TreetaggerModules

- the manual -

Foreword

The OSGi-bundle TreetaggerModules contains two Pepper-modules: the TreetaggerImporter to map a Treetagger model to a Salt model and the TreetaggerExporter to map a Salt model to a Treetagger model. The project TreetaggerModules further provides a module to tokenize textual data named Tokenizer in the same as the tool TreeTagger does.

Inhaltsverzeichnis

Pepper Workflow Description

To use one of the TreetaggerModules you can declare them inside the Pepper workflow description inside the part of the import or the export phase. The TreetagerImporter can be declared via its name:

```
<importerParams moduleName="TreetaggerImporter" sourcePath="..."/>
```

or via the format description of the corpus to import:

```
<importerParams     formatName="Treetagger"
                    formatVersion="1.0"
                    sourcePath="..."/>
```

To use the Tokenizer module in your workflow, you can adress the module with the following instructions:

```
<manipulatorParams  moduleName="Tokenizer"/>
```

The as for the TreetaggerImporter same holds for the TreetaggerExporter:

```
<exporterParams moduleName="TreetaggerExporter" sourcePath="..."/>
```

or via the format description of the corpus to import:

```
<exporterParams     formatName="Treetagger"
                    formatVersion="1.0"
                    sourcePath="..."/>
```

TreetaggerImporter

Input File

An input file for the TreetaggerImporter contains one tab separated row per token. The first column is mandatory and contains the token´s form. Any further column is optional and can be declared in the properties file. Each column is required to have a distinct name. By default, i.e. when there are no column declarations in the properties file, the second and third column are considered the part-of-speech annotation and the lemma annotation respectively. Note that this default is overridden if there is any declaration of columns in the properties file.

It is possible to use SGML tags to mark spans of tokens. Invalid SGML elements (e.g. missing opening or closing tag) will be ignored.

A whole treetagger document may be marked by a surrounding SGML element, which is required to have a certain name. This name is definable in the properties file and defaults to "meta".

The expected input file encoding defaults to "UTF-8" and is also definable in the properties file. Any input file´s name is required to end on ".tab" or ".tt".

```
<meta someDocumentAttribute="someDocumentValue">
<someSpan someSpanAttribute="someSpanValue">
TOKEN_1      POS_1 LEMMA_1
TOKEN_2      POS_2 LEMMA_2
</someSpan>
<someMoreSpan someMoreSpanAttribute="someMoreSpanValue">
TOKEN_3      POS_3 LEMMA_3
TOKEN_4      POS_4 LEMMA_4
TOKEN_5      POS_5 LEMMA_5
</someMoreSpan>
</meta>
```

Example 1: sample input file content with default column settings

Creating Treetagger Representation

The file´s content is converted to a Treetagger document containing a list of tokens and spans. If there is a document marking SGML element in the input file, an annotation for each of it´s attribute-value-pairs is added to the treetagger document. For all other SGML elements, spans are created and annotations according to the element´s attribute-value-pairs are added. For each data row, a token is created. The token´s text attribute is set to the first column´s content. For each additional column, an annotation is created and added to the token. There are three different types of annotations: The POSAnnotation, used for part-of-speech annotations, the LemmaAnnotation, used for lemma annotations, and the AnyAnnotation, used for all user-defined annotations.

If the default settings for columns are used, a POSAnnotation for the second column´s content and a LemmaAnnotation for the third column´s content are created and added to the token.

If user-defined settings for columns are used, a POSAnnotation will be created for a column named "pos", a LemmaAnnotation will be created for a column named "lemma", and an AnyAnnotation will be created for each other column. Note that all names have to be distinct, and that a token can only have one POSAnnotation and one LemmaAnnotation, but any number of AnyAnnotations.

Mapping From Treetagger To Salt

Document Annotations

When converting a Treetagger document to Salt, a SDocument and it´s proper SDocumentGraph will be created. If there is a meta tag for the whole document, all it´s attributes will be added to the SDocument as SMetaAnnotations.

Tokens

Each token will be mapped to an SToken, which is added to the SDocumentGraph. If a token has a POSAnnotation, the SToken will get a corresponding SPOSAnnotation. If a token has a LemmaAnnotation, the SToken will get a corresponding SLemmaAnnotation.

All the token´s forms, separated by space characters, will be contained in the SText attribute of one STextualDS, which also is also added to the SDocumentGraph.

Spans

Each Treetagger span is mapped to a SSpan, which is added to the SDocumentGraph. The SSpan´s name is set to the treetagger span´s name. The annotations on the treetagger span are mapped to SAnnotations and added to the SSpan. For each span, a SSpanningRelation between the span and all contained tokens is created. The SSpanningRelation is added to the SDocumentGraph as well.

There are two switches concerning the annotations on the treetagger spans. The one of them concerns spans without any annotations and will add a SAnnotation, having the span´s name as name and as value, to the SSpan. The other one will do the same, but applies to all spans, regardless of the presence of annotations.

Tokenizer

The Tokenizer module creates for each STextualDS object a tokenization in the same way as the tool Treetagger does (see: http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/).

Mapping

That means for each token a `SToken` object will be created and referenced via a `STextualRelation` object to the corresponding `STextualDS` object. For tokenization a list of abbreviations corresponding to the current used language is used. It is possible to manipulate or extend the lists, given in the resources folder of this project under tokenizer/abbreviation. The current supported languages are english, french, german and italian.

Properties

The Tokenizer does not automatically identify the source language, therefore you can create a property file, which looks like this:

```
treetagger.tokenizer.language=en
```

, where en stands for english, fr for french, de for german and it for italian. This property file, lets assume its name is tokenizer.prop can be set to the workflow via using the specialparameter-attribute:

```
<moduleParams  moduleName="Tokenizer"
               specialParams="file:/./tokenizer.prop"/>
```

If no property file is given, the Tokenizer assumes, that the language is english.

TreetaggerExporter

Mapping

Document Annotations

When converting from Salt to Treetagger, a Treetagger document will be created, and all SMetaAnnotations of the SDocument will be mapped to Annotations of that document.

Tokens

Each SToken will be mapped to a Treetagger token. The token´s text comes from the STextualDS of the SDocumentGraph. If there is a SPOSAnnotation for the SToken, or if there is a SAnnotation named "pos", "part-of-speech" or "partofspeech" (all case insensitive), it will be mapped to the POSAannotation of the token. If there is a SLemmaAnnotation for the SToken, or if there is a SAnnotation named "lemma", "lemmatisation", "lemmatization" or "lemmata" (all case insensitive), it will be mapped to the LemmaAnnotation of the token. All other SAnnotations will be mapped to an "AnyAnnotation" of the treetagger token.

Spans

The SSpans from all SSpanningRelations in the SDocumentGraph will be mapped to spans of the treetagger document. There is a switch in the properties file for the processing of SSpans with generic names ("sSpan", followed by a number). If this switch is set "true", these names will be replaced by the name of the first annotation found on the span.

Output File

An output file from the TreetaggerExporter always contains a SGML element marking the whole document. The tag for the element defaults to "meta". This tag is definable in the properties file. All the annotations on the treetagger document will be added to the SGML element as attribute-value-pairs.

The output file contains one tab separated row per token. The first column contains the token´s form. If there is a POSAnnotation for the token, the second column contains it´s value, else it remains empty. If there is a LemmaAnnotation for the token, the third column contains it´s value, else it remains empty. The output of AnyAnnotations can be set in the properties file. If it is set "true", a column for each distinctly named AnyAnnotation will appears in the output file, sorted alphabetically by the AnyAnnotations´ names. Note that these names do not appear in the output file. However, the names and the order of the columns will be logged on the info-level of the conversion process.

All Spans will appear as SGML elements in the ouput file. The SGML element´s name is the Spans name, and all it´s annotations will be added to the element as attribute-value-pairs. In the properties file, the renaming of generically named Spans

The default encoding for output files is "UTF-8". This also is definable in the properties file. All output files´ names end on ".tt".

Properties

File

A properties file is used to set certain options for the conversion process. Importer and Exporter use the same properties file. Since all options have default settings, this file is not mandatory.

A properties file has to be given as a special parameter and needs to be conform to a java poperty file.

To use a properties file, you have to expand the Pepper workflow description via the xml attribute `specialParams` as shown here:

```
<importerParams ... specialParams="PROP_FILE"/>
```

or for the exporter:

```
<exporterParams ... specialParams="PROP_FILE"/>
```

Note that the file has to be conform to the URI notation. On a local file system, this means adding the prefix `file:/` to the filepath. The declaration of a properties file is the same when using the format description for identifying the Pepper module.

Attributes

This is a list of all attributes, wherein the given values are the default settings.

```
treetagger.input.fileEncoding=UTF-8
```

States the encoding of the input file(s).

```
treetagger.input.metaTag=meta
```

States the meta tag used to mark the treetagger document in the input file(s).

```
treetagger.output.fileEncoding=UTF-8
```

Sets the encoding of the output file(s).

```
treetagger.output.metaTag=meta
```

Sets the meta tag used to mark the treetagger document in the output file(s).

```
treetagger.output.exportAnyAnnotation=false
```

If set `true`, each AnyAnnotation of tokens will appear in the output file.

`treetagger.input.columnX=NAME` *where X is a number and NAME a name*

Sets the names for input columns, and thus for the corresponding annotations of token. "pos" will result in a SPOSAnnotation, "lemma" will result in a SLemmaAnnotation. All other names will result in a SAnnotation. An arbitrary number of such entries is possible. Ensure that consecutive numbers are used, beginning at 1. Mind that column 0 is reserved for the word form. If no such entry exists, column 1 and 2 are interpreted as "pos" and "lemma" respectively.

`treetagger.input.annotateUnannotatedSpans=false`

If set `true`, this switch will cause the module to annotate all spans without attributes with their name as attribute and value, i.e.

```
<MyTag>
```

will be treated as

```
<MyTag mytag="mytag">
```

`treetagger.input.annotateAllSpansWithSpanName=false`

If set `true`, this switch will cause the module to annotate all spans with their name as attribute and value, i.e.

```
<MyTag attribute="value">
```

will be treated as

```
<MyTag attribute="value" mytag="mytag">
```

`treetagger.output.replaceGenericSpanNames=false`

If set `true`, generic span names like "sSpan123" will be replaced with the first annotation of the span found. If the span has no annotations, the generic name will not be replaced.

`treetagger.output.flatten=false`

If set `true`, the output directory structure is flat: all documents are put in the output root directory.