the analysis may include special tags and additional information.

Contracted forms. For English, French, Italian, and Portuguese, `^=` is a separator for a contraction or elision.

English example:

```
doesn't    does[^=]not[+VDPRES]
```

Multi-Word Analysis. For English, Italian, Spanish, and Dutch, `^_` indicates a space.

English example:

```
IBM    International[^_]Business[^_]Machines[+PROP]
```

Compound Boundary. For Danish, Dutch, Norwegian, German, and Swedish, `^#` indicates the boundary between elements in a compound word. For Hungarian, the compound boundary tag is `^CB+`.

German example:

```
heimatländern    Heimat[^#]Land[+NOUN]
```

Compound Linking Element. For German `^/`, indicates a compound linking element. For Dutch, use `^//`.

German example:

```
arbeitskreis    Arbeit[^/]s[^#]Kreis[+NOUN]
```

Derivation Boundary or Separator for Clitics. For Italian, Portuguese, and Spanish, `^|` indicates a derivation boundary or separator for clitics.

Spanish example with derivation boundary:

```
duramente    duro[^|][+ADV]
```

Italian example with separator for clitics:

```
farti    fare[^|]tu[+VINF_CLIT]
```

Japanese Readings and Normalized Forms. For Japanese, `[^r]` precedes a reading (there may be more than one), and `[^n]` precedes a normalization. For example:

```
行わ    行う[^r]オコナワ[+V]
tv    テレビ[^r]テレビ[^n]テレビ[+NC]
アキュムレータ    アキュムレーター[^r]アキュムレータ[^n]アキュムレーター[+NC]
```

Korean Analysis. A Korean analysis uses a different pattern than the analysis for other languages. The pattern for an analysis in a user Korean dictionary is as follows:

```
Token    Mor1[/Tag1][^+]Mor2[/Tag2][^+]Mor3[/Tag3]
```

Where each `MorN` is a morpheme, consisting of one or more Korean characters, and `TagN` is the POS tag for that morpheme. `[^+]` indicates the boundary between morphemes.

Here's an example:

```
유전자이다    유전자[/NPR][^+]이[/CO][^+]다[/ECS]
```

If the analysis contains one noun morpheme, that morpheme is the lemma and the POS tag is the POS tag for that morpheme. If more than one of the morphemes are nouns, the lemma is the concatenation of those nouns (a compound). Example:

```
정보검색    정보[/NNC][^+]검색/[NNC]
```

Otherwise, the lemma is the first morpheme, and the POS tag is the POS tag associated with that morpheme.

You can override this algorithm for identifying the lemma and/or POS tag in a user dictionary entry by placing `[^L]`*`lemma`* and/or [^P][/*Tag*] at the end of the analysis. The lemma may or may not correspond to one of the morphemes in the analysis. For example:

```
유전자이다    유전자[/NNC][^+]이[/CO][^+]다[/ECS][^L]유전[^P][/NPR]
```

The `KoreanAnalysis` interface provides access to the morphemes and tags associated with a given token in either the standard Korean dictionary or a user Korean dictionary.

User Analysis Dictionary API

Class	Method	Task
`BaseLinguisticsFactory`	`addUserLemDictionary`	Add a user analysis dictionary
	`addUserAnalysisDictionary`	
	`addDynamicAnalysisDictionary`	Add a dynamic analysis dictionary
`AnalyzerFactory`	`addUserDefinedDictionary`	Add a user analysis dictionary
	`addDynamicUserDictionary`	Add a dynamic analysis dictionary

10.7. Many-to-one Normalization Dictionaries

A many-to-one normalization dictionary maps one or more variants to a normalized form. The first value on each line is the normalized form. The remainder of the entries on the line are the variants to be mapped to the normalized form. All values on the line are separated by tabs.

Example:

```
norm1    var1    var2
norm1    var3    var4    var5
```

User Many-to-one Normalization Dictionary API

Class	Method	Task
`BaseLinguisticsFactory`	`addDynamicNormalizationDictionary`	Create and load new dynamic normalization dictionary
`AnalyzerFactory`	`addDynamicNormalizationDictionary`	Create and load new dynamic normalization dictionary

Use the option `normalizationDictionaryPaths` to specify the static user normalization dictionaries.

10.8. CLA and JLA Dictionaries

The source file for a Chinese or Japanese user dictionary is UTF-8 encoded (see Valid Characters for Chinese or Japanese User Dictionary Entries [37]). The file may begin with a byte order mark (BOM). Empty lines are ignored. A comment line begins with #. The first line of a Japanese dictionary may begin `!DICT_LABEL` followed by **Tab** and an arbitrary string to set the dictionary's name, which is not currently used anywhere.

Each entry in the dictionary source file is a single line:

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

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

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

Chinese User Dictionary POS Tags

- ABBREVIATION
- ADJECTIVE
- ADVERB
- AFFIX
- CONJUNCTION
- CONSTRUCTION
- DERIVATIONAL_AFFIX
- DIRECTION_WORD
- FOREIGN_PERSON
- IDIOM
- INTERJECTION
- MEASURE_WORD
- NON_DERIVATIONAL_AFFIX

- NOUN
- NUMERAL
- ONOMATOPE
- ORGANIZATION
- PARTICLE
- PERSON
- PLACE
- PREFIX
- PREPOSITION
- PRONOUN
- PROPER_NOUN
- PUNCTUATION
- SUFFIX
- TEMPORAL_NOUN
- VERB
- VERB_ELEMENT

Japanese User Dictionary POS Tags

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

Note: For examples of standard (non-user-dictionary) use of the one and two-letter POS tags in the preceding list, see Japanese POS Tags – `BT_JAPANESE_RBLJE_2` [73].

Examples (the last three entries include readings)

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

The POS and decomposition pattern can be in 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 the tokenizer to decompose this compound entry into

```
東京
証券
取引所
```

CLA and JLA User Analysis Dictionary API

Class	Method	Task
BaseLinguisticsFactory	addUserLemDictionary	Add a user analysis dictionary
	addUserAnalysisDictionary	
	addDynamicAnalysisDictionary	Add a dynamic analysis dictionary
AnalyzerFactory	addUserDefinedDictionary	Add a user analysis dictionary
	addDynamicUserDictionary	Add a dynamic analysis dictionary

JLA Readings Dictionary API

Class	Method	Task
BaseLinguisticsFactory	addUserReadingDictionary	Adds a JLA readings dictionary.
	addDynamicReadingDictionary	Create and load new dynamic JLA readings dictionary
TokenizerFactory	addUserReadingDictionary	Adds a JLA readings dictionary
	addDynamicReadingDictionary	Create and load new dynamic JLA readings dictionary

11. Chinese Script Converter (CSC)

11.1. Overview

There are two standard forms of written Chinese: Simplified Chinese (SC) and Traditional Chinese (TC). SC is used in Mainland China. TC is used in Taiwan, Hong Kong, and Macau.

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, listed here from simplest to the most complex:

1. **Codepoint Conversion:** Codepoint conversion 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 that 头发 is a word, so the choice could be 發, in which case the end result 頭發 is nonsense. On the other hand, the choice of 髮 leads to errors for other words. So while conversion mapping is straightforward, it is unreliable.

2. **Orthographic Conversion:** The second level of conversion is orthographic. This level relies upon identification of the words in the input text. Within each word, orthographic variants of each character may be reflected in the conversion. In the above example, 头发 is identified as a word and is converted to a traditional variant of the word, 頭髮. There is no basis for converting it to 頭發, because the conversion considers the word as a whole rather than as a collection of individual characters.

3. **Lexemic Conversion:** 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 電腦 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.

NOTE

If the converter cannot provide the level of conversion requested (lexemic or orthographic), the next simpler level of conversion is provided.

For example, if you ask for a lexemic conversion, and none is available for a given token, CSC provides the orthographic conversion unless it is not available, in which case CSC provides a codepoint conversion.

The Chinese input may contain a mixture of TC and SC, and even some non-Chinese text. The Chinese Script Converter converts to the target (SC or TC), leaving any tokens already in the target form and any non-Chinese text unchanged.

11.2. CSC Options

CSC Options

Option	Description	Type (Default)	Supported Languages
`conversionLevel`	Indicates most complex conversion level to use	`CSConversionLevel` (`lexemic`)	Chinese
`language`	The language from which the `CSCAnalyzer` is converting	`LanguageCode`	Chinese, Simplified Chinese, Traditional Chinese
`targetLanguage`	The language to which the `CSCAnalyzer` is converting	`LanguageCode`	Chinese, Simplified Chinese, Traditional Chinese

See Initial and Path Options [9] for additional options

Enum Classes:

- `BaseLinguisticsOption`
- `CSCAnalyzerOption`

11.3. Using CSC with the ADM API

This example uses the `BaseLinguisticsFactory` and `CSCAnalyzer`.

1. Create a `BaseLinguisticsFactory` and set the required options.

```
final BaseLinguisticsFactory factory = new BaseLinguisticsFactory();
factory.setOption(BaseLinguisticsOption.rootDirectory, rootDirectory);
factory.setOption(BaseLinguisticsOption.licensePath, licensePath);
factory.setOption(BaseLinguisticsOption.conversionLevel, CSConversionLevel.orthographic.levelName());
factory.setOption(BaseLinguisticsOption.language, LanguageCode.SIMPLIFIED_CHINESE.ISO639_3());
factory.setOption(BaseLinguisticsOption.targetLanguage, LanguageCode.TRADITIONAL_CHINESE.ISO639_3());
```

2. Create an annotator to get translations.

```
final EnumMap<BaseLinguisticsOption, String> options = new EnumMap<>(BaseLinguisticsOption.class);
options.put(BaseLinguisticsOption.language, LanguageCode.SIMPLIFIED_CHINESE.ISO639_3());
options.put(BaseLinguisticsOption.targetLanguage, LanguageCode.TRADITIONAL_CHINESE.ISO639_3());
final Annotator cscAnnotator = factory.createCSCAnnotator(options);
```

3. Annotate the input text for tokens and translations.

```
final AnnotatedText annotatedText = cscAnnotator.annotate(inputText);
final Iterator<Token> tokenIterator = annotatedText.getTokens().iterator();
for (final String translation : annotatedText.getTranslatedTokens().get(0).getTranslations()) {
    final String originalToken = tokenIterator.hasNext() ? tokenIterator.next().getText() : "";
    outputData.format(OUTPUT_FORMAT, originalToken, translation);
}
```

11.4. Using CSC with the Classic API

The RBL distribution includes a sample (`CSCAnalyze`) that you can compile and run with an `ant` build script.

In a Bash shell script (Unix) or Command Prompt (Windows), navigate to `rbl-je-<version>/samples/csc-analyze` and call

```
ant run
```

The sample reads an input file in SC and prints each token with its TC conversion to standard out.

This example uses the `TokenizerFactory` and `CSCAnalyzer`.

1. Set up a `TokenizerFactory`.

```
TokenizerFactory tf = new TokenizerFactory();
tf.setOption(TokenizerOption.rootDirectory, rootDirectory);
tf.setOption(TokenizerOption.licensePath, licensePath);
```

2. Use the `TokenizerFactory` to create a `Tokenizer` to tokenize Chinese text.

```
Tokenizer tokenizer = tf.create(new StringReader(tcInput), LanguageCode.CHINESE);
```

3. Set up a `CSCAnalyzerFactory` with a conversion level.

```
CSCAnalyzerFactory caf = new CSCAnalyzerFactory();
caf.setOption(CSCAnalyzerOption.rootDirectory, rootDirectory);
caf.setOption(CSCAnalyzerOption.licensePath, licensePath);
caf.setOption(CSCAnalyzerOption.conversionLevel, "orthographic");
```

4. Use the `CSCAnalyzerFactory` to create a `CSCAnalyzer` to convert from TC to SC.

```
CSCAnalyzer cscAnalyzer =
    caf.create(LanguageCode.TRADITIONAL_CHINESE, LanguageCode.SIMPLIFIED_CHINESE);
```

5. Use the `CSCAnalyzer` to analyze each `Token` found by the `Tokenizer`.

```
Token token;
while ((token = tokenizer.next()) != null) {
    String tokenIn = new String(token.getSurfaceChars(),
        token.getSurfaceStart(),
        token.getLength());
    System.out.println("Input: " + tokenIn);
    cscAnalyzer.analyze(token);
```

6. Get the conversion (SC or TC) from each `Token`.

```
System.out.println("SC translation: " + token.getTranslation());
```

11.5. CSC User Dictionaries

CSC user dictionaries support orthographic and lexemic conversions between Simplified Chinese and Traditional Chinese. They are not used for codepoint conversion.

CSC user dictionaries follow the same format as other user dictionaries:

- The source file is UTF-8 encoded.
- The file may begin with a byte order mark (BOM).
- Each entry is a single line.
- Empty lines are ignored.

Once complete, the source file is compiled into a binary format for use in RBL.

Each entry contains two or three tab-delimited elements:

```
input_token orthographic_translation [lexemic_translation]
```

The input_token is the form you are converting from and the orthographic_translation and optional lexemic_translation are the form you are converting to.

Sample entries for a TC to SC user dictionary:

```
電腦      电脑      计算机
宇宙飛船      宇宙飞船
```

Compiling a CSC User Dictionary. In the `tools/bin` directory, RBL includes a shell script for Unix

```
rbl-build-csc-dictionary
```

and a .bat file for Windows

```
rbl-buld-csc-dictionary.bat
```

The script uses Java to compile the user dictionary. The operation is performed in memory, so you may require more than the default heap size. You can set heap size with the `JAVA_OPTS` environment variable. For example, to provide 8 GB of heap size, set `JAVA_OPTS` to `-Xmx8g`.

Unix shell:

```
export JAVA_OPTS=-Xmx8g
```

Windows command prompt:

```
set JAVA_OPTS=-Xmx8g
```

Compile the CSC user dictionary from the RBL root directory:

```
tools/bin/rbl-build-csc-dictionary INPUT_FILE OUTPUT_FILE
```

`INPUT_FILE` is the pathname of the source file you have created, and **`OUTPUT_FILE`** is the pathname of the binary compiled dictionary the tool creates. For example:

```
tools/bin/rbl-build-csc-dictionary my_tc2sc.txt my_tc2sc.bin
```

CSC User Dictionary API

Class	Method	Task
`BaseLinguisticsFactory`	`addUserCscDictionary`	Add a user CSC dictionary for a given language.
	`addDynamicCscDictionary`	Add dynamic CSC dictionary
`CSCAnalyzerFactory`	`addUserDefinedDictionary`	Add a user CSC dictionary
	`addDynamicCscDictionary`	Add a dynamic CSC dictionary

12. Using RBL in Apache Lucene

12.1. Introduction

RBL provides an API for integrating with Apache Lucene 4.3–8.8. See the Javadoc that accompanies this package for the complete API documentation.

A Lucene analyzer examines the text of fields and generates a token stream. Rosette provides an RBL base linguistics analyzer for Lucene, which uses RBL's linguistic tools.

RBL supports two usage patterns for incorporating Rosette Base Linguistics in a Lucene application.

- Use the RBL Lucene base linguistics analyzer [49] (`BaseLinguisticsAnalyzer`) to parse an input stream in one of the languages RBL supports and to generate a token stream with tokens and lemmas.
- Create your own analysis chain [50], using a language-specific RBL tokenizer and token filter.

12.2. Samples

Samples are included in the use cases below. The code comes from the Lucene 5.0 samples found in `rbl-je-<version>/samples/lucene-5_0`. Samples for other versions of Lucene are located in the sample directory with the correct Lucene version.

Lucene/Solr Version	RBL File name version	Solr lib JAR file[a]	Lucene Sample Directory
4.3–4.8	4_3	`btrbl-je-lucene-solr-4_3-<version>.jar`	`rbl-je-<version>/samples/lucene-4_3`
4.9	4_9	`btrbl-je-lucene-solr-4_9-<version>.jar`	`rbl-je-<version>/samples/lucene-4_9`
4.10	4_10	`btrbl-je-lucene-solr-4_10-<version>.jar`	`rbl-je-<version>/samples/lucene-4_10`
5.0–5.5	5_0	`btrbl-je-lucene-solr-5_0-<version>.jar`	`rbl-je-<version>/samples/lucene-5_0`
6.0–6.1	6_0	`btrbl-je-lucene-solr-6_0-<version>.jar`	`rbl-je-<version>/samples/lucene-6_0`
6.2–8.9	6_2	`btrbl-je-lucene-solr-6_2-<version>.jar`	`rbl-je-<version>/samples/lucene-6_2`

[a] <version> is RBL version

12.3. Lucene Options

The following options are passed in through an options `Map`.

Lucene Filter Options

Option	Description	Type (Default)	Supported Languages
`addLemmaTokens`	Indicates whether the token filter should add the lemmas (if none, the steps) of each surface token to the tokens being returned..	Boolean (true)	All
`addReadings`	Indicates whether the token filter should add the readings of each surface token to the tokens being returned	Boolean (false)	Chinese, Japanese
`identifyContractionComponents`	Indicates whether the token filter should identify contraction components as contraction components rather than as lemmas	Boolean (false)	All
`replaceTokensWithLemmas`	Indicates whether the token filter should replace a surface token with its lemma. Disambiguation must be enabled.	Boolean (false)	All

Enum Classes:

- `FilterOption`

12.4. Using the RBL Lucene Base Linguistics Analyzer

The RBL Lucene base linguistics analyzer (BaseLinguisticsAnalyzer) provides an analysis chain with a language-specific tokenizer and token filter.

You can configure the analyzer with a number of tokenizer and token filter options.

The analysis chain includes the Lucene filters LowerCaseFilter and CJKWidthFilter,. It also provides support for including a StopFilter.

Japanese Analyzer Sample. This Lucene sample, `JapaneseAnalyzerSample.java`, uses the base linguistics analyzer to generate an enriched token stream from Japanese text.

1. Assemble a set of options that include the root directory, the path to the Rosette license, the language to analyze (Japanese), and the generation of part-of-speech tags for the tokens.

```
File rootPath = new File(rootDirectory);
String licensePath = new File(
       rootPath, "licenses/rlp-license.xml").getAbsolutePath();

Map<String, String> options = new HashMap<>();
options.put("language", "jpn");
options.put("rootDirectory", rootDirectory);
options.put("licensePath", licensePath);
options.put("addReadings", "true");
```

2. Instantiate a Japanese base linguistics analyzer with these options.

```
rblAnalyzer = new BaseLinguisticsAnalyzer(options);
```

3. Read in a Japanese text file.

4. Use the analyzer to generate a token stream. The token stream contains tokens, lemmas that are not identical to their tokens, and readings. Disambiguation is turned off by default for Japanese, so multiple analyses may be returned for each token. To turn on disambiguation, add

```
options.put("disambiguate", "true");
```

   to the construction of the `options` Map.

5. Write each element in the token stream with its type attribute to an output file.

To run the sample, in a Bash shell (Unix) or Command Prompt (Windows), navigate to `rbl-je-<version>/samples/lucene-5_0` and use the Ant build script:

```
ant runAnalyzer
```

The sample reads `rbl-je-<version>/samples/data/jpn-text.txt` and writes the output to `jpn-Analyzed-byAnalyzer.txt`.

The output includes each token, lemma, and reading in the token stream on a separate line with the type attribute for each element: `<ALNUM>` for tokens, `<LEMMA>` for lemmas, and `<READING>` for readings. There may be more than one analysis (hence lemma and reading in the sample output) for a token; lemmas are not put into the token stream when identical to the token.

For example:

```
メルボルン    <ALNUM>
メルボルン    <READING>
で    <ALNUM>
行わ    <ALNUM>
行う    <LEMMA>
オコナワ    <READING>
れ    <ALNUM>
る    <LEMMA>
れる    <LEMMA>
レ    <READING>
```

12.5. Creating your own RBL Analysis Chain

When creating an analysis chain, you can do the following:

- Use the `BaseLinguisticsTokenizerFactory` to generate a language-specific tokenizer that applies Rosette Base Linguistics to tokenize text.
- Use the `BaseLinguisticsTokenFilterFactory` to generate a language-specific token filter that enhances a stream of tokens.
- Add other token filters to the analysis chain.

One of the tasks of the token filter is to set tokens' token types. For example, given a lemma token, the token filter gives it the type `<LEMMA>`. Given a contraction component token, the token filter has a choice: by default, the type is set to `<LEMMA>`, but when the `identifyContractionComponents` option is enabled, the type is set to `<CONT>`.

12.5.1. Japanese Tokenizer and Filter Sample

This Lucene sample, `JapaneseTokenizerAndFilterSample.java`, creates an analysis chain to generate an enriched token stream.

1. Use a tokenizer factory to set up a language-specific base linguistics tokenizer, which puts tokens in the token stream.

```
tokenizerFactory = new TokenizerFactory();
tokenizerFactory.setOption(TokenizerOption.rootDirectory, rootDirectory);
tokenizerFactory.setOption(TokenizerOption.licensePath, licensePath);
Map<String, String> options = new HashMap<>();
options.put("language", "jpn");
options.put("rootDirectory", rootDirectory);
options.put("addReadings", "true");
tokenFilterFactory = new BaseLinguisticsTokenFilterFactory(options);
```

2. Use a base linguistics token filter factory to set up language-specific base linguistics token filter, which adds lemmas and readings to the tokens in the token stream.

```
Tokenizer tokenizer = new BaseLinguisticsTokenizer(tokenizerFactory.create(null, LanguageCode.JAPANESE));
tokenizer.setReader(input);

TokenStream tokens = tokenFilterFactory.create(tokenizer);
```

3. To replicate the behavior of the analyzer in the previous example, this sample also includes the LowerCaseFilter and CJKWidthFilter.

```
tokens = new LowerCaseFilter(tokens);
tokens = new CJKWidthFilter(tokens);
```

4. Write each element in the token stream with its type attribute to an output file.

To run the sample, in a Bash shell (Unix) or Command Prompt (Windows), navigate to `rbl-je-<version>/samples/lucene-5_0` and use the Ant build script:

```
ant runTokenizerAndFilter
```

The example reads the same file as the previous sample and writes the output to a `jpn-analyzed-byTokenizerAndFilter.txt`. The content matches the content generated by the previous example.

12.5.2. Using the BaseLinguisticsSegmentationTokenFilter

If you are using your own whitespace tokenizer and processing text that requires segmenting Chinese, Japanese, or Thai, you can use the `BaseLinguisticsSegmentationTokenFilterFactory` to create a `BaseLinguisticsSegmentationTokenFilter`, then place the segmentation token filter in an analysis chain following the whitespace tokenizer and preceding other filters, such as a base linguistics token filter.

The segmentation token filter segments each of the tokens from the whitespace tokenizer into individual tokens where necessary. Refer to the Javadoc for the RBL API for Lucene for more information.

12.6. Analyses Attributes

You can use the `com.basistech.rosette.lucene.AnalysesAttribute` object to gather linguistic data about the text in a document. Depending on the language, the data may include tokens, normalized tokens, lemmas, part-of-speech tags, readings, compound components, and Semitic roots.

The Lucene sample, `AnalysesAttributeSample.java`, illustrates this.

To run the sample with the German sample file, navigate to `rbl-je-<version>/samples/lucene-<version>`, and call ant as follows:

```
ant -Dtest.language=deu runAnalysesAttribute
```

The sample writes the output to `deu-analysesAttributes.txt`.

12.7. Case Sensitivity During the Analysis

In some languages, case distinctions are meaningful. For example, in German, a word may be a noun if it begins with an upper-case letter, and not a noun if it does not. As a result, RBL delivers higher accuracy in selecting lemmas and splitting compounds when it can process text with correct casing. On the other hand, users typing in queries may be sloppy with capital letters.

For this reason, the default behavior of the Lucene integration is to perform the following analysis steps:

1. tokenize
2. determine lemmas
3. map to lowercase

The result is that the index contains the lowercase form of the most accurately selected lemma.

However, some applications work with text in which case distinctions are not reliably present, even in languages where they are important. These applications need to determine lemmas and compound components even though the spelling is nominally incorrect with respect to case.

To support these applications, RBL provides a 'case-insensitive' mode of operation. In this mode, RBL performs the following analysis steps:

1. tokenize, ignoring the case of abbreviations and such
2. determine lemmas, ignoring case in choosing lemmas and compound components
3. map to lowercase

The mapping is still required to ensure that the index or query ends up with uniformly lowercase text.

To specify case sensitivity for the analysis, set `com.basistech.rosette.bl.AnalyzerOption.caseSensitive` to `true` or `false`. By default, the setting is `true`, except for Danish, Norwegian, and Swedish, for which our dictionaries are lowercase and the setting is `false` irrespective of the user setting.

When you are making this setting in the `com.basistech.rosette.lucene` package, include the `caseSensitive` option as a string. For example:

```
Map<String, String> options = new HashMap<>();
options.put("language", LanguageCode.ITALIAN.ISO639_30);
options.put("caseSensitive", "true");
TokenFilterFactory factory = new BaseLinguisticsTokenFilterFactory(options);
```

12.8. Activating User Dictionaries in Lucene

User Dictionaries [34] can be used when using RBL with Lucene.

In the `com.basistech.rosette.lucene` package, `BaseLinguisticsTokenizerFactory` and `BaseLinguisticsTokenFilterFactory` can load segmentation and analysis dictionaries respectively.

The path options are provided as a list of paths, separated by semicolons or the OS-specific path separator.

Lucene User Dictionary Path Options

Option	Description	Type	Supported Languages
`lemDictionaryPath`	A list of paths to user lemma dictionaries.	List of Paths	Chinese, Czech, Danish, Dutch, English, French, German, Greek, Hebrew, Hungarian, Italian, Japanese, Korean, Norwegian, Polish, Portuguese, Russian, Spanish, Swedish, Thai
`segDictionaryPath`	A list of paths to user segmentation dictionaries.	List of Paths	All

Option	Description	Type	Supported Languages
`userDefinedDictionaryPath`	A list of paths to user dictionaries.	List of Paths	All
`userDefinedReadingDictionaryPath`	A list of paths to reading dictionaries.	List of Paths	Japanese

`BaseLinguisticsTokenizerFactory` provides the method `addUserDefinedDictionary` for adding a segmentation dictionary. For example:

```
Map<String, String> args = new HashMap<>();
args.put(TokenizerOption.language.name(), LanguageCode.JAPANESE.ISO639_30);
args.put(TokenizerOption.nfkcNormalize.name(), "true");
BaseLinguisticsTokenizerFactory factory = new BaseLinguisticsTokenizerFactory(args);
factory.addUserDefinedDictionary(LanguageCode.JAPANESE, "/path/to/my/jpn-dict.bin");
```

The constructor for `BaseLinguisticsTokenFilterFactory` takes a `Map` of options. Use the `userDefinedDictionaryPath` option to load an analysis dictionary:

```
Map<String, String> options = new HashMap<>();
options.put("language", LanguageCode.ITALIAN.ISO639_30);
options.put("userDefinedDictionaryPath", "/path/to/my/ita-dict.bin");
options.put("caseSensitive", "true");
TokenFilterFactory factory = new BaseLinguisticsTokenFilterFactory(options);
```

12.9. Using CSC with Lucene

1. Set up a `com.basistech.rosette.lucene.BaseLinguisticsTokenizerFactory`.
2. Use the `TokenizerFactory` to create a `com.basistech.rosette.lucene.BaseLinguisticsTokenizer`, which contains a Lucene `Tokenizer`.
3. Set up a `com.basistech.rosette.lucene.BaseLinguisticsCSCTokenFilterFactory`
4. Use the `BaseLinguisticsCSCTokenFilterFactory` to create a `com.basistech.rosette.lucene.BaseLinguisticsCSCTokenFilter` to convert from TC to SC or vice versa.
5. Use the `BaseLinguisticsCSCTokenFilter` to convert each `Token` found by the `Tokenizer`.

RBL/Lucene Distribution Sample. For supported versions of Lucene, the RBL distribution includes a sample (`CSCCharTermAttributeSample`) that you can compile and run with an `ant` build script.

In a Bash shell script (Unix) or Command Prompt (Windows), navigate to the samples directory (`rbl-je-<version>/samples) and the file for your version of lucene (/csc-analyze-<luceneversion>` and call:

```
ant run
```

The sample reads an input file in SC and prints the TC conversion for each token to standard out.

13. Using RBL in Apache Solr

13.1. Introduction

You can configure Apache Solr search to use RBL for both indexing documents as well as for queries.

To index and search documents with RBL in a Solr application, you must add JARs to the Solr classpath and define Solr analysis chains that apply the RBL analysis components to process text at index and query time.

13.2. Adding to the Solr Classpath

Add the following `lib` elements to the `solrconfig.xml` for each Solr collection you are using.

```
<lib path="<RBLJE_ROOT>/lib/btrbl-je-<version>.jar"/>
<lib path="<RBLJE_ROOT>/lib/btcommon-api-<version>.jar"/>
<lib path="<RBLJE_ROOT>/lib/slf4j-api-<version>.jar"/>
<lib path="<RBLJE_ROOT>/lib/btrbl-je-lucene-solr-<version>-<version>.jar"/>
```

where you replace **<RBLJE_ROOT>** with the path to the root of your RBL installation and take the full file names with correct <version> values from the RBL `<lib path>`. They can change with each new release.

For example, if the root of the RBL installation is `/opt/local/bt/rbl-je` and the version of Solr is 8.6, the lib paths are:

```
<lib path="/opt/local/bt/rbl-je/lib/btrbl-je-7.36.0.c62.2.jar"/>
<lib path="/opt/local/bt/rbl-je/lib/btcommon-api-37.0.1.jar"/>
<lib path="/opt/local/bt/rbl-je/lib/slf4j-api-1.7.28.jar"/>
<lib path="/opt/local/bt/rbl-je/btrbl-je-lucene-solr-6_2-7.36.0.c62.2.jar"/>
```

The correct Lucene/Solr version files are listed in the section Lucene/Solr Versions [7].

The SLF4J JARs enable logging [7].

13.3. Defining a Solr Analysis Chain

In the Solr `schema.xml` or `managed-schema`, add a `fieldType` element and a corresponding `field` element for the language of the documents processed by the application.

Field Type. The `fieldType` includes two analyzers: one for indexing documents and one for querying documents. Each analyzer contains a tokenizer and a token filter.

Here, for example, is a `fieldType` for Japanese:

```
<fieldtype name="basis-japanese" class="solr.TextField">
  <analyzer type="index">
    <tokenizer class="com.basistech.rosette.lucene.BaseLinguisticsTokenizerFactory"
               language="jpn"
               rootDirectory="<RBLJE_ROOT>"
    />
    <filter class="com.basistech.rosette.lucene.BaseLinguisticsTokenFilterFactory"
            language="jpn"
            rootDirectory="<RBLJE_ROOT>"
    />
  </analyzer>
  <analyzer type="query">
    <tokenizer class="com.basistech.rosette.lucene.BaseLinguisticsTokenizerFactory"
               language="jpn"
               rootDirectory="<RBLJE_ROOT>"
    />
    <filter class="com.basistech.rosette.lucene.BaseLinguisticsTokenFilterFactory"
            language="jpn"
            rootDirectory="<RBLJE_ROOT>"
            query="true"
    />
  </analyzer>
</fieldtype>
```

where you replace **<RBLJE_ROOT>** with the path to the root of your RBL installation. The `fieldType name` indicates the language, and each `language` attribute is set to the ISO 693-3 [57] language code for Japanese.

NOTE
You can incorporate any Solr filter you need, such as the Solr lowercase filter; however, they should be added into the chain after the Base Linguistics token filter. If you modify the token stream too radically before RBL, you will degrade its ability to analyze the text.

Field. The analysis chain requires a `field` definition with a `type` attribute that maps to the `fieldType`. For the Japanese example above, add the following `field` definition to `schema.xml`.

```
<field name="text-japanese" type="basis-japanese" indexed="true" stored="true"/>
```

In your Solr application, you can now index and query Japanese documents placed in the `text-japanese` field.

13.4. Using Options in Solr

Most API options [88] can be used in a Solr analysis chain. In Solr, you do not directly use the option enum classes. Instead, options are specified in the schema using the format `option="value"`.

When specifying options for the `tokenizer` (class `BaseLinguisticsTokenizerFactory`), use options in the `TokenizerOption` enum class. When specifying options for a `filter` (class `BaseLinguisticsTokenFilterFactory`), use options in the `AnalyzerOption` and `FilterOption` enum classes.

Example:

```
<fieldType class="solr.TextField" name="basis-french">
    <analyzer type="index">
        <tokenizer class="com.basistech.rosette.lucene.BaseLinguisticsTokenizerFactory" language="fra"
        rootDirectory="${bt_root}"/>
            <filter class="com.basistech.rosette.lucene.BaseLinguisticsTokenFilterFactory" language="fra"
        rootDirectory="${bt_root}"/>
            <filter class="solr.LowerCaseFilterFactory"/>
            <filter class="org.apache.solr.analysis.RemoveDuplicatesTokenFilterFactory"/>
    </analyzer>
    <analyzer type="query">
         <tokenizer class="com.basistech.rosette.lucene.BaseLinguisticsTokenizerFactory" language="fra"
    query="true" rootDirectory="${bt_root}"/>
         <filter class="com.basistech.rosette.lucene.BaseLinguisticsTokenFilterFactory" language="fra"
    query="true" rootDirectory="${bt_root}"/>
         <filter class="solr.LowerCaseFilterFactory"/>
         <filter class="org.apache.solr.analysis.RemoveDuplicatesTokenFilterFactory"/>
    </analyzer>
</fieldType>
```

13.5. Activating User Dictionaries in Solr

User Dictionaries [34] can be used when using RBL with Solr.

Use the option `userDefinedDictionaryPath` as shown in this example:

```
<fieldType class="solr.TextField" name="basis-japanese">
    <analyzer>
        <tokenizer class="com.basistech.rosette.lucene.BaseLinguisticsTokenizerFactory"
            tokenizerType="spaceless_lexical"
            language="jpn"
            rootDirectory="${bt_root}"
            userDefinedDictionaryPath= "/path/to/my/jpn-udd.bin"/>
        <filter class="com.basistech.rosette.lucene.BaseLinguisticsTokenFilterFactory"
            tokenizerType="spaceless_lexical"
            language="jpn"
            rootDirectory="${bt_root}"/>
    </analyzer>
</fieldType>
```

Here is an example of using a JLA reading dictionary:

```
<fieldType class="solr.TextField" name="basis-japanese-rd">
    <analyzer>
        <tokenizer class="com.basistech.rosette.lucene.BaseLinguisticsTokenizerFactory"
            tokenizerType="spaceless_lexical"
            readings="true"
            language="jpn"
            rootDirectory="${bt_root}"
            userDefinedReadingDictionaryPath="/path/to/my/readings.bin"/>
        <filter class="com.basistech.rosette.lucene.BaseLinguisticsTokenFilterFactory"
            tokenizerType="spaceless_lexical"
            language="jpn"
            rootDirectory="${bt_root}"/>
    </analyzer>
</fieldType>
```

14. Language Codes

14.1. Canonical Language Code

RBL canonicalizes some language codes to other language codes. These canonicalization rules apply to all APIs.

`LanguageCode.NORWEGIAN` is canonicalized to `LanguageCode.NORWEGIAN_BOKMAL` immediately upon any input to the API. This means that there can be no distinguishing between them. In particular, an `Analyzer` built from a factory configured to use Norwegian will report its language as Norwegian Bokmål instead.

Similarly, `LanguageCode.SIMPLIFIED_CHINESE` and `LanguageCode.TRADITIONAL_CHINESE` are canonicalized to `LanguageCode.CHINESE` immediately. The one exception is that they are not canonicalized as inputs to or outputs from the Chinese Script Converter.

Those are the only language code canonicalizations. Although RBL internally treats Afghan Persian and Iranian Persian as Persian, they are not considered the same language. This makes it possible to configure different user dictionaries for each variety of Persian, even though they are otherwise processed identically.

14.2. ISO 639-3 Language Codes

RBL uses ISO 639-3 language codes to specify languages as strings. There are a few nonstandard language codes, as indicated. RBL also accepts 2-letter codes specified by the ISO-639-1 standard. See the Javadoc for more details on language codes.

Language	Code
Arabic	`ara`
Catalan	`cat`
Chinese	`zho`
Chinese (Simplified)	`zhs`[a]
Chinese (Traditional)	`zht`[a]
Czech	`ces`
Danish	`dan`
Dutch	`nld`
English	`eng`
Estonian	`est`
Finnish	`fin`
French	`fra`
German	`deu`
Greek	`ell`
Hebrew	`heb`
Hungarian	`hun`
Indonesian	`ind`
Italian	`ita`
Japanese	`jpn`
Korean	`kor`
Latvian	`lav`
Malay (Standard)	`zsm`
North Korean	`qkp`
Norwegian	`nor`
Norwegian Bokmål	`nob`
Norwegian Nynorsk	`nno`

Language	Code
Pashto	pus
Persian	fas
Persian, Afghan	prs
Persian, Iranian	pes
Polish	pol
Portuguese	por
Romanian	ron
Russian	rus
Serbian	srp
Slovak	slk
South Korean	qkr
Spanish	spa
Swedish	swe
Tagalog	tgl
Thai	tha
Turkish	tur
Urdu	urd
Unknown	xxx [a]

[a]Not a real ISO 639-3 code.

15. Part-of-Speech Tags

NOTE

For a mapping of the part-of-speech tags that appear in this appendix to the tags used in the Penn Treebank Project, see the tab-delimited .csv files (one per language) distributed along with this *Application Developer's Guide* in the `penn_treebank` subdirectory. The RBL Korean part-of-speech tags conform to the Penn Treebank standard.

In RBL, each language has its own set of POS tags and a few languages have multiple tag sets. Each tag set is identified by an identifier, which is a value of the `TagSet` enum. When RBL outputs a POS tag, it also lists the identifier for the tag set it came from. Output from a single language may contain POS tags from multiple tag sets, including the language-neutral set.

The header of each table lists the tag set identifier. For example, the English tag set is identified as `BT_ENGLISH`.

For Chinese and Japanese using statistical models for tokenization (tag sets `BT_CHINESE_RBLJE_2` and `BT_JAPANESE_RBLJE_2`), if the analyzer cannot find the lemma for a word in its analysis dictionary, it will return GUESS for a part-of-speech tag. GUESS is not really a part-of-speech and does not appear below in this appendix.

15.1. Arabic POS Tags – BT_ARABIC

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	طَوُنِي، 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	() ؛:،؟.:
PV	perfective verb	كَانَت، قَالَ
PV_PASS	passive perfective verb	أُعْتُبَر
RC_PART	resultative clause particle	فَلَمَا
REL_ADV	relative adverb	حَيْثُ
REL_PRON	relative pronoun	اَلَّذِي، اَللَّذَانِ
SUB_CONJ	subordinating conjunction	إِذَا، إِذ
VERB_PART	verbal particle	لَقَدْ

15.2. Chinese POS Tags – Simplified and Traditional – BT_CHINESE

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
FOREIGN	non-Chinese	c123	c123
I	interjection	吧	吧
J	conjunction	但是	但是

Tag	Description	Simplified Chinese	Traditional Chinese
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
WV	word element - verb	以	以
X	generic affix	老	老
XP	generic prefix	可	可
XS	generic suffix	员	員

15.3. Chinese POS Tags – Simplified and Traditional – `BT_CHINESE_RBLJE_2`

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

Tag	Description	Simplified Chinese	Traditional Chinese
W	derivational suffix	家	家
WL	direction word	下	下
WV	word element - verb	以	以
X	generic affix	老	老
XP	generic prefix	可	可
XS	generic suffix	员	員

15.4. Czech POS Tags – `BT_CZECH`

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 mladý [muž]
	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]
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]

Tag	Description	Example
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]

15.5. Dutch POS Tags – `BT_DUTCH`

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]
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

Tag	Description	Example
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
WADV	interrogative adverb	waarom [gaat hij]
WDET	interrogative or relative determiner	[de vrouw] wier [man....]
WPRO	interrogative or relative pronoun	[de vraag] wie ...

15.6. English POS Tags – `BT_ENGLISH`

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!

Tag	Description	Example
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 ...]
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 [!]
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

Tag	Description	Example
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]

15.7. French POS Tags – `BT_FRENCH`

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
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

Tag	Description	Example
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

15.8. German POS Tags – `BT_GERMAN`

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
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]

Tag	Description	Example
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
RELPRO	relative pronoun	[der Mann,] der [lacht]
REZPRO	reciprocal *einander*	einander
SENT	sentence final punctuation	. ? !
SYMB	symbols	@, %, x311
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?]

15.9. Greek POS Tags – `BT_GREEK`

Tag	Description	Example
ADJ	adjective	παιδικό
ADV	adverb	ευχαρίστως
ART	article	η, της
CARD	cardinal	χίλια
CLIT	clitic (pronoun)	τον, τού
CM	comma	,
COORD	coordinating conjunction	και
COSUBJ	conjunction with subjunctive	αντί [να]
CURR	currency	$
DIG	digits	123
FM	foreign word	article

Tag	Description	Example
FUT	future tense particle	θα
INTJ	interjection	χμ
ITEM	item	1.2
NEG	negation particle	μη
NOUN	common noun	βιβλίο
ORD	ordinal	τρίτα
PERS	personal pronoun	εγώ
POSS	possessive pronoun	μας, τους
PREP	preposition	άνευ
PREPART	preposition with article	στο
PRON	pronoun	αυτοί
PRONREL	relative pronoun	οποίες
PROP	proper noun	Μαρία
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)	γράψε
VIND	verb (indicative)	γράφεις
VINF	verb (infinitive)	γράφει
VPP	participle	δικασμένο

15.10. Hebrew POS Tags – `MILA_HEBREW`

Tag	Description	Example
adjective	adjective	נפלאי
adverb	adverb	מאחוריהם
conjunction	conjunction	אך, ועוד
copula	copula	איננה, תהיו
existential	existential	ההיה, יש
indInf	independent infinitive	היוסדה
interjection	interjection	אח, חבל"ז
interrogative	interrogative	לאן
modal	modal	צריכים
negation	negation particle	לא
noun	common noun	אורולוגיה
numeral	numeral	613, שמונה
participle	participle	משרטטה
passiveParticiple	passive participle	סגורה
preposition	preposition or prepositional phrase	עם, בפניו
pronoun	pronoun	/זה
properName	proper noun	שרון
punctuation	punctuation	.
quantifier	quantifier or determiner	רובן

Tag	Description	Example
title	title or honorific	גב', זצ"ל
unknown	unknown	ומלמם'מ
verb	verb	לכתוב
wPrefix	prefix	פסאודו

15.11. Hungarian POS Tags – `BT_HUNGARIAN`

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
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

Tag	Description	Example
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
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

15.12. Italian POS Tags – `BT_ITALIAN`

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.

Tag	Description	Example
LET	single letter	[di tipo] C
NPL	plural noun	case
NSG	singular noun	casa, balsamo
ORDPL	plural ordinal	terzi
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
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

Tag	Description	Example
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

15.13. Japanese POS Tags – `BT_JAPANESE`

Tag	Description	Example
AA	adnominal adjective	その[人], この[日], 同じ
AJ	normal adjective	美し, 嬉し, 易し
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	約, 翌, 築, 乾元
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

15.14. Japanese POS Tags – BT_JAPANESE_RBLJE_2

Tag	Description	Example
AA	adnominal adjective	その[人], この[日], 同じ
AJ	normal adjective	美し, 嬉し, 易し
AN	adjectival noun	きれい[だ], 静か[だ], 正確[だ]
AUXVB	auxiliary verb	た, ない, らしい
D	adverb	じっと, じろっと, ふと
FS	non-derivational suffix	[綺麗]な, [派手]だ
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	約, 翌, 築, 乾元
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	。, 「」 ( ) ;
UNKNOWN	unknown	デパ[地下], ヴェロ
V	verb	書く, 食べます, 来た
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

15.15. Korean POS Tags – BT_KOREAN

Tag	Description	Examples
ADC	conjunctive adverb	그리고, 그러나, 및, 혹은
ADV	constituent or clausal adverb	매우, 조용히, 제발, 만일
CO	copula	이
DAN	configurative or demonstrative adnominal	새, 헌, 그
EAN	adnominal ending	는/ㄴ
ECS	coordinate, subordinate, adverbial, complementizer ending	고, 므로, 게, 다고, 라고
EFN	final ending	는다/ㄴ다, 니, 는가, 는지, 어라/라, 자, 구나
ENM	nominal ending	기, 음
EPF	pre-final ending (tense, honorific)	었, 시, 겠
IJ	exclamation	아
NFW	word written in foreign characters	Clinton, Computer
NNC	common noun	학교, 컴퓨터
NNU	ordinal or cardinal number	하나, 첫째, 1, 세
NNX	dependent noun	것, 등, 년, 달라, 적
NPN	personal or demonstrative pronoun	그, 이것, 무엇
NPR	proper noun	한국, 클린톤
PAD	adverbial postposition	에서, 로
PAN	adnominal postposition	의, 이라는

Tag	Description	Examples
PAU	auxiliary postposition	만, 도, 는, 마저
PCA	case postposition	가/이, 을/를, 의, 야
PCJ	conjunctive postposition	와/과, 하고
SCM	comma	,
SFN	sentence ending marker	. ? !
SLQ	left quotation mark	‘ (“ {
SRQ	right quotation mark	’) ” }
SSY	symbol	… ; : -
VJ	adjective	예쁘, 다르
VV	verb	가, 먹
VX	auxiliary predicate	있, 하
XPF	prefix	제
XSF	suffix	님, 들, 적
XSJ	adjectivization suffix	스럽, 답, 하
XSV	verbalization prefix	하, 되, 시키

15.16. Language Neutral POS Tags – `BT_LANGUAGE_NEUTRAL`

Tag	Description	Example
ATMENTION	@mention	@basistechnology
EMAIL	email address	email@example.com
EMO	emoji or emoticon	:-)
HASHTAG	hashtag	#BlindGuardian
URL	URL	http://www.basistech.com/

15.17. Persian POS Tags – `BT_PERSIAN`

Tag	Description	Example
ADJ	adjective	بزرگ
ADV	adverb	تقریبا
CONJ	conjunction	یا
DET	indefinite article/determiner	هر
EOS	end of sentence indicator	.
INT	interjection or exclamation	عجب
N	noun	افزایش
NON_FARSI	not Arabic script	a b c
NPROP	proper noun	مائیکروسافت
NUM	number	ده
PART	particle	می
PREP	preposition	به
PRO	pronoun	این
PUNC	punctuation, other than end of sentence	, : “ ”
UNK	unknown	نرژی
VERB	verb	گفتم
VINF	infinitive	خریدن

15.18. Polish POS Tags – `BT_POLISH`

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, 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, fiku-miku
PPERS	personal pronoun	ja, on, ona, my, wy, mnie, tobie, jemu, nam, mi, 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]
PRON	pronoun: demonstrative	[w] tym [czasie]
	pronoun: indefinite	wszystkie [stopy procentowe], jakieś [nienaturalne rozmiary]
	pronoun: possessive	nasi [dwaj bracia]
	pronoun: interrogative	Jaki [masz samochód?]
PRTCL	particle	także, nie, tylko, już
PT/AUX	particle with auxiliary *być*	gdzie [byliście]
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]

Tag	Description	Example
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...]

15.19. Portuguese POS Tags – `BT_PORTUGUESE`

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
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

Tag	Description	Example
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
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

15.20. Russian POS Tags – `BT_RUSSIAN`

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]

15.21. Spanish POS Tags – `BT_SPANISH`

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
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

Tag	Description	Example
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)	' ¡ ¿ : {
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

15.22. Urdu POS Tags – `BT_URDU`

Tag	Description	Example
ADJ	adjective	بہترین
ADV	adverb	تاہم
CONJ	conjunction	یا
DET	indefinite article/determiner	ایک
EOS	end of sentence indicator	.
INT	interjection or exclamation	جیئے
N	noun	مہینہ

Tag	Description	Example
NON_URDU	not Arabic script	a b c
NPROP	proper noun	مائیکروسافٹ
NUM	number	اٹھارہ
PART	particle	غیر
PREP	preposition	با
PRO	pronoun	وہ
PUNC	punctuation other than end of sentence	،
UNK	unknown	اپنیاں
VERB	verb	کرتی

15.23. Universal POS Tags – `UPT16_V1`

The universal tags are coarser than the language-specific tags, but enable tracking and comparison across languages.

To return universal POS tags in place of language-specific tags, use the Annotated Data Model (ADM) and `BaseLinguisticsFactory` to set `BaseLinguisticsOption.universalPosTag` to `true`. See Returning Universal POS Tags. [24]

Tag	Description
ADJ	adjective
ADP	adposition
ADV	adverb
AUX	auxiliary verb
CONJ	coordinating conjunction
DET	determiner
INTJ	interjection
NOUN	noun
NUM	numeral
PART	particle
PRON	pronoun
PROPN	proper noun
PUNCT	punctuation
SCONJ	subordinating conjunction
SYM	symbol
VERB	verb
X	other

16. Options

16.1. General Options

The following options are described in more detail in Initial and Path Options [9].

If the option `rootDirectory` is specified, then the string `${rootDirectory}` takes that value in the `dictionaryDirectory`, `modelDirectory`, and `licensePath` options.

Initial and Path Options

Option	Description	Type (Default) (Default)	Supported Languages
`dictionaryDirectory`	The path of the lemma and compound dictionary, if it exists.	Path ${rootDirectory}/dicts	All
`language`	The language to process by analyzers or tokenizers created by the factory.	Language code	All
`licensePath`	The path of the RBL license file.	Path ${rootDirectory}/licenses/rlp-license.xml	All
`licenseString`	The XML license content, overrides `licensePath`.	String	All
`modelDirectory`	The directory containing the model files.	Path ${rootDirectory}/models	All
`rootDirectory`	Set the root directory. Also sets default values for other required options (`dictionaryDirectory`, `licensePath`, `licenseString`, and `modelDirectory`).	Path	All

16.2. Tokenizer Options

The following options are described in more detail in Tokenizers [14].

General Tokenizer Options

Option	Description	Type (Default)	Supported Languages
`tokenizerType`	Selects the tokenizer to use	`TokenizerType` `SPACELESS_STATISTICAL` for Chinese, Japanese, Thai; `ICU` for all other languages	All
`caseSensitive`	Indicates whether tokenizers produced by the factory are case sensitive. If false, they ignore case distinctions.	Boolean (true)	Czech, Danish, Dutch, English, French, German, Greek, Hebrew, Hungarian, Indonesian, Italian, Malay (Standard), Norwegian, Polish, Portuguese, Russian, Spanish, Swedish, Tagalog
`defaultTokenizationLanguage`	Specify language to use for script regions, other than the script of the overall language.	Language code (`xxx`)	Chinese, Japanese, Thai

Option	Description	Type (Default)	Supported Languages
`minNonPrimaryScriptRegionLength`	Minimum length of sequential characters that are not in the primary script. If a non-primary script region is less than this length and adjacent to a primary script region, it is appended to the primary script region.	Integer (10)	Chinese, Japanese, Thai
`tokenizeForScript`	Indicates whether to use a different word-breaker for each script. If false, uses script-specific breaker for primary script and default breaker for other scripts.	Boolean (false)	Chinese, Japanese, Thai
`nfkcNormalize`	Turns on Unicode NFKC normalization before tokenization. `tokenizerType` must not be `FST` or `SPACELESS_LEXICAL`.	Boolean (false)	All
`query`	Indicates the input will be queries, likely incomplete sentences. If true, tokenizers may change their behavior.	Boolean (false)	All

The following options are described in more detail in Structured Text [16].

Structured Text Options

Option	Description	Type (Default)	Supported Languages
`fragmentBoundaryDetection`	Turn on fragment boundary detection.	Boolean (true)	All
`fragmentBoundaryDelimiters`	Specify the fragment boundary delimiters.	String ("\u0009\u000B\u000C")	All
`maxTokensForShortLine`	The maximum length of a short line.	Integer (6)	All

The following options are described in more detail in Social Media Tokens: Emoji & Emoticons, Hashtags, @Mentions, Email Addresses, URLs [17].

Social Media Token Options

Option	Description	Default	Supported Languages
n/a[a]	Enables emoji tokenization	`true`	All
`emoticons`	Enables emoticon tokenization	`false`	All
`atMentions`	Enables atMention tokenziation	`false`	All
`hashtags`	Enables hashtag tokenization	`false`	All
`emailAddresses`	Enables emailAdress tokenization	`false`	All
`urls`	Enables url tokenization	`false`	All

[a]Emoji tokenization and POS-tagging is always enabled and cannot be disabled.

16.3. Analyzer Options

The following options are described in more detail in Analyzers [20].

General Analyzer Options

Option	Description	Type (Default)	Supported Languages
`analysisCacheSize` `cacheSize`	Maximum number of entries in the analysis cache. Larger values increase throughput, but use extra memory. If zero, caching is off.	Integer (100.000)	All
`caseSensitive`	Indicates whether analyzers produced by the factory are case sensitive. If false, they ignore case distinctions.	Boolean (true)	Czech, Danish, Dutch, English, French, German, Greek, Hebrew, Hungarian, Indonesian, Italian, Malay (Standard), Norwegian, Polish, Portuguese, Russian, Spanish, Swedish, Tagalog
`deliverExtendedTags`	Indicates whether the analyzers should return extended tags with the raw analysis. If true, the extended tags are returned.	Boolean (false)	All
`normalizationDictionaryPaths`	A list of paths to user many-to-one normalization dictionaries, separated by semicolons or the OS-specific path separator.	List of paths	All
`query`	Indicates the input will be queries, likely incomplete sentences. If true, analyzers may change their behavior (e.g. disable disambiguation)	Boolean (false)	All

The following options are described in more detail in Compounds [22].

Compound Options

Option	Description	Type (Default)	Supported Languages
`decomposeCompounds`	Indicates whether to decompose compounds. For Chinese and Japanese, `tokenizerType` must be `SPACELESS_LEXICAL`. If `koreanDecompounding` is enabled but `decomposeCompounds` is disabled, compounds will be decomposed.	Boolean (true)	Chinese, Danish, Dutch, German, Hungarian, Japanese, Korean, Norwegian (Bokmål, Nynorsk), Swedish

Option	Description	Type (Default)	Supported Languages
`compoundComponentSurfaceForms`	Indicates whether to return the surface forms of compound components. When this option is enabled and ADM results are returned, `getText` returns the surface form of a component `Token`, and its lemma can be retrieved using `Token#getAnalyses()` and `MorphoAnalysis#getLemma()`. When this option is enabled and the results are not in ADM format, `getCompoundComponentSurfaceForms` returns the surface forms of a compound word's `Analysis`, and its surface form is not available. This option has no effect when `decomposeCompounds` is set to `false`.	Boolean (false)	Dutch, German, Hungarian

The following options are described in more detail in Disambiguation [23].

Disambiguation Options

Option	Description	Type (Default)	Supported Languages
`disambiguate`	Indicates whether the analyzers should disambiguate the results.	Boolean (true)	Arabic, Chinese, Czech, Dutch, English, French, German, Greek, Hebrew, Hungarian, Italian, Japanese, Korean, Polish, Portuguese, Russian, Spanish
`alternativeEnglishDisambiguation`	Enables faster part of speech disambiguation for English.	Boolean (false)	English
`alternativeGreekDisambiguation`	Enables faster part of speech disambiguation for Greek	Boolean (false)	Greek
`alternativeSpanishDisambiguation`	Enables faster part of speech disambiguation for Spanish.	Boolean (false)	Spanish

The following options are described in more detail in Returning Universal Part-of-Speech (POS) Tags [24].

Universal POS Tag Options

Option	Description	Type (Default)	Supported Languages
`universalPosTags`	Indicates if POS tags should be converted to unversal versions	Boolean (false)	POS tags are defined for Arabic, Chinese, Czech, Dutch, English, French, German, Greek, Hebrew, Hungarian, Italian, Japanese, Korean, Persian, Polish, Portuguese, Russian, Spanish, and Urdu.
`customPosTagsUri`	URI of a POS tag map	URI	POS tags are defined for Arabic, Chinese, Czech, Dutch, English, French, German, Greek, Hebrew, Hungarian, Italian, Japanese, Korean, Persian, Polish, Portuguese, Russian, Spanish, and Urdu.

The following options are described in more detail in Contraction Splitting Rule File Format [26].

Contraction Splitting Options

Option	Description	Type (Default)	Supported Languages
`tokenizeContractions`	Indicates whether to deliver contractions as multiple tokens. If `false`, they are delivered a a single token.	Boolean (false)	All
`customTokenizeContractionRulesUri`	URI of contraction rule file.	URI	All

The following options are only available when using the ADM API.

Annotator Object Options

Option	Description	Type (Default)	Supported Languages
`analyze`	Enables analysis. If false, the annotator will only perform tokenization.	Boolean (`true`)	All
`customPosTagsUri`	URI of a POS tag map file for use by the `universalPosTags` option.	URI (`true`)	Czech, Danish, Dutch, English, French, German, Greek, Hebrew, Hungarian, Italian, Norwegian, Polish, Portuguese, Russian, Spanish, Swedish

16.4. Chinese and Japanese Options

The following options are described in more detail in Chinese and Japanese Lexical Tokenization [27].

Chinese and Japanese Lexical Options

Option	Description	Default value	Supported languages
`breakAtAlphaNumIntraWordPunct`	Indicates whether to consider punctuation between alphanumeric characters as a break. Has no effect when `consistentLatinSegmentation` is `true`.	`false`	Chinese
`consistentLatSegmentation`	Indicates whether to provide consistent segmentation of embedded text not in the primary script. If false, then the setting of `segmentNonJapanese` is ignored.	`true`	Chinese, Japanese
`decomposeCompounds`	Indicates whether to decompose compounds.	`true`	Chinese, Japanese
`deepCompoundDecomposition`	Indicates whether to recursively decompose each token into smaller tokens, if the token is marked in the dictionary as being decomposable. If deep decompounding is enabled, the decomposable tokens will be further decomposed into additional tokens. Has no effect when `decomposeCompounds` is `false`.	`false`	Chinese, Japanese
`favorUserDictionary`	Indicates whether to favor words in the user dictionary during segmentation.	`false`	Chinese, Japanese
`ignoreSeparators`	Indicates whether to ignore whitespace separators when segmenting input text. If `false`, whitespace separators will be treated as morpheme delimiters. Has no effect when `whitespaceTokenization` is `true`.	`true`	Japanese
`ignoreStopwords`	Indicates whether to filter stop words [29] out of the output.	`false`	Chinese, Japanese
`minLengthForScriptChange`	Sets the minimum length of non-native text to be considered for a script change. A script change indicates a boundary between tokens, so the length may influence how a mixed-script string is tokenized. Has no effect when `consistentLatinSegmentation` is `false`.	10	Chinese, Japanese

Option	Description	Default value	Supported languages
`pos`	Indicates whether to add parts of speech to morphological analyses.	`true`	Chinese, Japanese
`segmentNonJapanese`	Indicates whether to segment each run of numbers or Latin letters into its own token, without splitting on medial number/word joiners. Has no effect when `consistentLatinSegmentation` is `true`.	`true`	Japanese
`separateNumbersFromCounters`	Indicates whether to return numbers and counters as separate tokens.	`true`	Japanese
`separatePlaceNameFromSuffix`	Indicates whether to segment place names from their suffixes.	`true`	Japanese
`whiteSpaceIsNumberSep`	Indicates whether to treat whitespace as a number separator. Has no effect when `consistentLatinSegmentation` is `true`.	`true`	Chinese
`whitespaceTokenization`	Indicates whether to treat whitespace as a morpheme delimiter.	`false`	Chinese, Japanese

The following options are described in more detail in Chinese and Japanese Readings [28].

Chinese and Japanese Readings

Option	Description	Default value	Supported languages
`generateAll`	Indicates whether to return all the readings for a token. For characters with multiple readings, all the readings are returned in brackets and separated by semicolons. Has no effect when `readings` is `false`.	`false`	Chinese
`readingByCharacter`	Indicates whether to skip directly to the fallback behavior of `readings` without considering readings for whole words. Has no effect when `readings` is `false`.	`false`	Chinese, Japanese
`readings`	Indicates whether to add readings to morphological analyses. The annotator will try to add readings by whole words. If it cannot, it will concatenate the readings of individual characters.	`false`	Chinese, Japanese
`readingsSeparateSyllables`	Indicates whether to add a separator character between readings when concatenating readings by character. Has no effect when `readings` is `false`.	`false`	Chinese, Japanese
`readingType`	Sets the representation of Chinese readings. Possible values (case-insensitive) are: • `cjktex`: macros for the CJKTeX `pinyin.sty` style • `no_tones`: pinyin without tones • `tone_marks`: pinyin with diacritics over the appropriate vowels • `tone_numbers`: pinyin with a number from 1 to 4 suffixed to each syllable, or no number for neutral tone	`tone_marks`	Chinese
`useVForUDiaeresis`	Indicates whether to use 'v' instead of 'ü' in pinyin readings, a common substitution in environments that lack diacritics. The value is ignored when `readingType` is `cjktex` or `tone_marks`, which always use 'v' and 'ü' respectively. It is probably most useful when `readingType` is `tone_numbers`. Has no effect when `readings` is `false`.	`false`	Chinese

16.5. Hebrew Options

The following options are described in more detail in Hebrew Analyses [29].

Hebrew Options

Option	Description	Type (Default)
`guessHebrewPrefixes`	Splits prefixes off unknown Hebrew words	Boolean (false)
`includeHebrewRoots`	Indicates whether to generate Semitic root forms	Boolean (false)

Hebrew Disambiguation Options

Option	Description	Type (Default)	Supported Languages
`disambiguatorType`	Selects which disambiguator to use for Hebrew.	DisambiguatorType (`PERCEPTRON`)	Hebrew

16.6. Chinese Script Converter Options

The following options are described in more detail in Chinese Script Converter (CSC) [43].

CSC Options

Option	Description	Type (Default)	Supported Languages
`conversionLevel`	Indicates most complex conversion level to use	`CSConversionLevel` (`lexemic`)	Chinese
`language`	The language from which the `CSCAnalyzer` is converting	`LanguageCode`	Chinese, Simplified Chinese, Traditional Chinese
`targetLanguage`	The language to which the `CSCAnalyzer` is converting	`LanguageCode`	Chinese, Simplified Chinese, Traditional Chinese

16.7. Lucene Options

The following options are described in more detail in Using RBL in Apache Lucene [47].

Lucene Filter Options

Option	Description	Type (Default)	Supported Languages
`addLemmaTokens`	Indicates whether the token filter should add the lemmas (if none, the steps) of each surface token to the tokens being returned..	Boolean (true)	All
`addReadings`	Indicates whether the token filter should add the readings of each surface token to the tokens being returned	Boolean (false)	Chinese, Japanese
`identifyContractionComponents`	Indicates whether the token filter should identify contraction components as contraction components rather than as lemmas	Boolean (false)	All
`replaceTokensWithLemmas`	Indicates whether the token filter should replace a surface token with its lemma. Disambiguation must be enabled.	Boolean (false)	All

Lucene User Dictionary Path Options

Option	Description	Type	Supported Languages
`lemDictionaryPath`	A list of paths to user lemma dictionaries.	List of Paths	Chinese, Czech, Danish, Dutch, English, French, German, Greek, Hebrew, Hungarian, Italian, Japanese, Korean, Norwegian, Polish, Portuguese, Russian, Spanish, Swedish, Thai
`segDictionaryPath`	A list of paths to user segmentation dictionaries.	List of Paths	All
`userDefinedDictionaryPath`	A list of paths to user dictionaries.	List of Paths	All
`userDefinedReadingDictionaryPath`	A list of paths to reading dictionaries.	List of Paths	Japanese

17. Alphabetical List of Options