Apache UIMA Dictionary Annotator Documentation

Authors: The Apache UIMA Development Community

Version 2.3.0

Incubation Notice and Disclaimer. Apache UIMA is an effort undergoing incubation at the Apache Software Foundation (ASF). Incubation is required of all newly accepted projects until a further review indicates that the infrastructure, communications, and decision making process have stabilized in a manner consistent with other successful ASF projects. While incubation status is not necessarily a reflection of the completeness or stability of the code, it does indicate that the project has yet to be fully endorsed by the ASF.

Table of Contents

Introduction

The DictionaryAnnotator is an Apache UIMA analysis engine that annotates words based on dictionary entries. For each word in the document text that is available in the dictionary a new annotation is created. The annotator can be configured with one or more independent dictionaries. The dictionaries can easily be created with the dictionary creator command line tooling. For advanced usage of the annotator the matching can also be improved by specifying multi word capabilities, match input type properties and input type filter settings.

Chapter 1. Processing Overview

To use the DictionaryAnnotator at first a dictionary must be created because so far the annotator does not provide any dictionaries. The creation of a dictionary is very simple when using the dictionary creator command line tooling. The tooling takes as input a list words that should be added to the dictionary. The output of the dictionary creator is the created dictionary as XML file and can be used to configure the annotator. For each dictionary additional meta data like the annotation output type for the created annotation can be set. The dictionary and the DictionaryAnnotator can be configured to work with single word dictionary entries like "Apache" or with multi word entries like "Apache UIMA".

After the annotator is configured with the created dictionary the lookup strategy settings must be defined. The dictionary lookup inside the annotator works with tokens. A token is a word or an arbitrary text fragment that is used for the dictionary lookup. If a token match a dictionary entry an annotation is created. The kind of tokens that are used for the lookup can be configured and enhanced with filter capabilities. To improve the dictionary lookup it is recommended that the tokenization for the dictionary entries and the tokenization for the document text is the same. This can be achieved when using the dictionary creator with some advanced settings.

During the annotator processing for each token in the document text that is available in the dictionary a new annotation with the dictionary output type is created. These annotations can be used in a succeeding step to do some further processing.

Chapter 2. Dictionary Creation

To automatically create a dictionary, the DictionaryCreator command line tooling is provied.

2.1. Dictionary Creator

The DictionaryCreator command line tool should be used to create the DictionaryAnnotator dictionaries. The input for the DictionaryCreator is a text file that contains the dictionary entries, one entry per line. The output is the created dictionary as XML file.

The usage below shows all possible command line parameters.

```
java
  -cp uimaj-an-dictionary.jar
  org.apache.uima.annotator.dict_annot.dictionary.impl.DictionaryCreator
  -input <InputFile>
  -encoding <InputFileEncoding>
  -output <OutputFile>
  [-tokenizer <TokenizerPearFile> -tokenType <tokenType>]
  [-separator <separatorChar>]
  [-lang <dictionaryLanguage>]
```

When just using the mandatory settings the input content for the dictionary is tokenized/separated by using the whitespace character. This means that if the line contains a whitespace character as in "Apache UIMA" the dictionary entry is treated as multi word entry where the mutli word consists of the two tokens "Apache" and "UIMA". If the line just contains "DictionaryAnnotator" the dictionary entry in treated as single word entry and has only one token called "DictionaryAnnotator".

A sample XML dictionary file is shown below.

```
<dictionary
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="dictionary.xsd">
   <typeCollection>
      <dictionaryMetaData
         caseNormalization="true"
         multiWordEntries="true"
         multiWordSeparator=" "/>
      <typeDescription>
         <typeName> ADD DICTIONARY OUTPUT TYPE HERE</typeName>
      </typeDescription>
      <entries>
         <entry>
            <key>DictionaryAnnotator</key>
         </entry>
         <entry>
```

```
            <key>Apache UIMA</key>
         </entry>
      </entries>
   </typeCollection>
</dictionary>
```

In addition to the default creation, the DictionaryCreator can be configured with additional parameters.

These are:

- `tokenization <TokenizerPearFile>` - To use an Apache UIMA tokenizer annotator PEAR that tokenize the input instead of the simple whitespace tokenization that is done by default. When using a special tokenizer the `tokenType <tokenType>` parameter must also be set.
- `tokenType <tokenType>` - Specifies the token type to get the tokens created by the tokenizer. These tokens are used to create the single or multi word dictionary entries for each line of the input.
- `lang <languageCode>` - In some cases it is necessary to specify the language for the created dictionary and for the used tokenization.
- `separator <separatorChar>` - If no special tokenizer is used for the tokenization of the input dictionary content, by default the whitespace character is used to tokenizer the content. If another separator character should be used instead, it can be specified by using this parameter.

After the dictionary is created, it is necessary to update the created dictionary with some additional meta data. The most important one that must be set is the `typeName` entry. The `typeName` entry after the creation looks like `<typeName> ADD DICTIONARY OUTPUT TYPE HERE</typeName>` and must be updated with the UIMA type that should be used if the DictionaryAnnotator creates an annotation for a word based on this dictionary. For more details about the other meta data entries of the dictionary, please refer to Section 2.2, “Dictionary XML Format” [4].

2.2. Dictionary XML Format

The Dictionary XML Format is shown with an example below:

```
<dictionary
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="dictionary.xsd">
   <typeCollection>
      <dictionaryMetaData
         caseNormalization="true"
         multiWordEntries="true"
         multiWordSeparator=" "/>
      <languageId>en</languageId>
```

```
      <typeDescription>
         <typeName>org.apache.uima.DictionaryEntry</typeName>
      </typeDescription>
      <entries>
         <entry>
            <key>DictionaryAnnotator</key>
         </entry>
         <entry>
            <key>Apache UIMA</key>
         </entry>
      </entries>
   </typeCollection>
</dictionary>
```

The `<dictionaryMetaData>` element specifies how the dictionary is used inside the DictionaryAnnoator. The attributes for the element are:

- `caseNormalization` - If this parameter is set to `true` all dictionary entries are treated case normalized. This means that the dictionary matching is not case sensitive.
- `multiWordSeparator` - Specifies the multi word separator character that is used in the XML document for multi words. If the DictionaryCreator creates the dictionary files this is by default the "|" character.
- `multiWordEntries` - If this parameter is `true` the dictionary is treated as multi word dictionary. This means that dictionary entries that are separated by the `multiWordSpearator` are treated as multi word entries. So for example "Apache| UIMA" is treated as multi word entry and the document text must have after the tokenization two tokens "Apache" and "UIMA" to match the dictionary entry.

The `<languageId>` element specifies the language of the current dictionary if all entries have the same language. This settings is not mandatory and can also be omitted. content.

The `<typeName>` element specifies the output type that is used if an annotation is created for a dictionary entry.

The `<key>` elements specifies the dictionary entries. For each entry an own `<key>` element is used.

Chapter 3. Annotator Configuration

To use the DictionaryAnnotator it must be configured with at least one dictionary and with the input match type settings - the tokens - that the annotator will use to do the lookup. In addition to these mandatory settings it is possible to define input match type filters to filter the used annotations before they are used for the lookup. The following paragraphs will explain in detail how to configuration is done.

3.1. Dictionary Files

To specify the annotator dictionary files there is a configuration parameter definition in the annotator descriptor that looks like:

```
<configurationParameter>
   <name>DictionaryFiles</name>
   <description>
      list of dictionary files to configure the annotator
   </description>
   <type>String</type>
   <multiValued>true</multiValued>
   <mandatory>true</mandatory>
</configurationParameter>
```

This parameter is mandatory and multi valued. This means that the setting must be available and one or more dictionary files can be specified with the same parameter. A sample setting for two dictionary files can look like:

```
<nameValuePair>
   <name>DictionaryFiles</name>
   <value>
      <array>
         <string>dictionary1.xml</string>
         <string>http://localhost/mydict/dictionary.xml</string>
      </array>
   </value>
</nameValuePair>
```

The specified dictionary file names must be available in the classpath or in the UIMA datapath. Additionally it is possible to specify an HTTP URL to load the dictionary file.

3.2. Input Match Type

The `InputMatchType` parameter defines the annotation type that is used for the dictionary lookup. All annotations of type `InputMatchType` are used for the lookup in the dictionary. In most cases this type should be the output type of the tokenizer annotator component. If the dictionary was created by using the DictionaryCreator configured with a tokenizer, it is recommended that the same tokenizer is also used in the annotator flow. Beyond

that the `InputMatchType` should be the same as the tokenType used for the dictionary creation.

The parameter that defines the input match type is:

```
<configurationParameter>
   <name>InputMatchType</name>
   <description></description>
   <type>String</type>
   <multiValued>false</multiValued>
   <mandatory>true</mandatory>
</configurationParameter>
```

The parameter setting is mandatory and single valued. A sample setting for the `InputMatchType` looks like:

```
<nameValuePair>
   <name>InputMatchType</name>
   <value>
      <string>org.apache.uima.TokenAnnotation</string>
   </value>
</nameValuePair>
```

3.2.1. Input Match Type Feature Path

In some special cases it may be necessary to use a feature value or a featurePath value of the `InputMatchType` for the dictionary lookup. In that case not the covered text of the `InputMatchType` annotation is used for the lookup but the specified feature or featurePath value.

To define a feature or featurePath that is used for the lookup the following parameter must be used:

```
<configurationParameter>
   <name>InputMatchFeaturePath</name>
   <description></description>
   <type>String</type>
   <multiValued>false</multiValued>
   <mandatory>false</mandatory>
</configurationParameter>
```

The parameter is not mandatory, it is just an optional addition. But if the parameter is used, the defined feature or featurePath must be valid for the `InputMatchType`. A sample configuration with a feature called `baseFormToken` is shown below:

```
<nameValuePair>
   <name>InputMatchFeaturePath</name>
   <value>
      <string>baseFormToken</string>
   </value>
```

```
</nameValuePair>
```

If a featurePath is specified the path separator for the feature is "/".

3.3. Input Match Type Filters

If not all `InputMatchType` annotations should be used for the dictionary lookup it is possible to define filters to filter the used annotations. To define a filter three settings are necessary. The first one is the `InputMatchFilterFeaturePath` that specifies the feature or featurePath that should be used for the filtering. The second parameter is the `FilterConditionOperator` that defines the filter condition operator. The last parameter is `FilterConditionValue` that defines the condition value for the comparison.

The parameter definition for all three parameters looks like:

```
<configurationParameter>
   <name>InputMatchFilterFeaturePath</name>
   <description></description>
   <type>String</type>
   <multiValued>false</multiValued>
   <mandatory>false</mandatory>
</configurationParameter>

<configurationParameter>
   <name>FilterConditionOperator</name>
   <description></description>
   <type>String</type>
   <multiValued>false</multiValued>
   <mandatory>false</mandatory>
</configurationParameter>

<configurationParameter>
   <name>FilterConditionValue</name>
   <description></description>
   <type>String</type>
   <multiValued>false</multiValued>
   <mandatory>false</mandatory>
</configurationParameter>
```

For the `InputMatchFilterFeaturePath` the same rules applies as for the `InputMatchFeaturePath`. The specified feature or featurePath must be valid for the `InputMatchType` definition. If a featurePath is specified, the features are separated by "/".

The value for the `FilterConditionOperator` can be one of:

- `NULL` - `InputMatchFilterFeaturePath` value must be NULL. No `FilterConditionValue` must be specified.
- `NOT_NULL` - `InputMatchFilterFeaturePath` value must be set and is not NULL. No `FilterConditionValue` must be specified.

- `EQUALS` - `InputMatchFilterFeaturePath` value must be equal to the `FilterConditionValue`.
- `NOT_EQUALS` - `InputMatchFilterFeaturePath` value is not equal to the `FilterConditionValue`
- `LESS` - `InputMatchFilterFeaturePath` value is less than the `FilterConditionValue`
- `LESS_EQ` - `InputMatchFilterFeaturePath` value is less or equal to the `FilterConditionValue`
- `GREATER` - `InputMatchFilterFeaturePath` value is greater than the `FilterConditionValue`
- `GREATER_EQ` - `InputMatchFilterFeaturePath` value is greater or equal to the `FilterConditionValue`

A sample configuration for a filter that only use noun tokens for the dictionary lookup is shown below:

```
<nameValuePair>
   <name>InputMatchFilterFeaturePath</name>
   <value>
      <string>partOfSpeach</string>
   </value>
</nameValuePair>

<nameValuePair>
   <name>FilterConditionOperator</name>
   <value>
      <string>EQUALS</string>
   </value>
</nameValuePair>

<nameValuePair>
   <name>FilterConditionValue</name>
   <value>
      <string>noun</string>
   </value>
</nameValuePair>
```