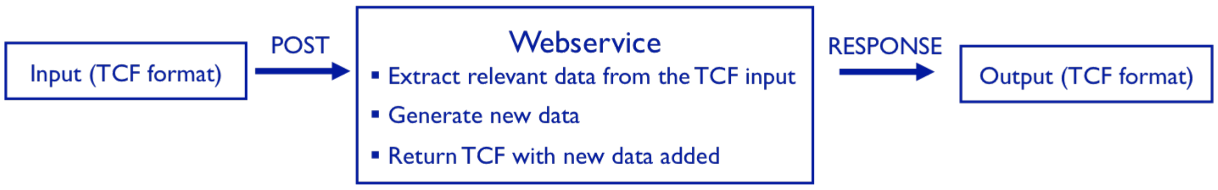
Tutorial for Creating WebLicht Webservices

Marie Hinrichs, Thomas Zastrow
University of Tübingen

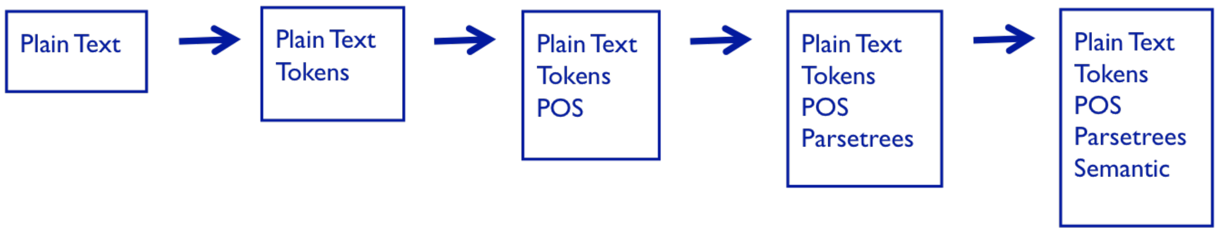
1. WebLicht Overview

Webservices for the WebLicht toolchain are implemented as RESTful webservices. The input is sent via the POST method of the HTTP protocol to the service. The output of the webservice is the response to that POST event.



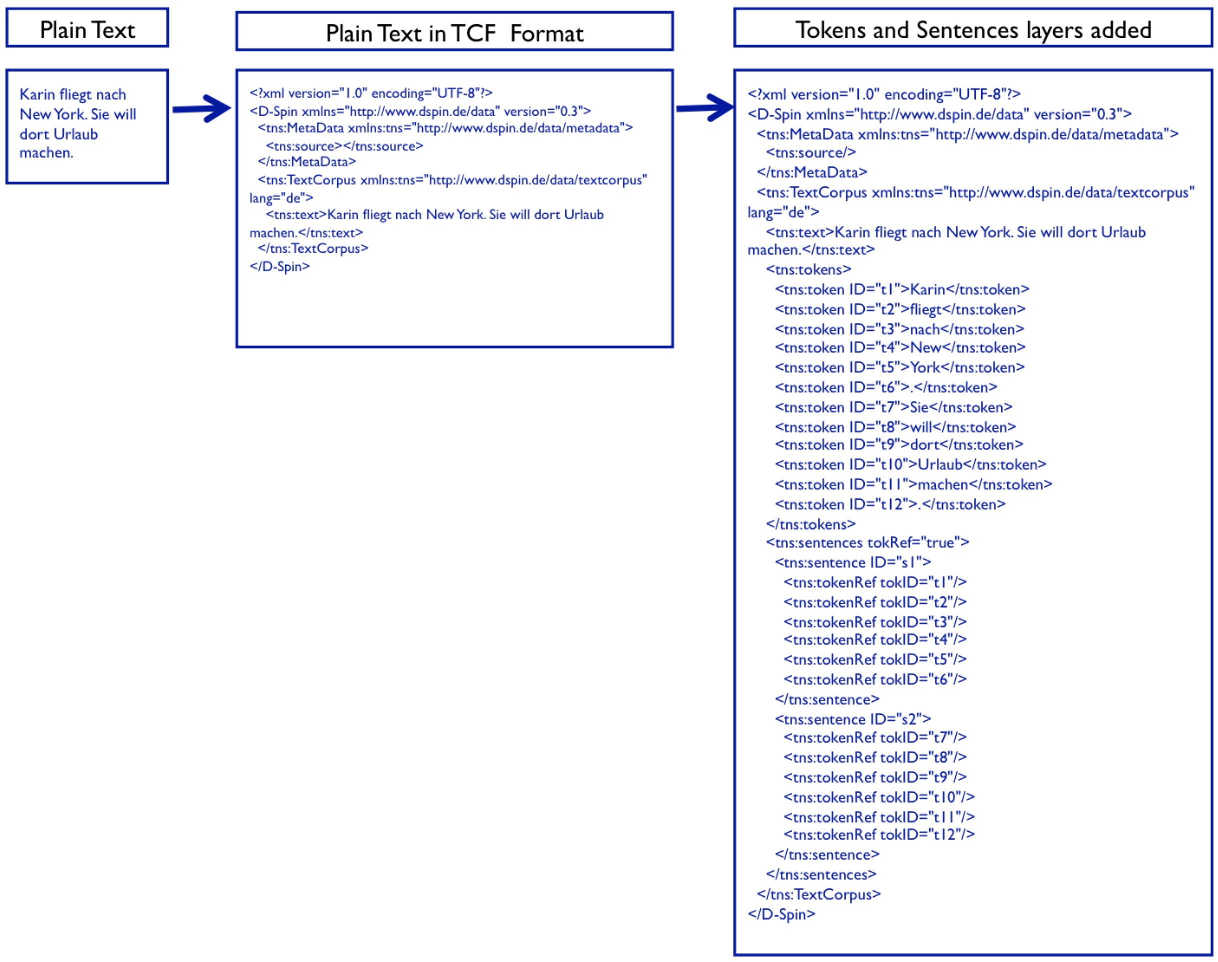
2. The TCF Format

The TCF format is a simple stand-off format for linguistic annotated textcorpora. In contrast to some other formats (for example PAULA or MAF), TCF stores all linguistic layers in one file. That means that during the chaining process, the file grows. A typical chain might look like this:


Plain Text
Plain Text Tokens
Plain Text Tokens POS
Plain Text Tokens POS Parsetrees
Plain Text Tokens POS Parsetrees Semantic

WebLicht is not restricted to the use of the TCF format, although most of the services currently registered do use it. The advantage is that services using the same format can be easily chained together. This tutorial assumes use of the TCF format.

Important: every webservice can add an arbitrary number of layers to the TCF file, but no webservice should change or remove an already existing layer! This can be seen in the following sample input and output of a typical chain, where the output of one service is used as input to the next.



```
Karin fliegt nach
New York. Sie will
dort Urlaub
machen.
```

```
<?xml version="1.0" encoding="UTF-8"?>
<D-Spin xmlns="http://www.dspin.de/data" version="0.3">
  <tns:MetaData xmlns:tns="http://www.dspin.de/data/metadata">
    <tns:source></tns:source>
  </tns:MetaData>
  <tns:TextCorpus xmlns:tns="http://www.dspin.de/data/textcorpus"
lang="de">
    <tns:text>Karin fliegt nach New York. Sie will dort Urlaub
machen.</tns:text>
  </tns:TextCorpus>
</D-Spin>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<D-Spin xmlns="http://www.dspin.de/data" version="0.3">
  <tns:MetaData xmlns:tns="http://www.dspin.de/data/metadata">
    <tns:source/>
  </tns:MetaData>
  <tns:TextCorpus xmlns:tns="http://www.dspin.de/data/textcorpus"
lang="de">
    <tns:text>Karin fliegt nach New York. Sie will dort Urlaub
machen.</tns:text>
    <tns:tokens>
      <tns:token ID="t1">Karin</tns:token>
      <tns:token ID="t2">fliegt</tns:token>
      <tns:token ID="t3">nach</tns:token>
      <tns:token ID="t4">New</tns:token>
      <tns:token ID="t5">York</tns:token>
      <tns:token ID="t6">.</tns:token>
      <tns:token ID="t7">Sie</tns:token>
      <tns:token ID="t8">will</tns:token>
      <tns:token ID="t9">dort</tns:token>
      <tns:token ID="t10">Urlaub</tns:token>
      <tns:token ID="t11">machen</tns:token>
      <tns:token ID="t12">.</tns:token>
    </tns:tokens>
    <tns:sentences tokRef="true">
      <tns:sentence ID="s1">
        <tns:tokenRef tokID="t1"/>
        <tns:tokenRef tokID="t2"/>
        <tns:tokenRef tokID="t3"/>
        <tns:tokenRef tokID="t4"/>
        <tns:tokenRef tokID="t5"/>
        <tns:tokenRef tokID="t6"/>
      </tns:sentence>
      <tns:sentence ID="s2">
        <tns:tokenRef tokID="t7"/>
        <tns:tokenRef tokID="t8"/>
        <tns:tokenRef tokID="t9"/>
        <tns:tokenRef tokID="t10"/>
        <tns:tokenRef tokID="t11"/>
        <tns:tokenRef tokID="t12"/>
      </tns:sentence>
    </tns:sentences>
  </tns:TextCorpus>
</D-Spin>
```

3. Example Service

The sample service included in this tutorial analyzes the occurrences of lemmas and tokens in the input document. A TCF layer called “uniqueLemmas” is created, which contains one entry per lemma, each entry in turn containing the corresponding tokens and the number of times which that token appears.

In order to proceed with the sample service included in this tutorial, you will need:

- A Java servlet container, such as Tomcat, Glassfish, or JBoss.
- An IDE, such as NetBeans or Eclipse. The project was developed with NetBeans, but it is possible to import it into Eclipse.
- A command-line tool such as `wget` or `curl` for testing

Included with this tutorial you will find:

- WebLichtSampleService.war: can be deployed to a Java servlet container
- sampleInput.xml: sample input for the service
- WebLichtSampleService.zip: unzip and open in IDE to view source code and build WAR file
- dspin_0_3.xsd, dspin_textcorpus_0_3.xsd: TCF schemas

Now we can take a closer look at the example. The sample service accepts as input a TCF document containing tokens and lemmas (possibly other layers also). From the tokens and lemmas layers, we will produce a map with the unique lemmas as keys and a list of respective tokens as values. The input document will be parsed with a StAX parser, where the tokens and lemmas will be extracted. During parsing, the document will be written to the response stream to preserve the original document. Before writing the TextCorpus close tag, we will insert our new layer. Finally, the remaining portion of the document will be written to the output as-is.

See the following pages for an example input to our service and the generated output. Notice that we only add information, but do not change or delete anything in the input.

Note that this service is not necessarily a good candidate for inclusion in WebLicht, since it only summarizes information already in the input. However, it will suffice as an example of how to create a service.

This sample service was developed with Java using the NetBeans IDE, and can be deployed in any Java servlet container, such as Tomcat, Glassfish, or JBoss.

Sample Input for the LemmaSummarizer Service:

```
<?xml version="1.0" encoding="UTF-8"?>
<D-Spin xmlns="http://www.dspin.de/data" version="0.3">
  <tns:MetaData xmlns:tns="http://www.dspin.de/data/metadata">
    <tns:source/>
  </tns:MetaData>
  <tns:TextCorpus xmlns:tns="http://www.dspin.de/data/textcorpus" lang="de">
    <tns:text>Karin aß Pizza. Max hat Pizza gegessen. Sie essen Pizza.</tns:text>
    <tns:tokens>
      <tns:token ID="t1">Karin</tns:token>
      <tns:token ID="t2">aß</tns:token>
      <tns:token ID="t3">Pizza</tns:token>
      <tns:token ID="t4">.</tns:token>
      <tns:token ID="t5">Max</tns:token>
      <tns:token ID="t6">hat</tns:token>
      <tns:token ID="t7">Pizza</tns:token>
      <tns:token ID="t8">gegessen</tns:token>
      <tns:token ID="t9">.</tns:token>
      <tns:token ID="t10">Sie</tns:token>
      <tns:token ID="t11">essen</tns:token>
      <tns:token ID="t12">Pizza</tns:token>
      <tns:token ID="t13">.</tns:token>
    </tns:tokens>
    <tns:sentences tokRef="true">
      <tns:sentence ID="s1">
        <tns:tokenRef tokID="t1"/>
        <tns:tokenRef tokID="t2"/>
        <tns:tokenRef tokID="t3"/>
        <tns:tokenRef tokID="t4"/>
      </tns:sentence>
      <tns:sentence ID="s2">
        <tns:tokenRef tokID="t5"/>
        <tns:tokenRef tokID="t6"/>
        <tns:tokenRef tokID="t7"/>
        <tns:tokenRef tokID="t8"/>
        <tns:tokenRef tokID="t9"/>
      </tns:sentence>
      <tns:sentence ID="s3">
        <tns:tokenRef tokID="t10"/>
        <tns:tokenRef tokID="t11"/>
        <tns:tokenRef tokID="t12"/>
        <tns:tokenRef tokID="t13"/>
      </tns:sentence>
    </tns:sentences>
    <tns:POStags tagset="STTS">
      <tns:tag tokID="t1">NE</tns:tag>
      <tns:tag tokID="t2">VVFIN</tns:tag>
      <tns:tag tokID="t3">NN</tns:tag>
      <tns:tag tokID="t4">$.</tns:tag>
      <tns:tag tokID="t5">NE</tns:tag>
      <tns:tag tokID="t6">VAFIN</tns:tag>
      <tns:tag tokID="t7">NN</tns:tag>
      <tns:tag tokID="t8">VVPP</tns:tag>
      <tns:tag tokID="t9">$.</tns:tag>
      <tns:tag tokID="t10">PPER</tns:tag>
      <tns:tag tokID="t11">VVFIN</tns:tag>
      <tns:tag tokID="t12">NN</tns:tag>
      <tns:tag tokID="t13">$.</tns:tag>
    </tns:POStags>
    <tns:lemmas>
      <tns:lemma tokID="t1">Karin</tns:lemma>
      <tns:lemma tokID="t2">essen</tns:lemma>
      <tns:lemma tokID="t3">Pizza</tns:lemma>
      <tns:lemma tokID="t4">.</tns:lemma>
      <tns:lemma tokID="t5">Max</tns:lemma>
      <tns:lemma tokID="t6">haben</tns:lemma>
      <tns:lemma tokID="t7">Pizza</tns:lemma>
      <tns:lemma tokID="t8">essen</tns:lemma>
      <tns:lemma tokID="t9">.</tns:lemma>
      <tns:lemma tokID="t10">Sie|sie|sie</tns:lemma>
      <tns:lemma tokID="t11">essen</tns:lemma>
      <tns:lemma tokID="t12">Pizza</tns:lemma>
      <tns:lemma tokID="t13">.</tns:lemma>
    </tns:lemmas>
  </tns:TextCorpus>
</D-Spin>
```

Sample Output of the LemmaSummarizer Service:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<D-Spin version="0.3" xmlns="http://www.dspin.de/data">
  <tns:MetaData xmlns:tns="http://www.dspin.de/data/metadata">
    <tns:source></tns:source>
  </tns:MetaData>
  <tns:TextCorpus lang="de"
xmlns:tns="http://www.dspin.de/data/textcorpus">
    <tns:text>Karin aß Pizza. Max hat Pizza gegessen. Sie essen
Pizza.</tns:text>
    <tns:tokens>
      <tns:token ID="t1">Karin</tns:token>
      <tns:token ID="t2">aß</tns:token>
      <tns:token ID="t3">Pizza</tns:token>
      <tns:token ID="t4">.</tns:token>
      <tns:token ID="t5">Max</tns:token>
      <tns:token ID="t6">hat</tns:token>
      <tns:token ID="t7">Pizza</tns:token>
      <tns:token ID="t8">gegessen</tns:token>
      <tns:token ID="t9">.</tns:token>
      <tns:token ID="t10">Sie</tns:token>
      <tns:token ID="t11">essen</tns:token>
      <tns:token ID="t12">Pizza</tns:token>
      <tns:token ID="t13">.</tns:token>
    </tns:tokens>
    <tns:sentences tokRef="true">
      <tns:sentence ID="s1">
        <tns:tokenRef tokID="t1"></tns:tokenRef>
        <tns:tokenRef tokID="t2"></tns:tokenRef>
        <tns:tokenRef tokID="t3"></tns:tokenRef>
        <tns:tokenRef tokID="t4"></tns:tokenRef>
      </tns:sentence>
      <tns:sentence ID="s2">
        <tns:tokenRef tokID="t5"></tns:tokenRef>
        <tns:tokenRef tokID="t6"></tns:tokenRef>
        <tns:tokenRef tokID="t7"></tns:tokenRef>
        <tns:tokenRef tokID="t8"></tns:tokenRef>
        <tns:tokenRef tokID="t9"></tns:tokenRef>
      </tns:sentence>
      <tns:sentence ID="s3">
        <tns:tokenRef tokID="t10"></tns:tokenRef>
        <tns:tokenRef tokID="t11"></tns:tokenRef>
        <tns:tokenRef tokID="t12"></tns:tokenRef>
        <tns:tokenRef tokID="t13"></tns:tokenRef>
      </tns:sentence>
    </tns:sentences>
    <tns:POStags tagset="STTS">
      <tns:tag tokID="t1">NE</tns:tag>
      <tns:tag tokID="t2">VVFIN</tns:tag>
      <tns:tag tokID="t3">NN</tns:tag>
      <tns:tag tokID="t4">$.</tns:tag>
      <tns:tag tokID="t5">NE</tns:tag>
      <tns:tag tokID="t6">VAFIN</tns:tag>
      <tns:tag tokID="t7">NN</tns:tag>
      <tns:tag tokID="t8">VVPP</tns:tag>
      <tns:tag tokID="t9">$.</tns:tag>
      <tns:tag tokID="t10">PPER</tns:tag>
      <tns:tag tokID="t11">VVFIN</tns:tag>
      <tns:tag tokID="t12">NN</tns:tag>
      <tns:tag tokID="t13">$.</tns:tag>
    </tns:POStags>
    <tns:lemmas>
      <tns:lemma tokID="t1">Karin</tns:lemma>
      <tns:lemma tokID="t2">essen</tns:lemma>
      <tns:lemma tokID="t3">Pizza</tns:lemma>
      <tns:lemma tokID="t4">.</tns:lemma>
      <tns:lemma tokID="t5">Max</tns:lemma>
      <tns:lemma tokID="t6">haben</tns:lemma>
      <tns:lemma tokID="t7">Pizza</tns:lemma>
      <tns:lemma tokID="t8">essen</tns:lemma>
      <tns:lemma tokID="t9">.</tns:lemma>
      <tns:lemma tokID="t10">Sie|sie|sie</tns:lemma>
      <tns:lemma tokID="t11">essen</tns:lemma>
      <tns:lemma tokID="t12">Pizza</tns:lemma>
      <tns:lemma tokID="t13">.</tns:lemma>
    </tns:lemmas>
    <tns:uniqueLemmas>
      <tns:uniqueLemma lemma="essen">
        <tns:count token="aß">1</tns:count>
        <tns:count token="essen">1</tns:count>
        <tns:count token="gegessen">1</tns:count>
      </tns:uniqueLemma>
      <tns:uniqueLemma lemma="Karin">
        <tns:count token="Karin">1</tns:count>
      </tns:uniqueLemma>
      <tns:uniqueLemma lemma="Max">
        <tns:count token="Max">1</tns:count>
      </tns:uniqueLemma>
      <tns:uniqueLemma lemma="Sie|sie|sie">
        <tns:count token="Sie">1</tns:count>
      </tns:uniqueLemma>
      <tns:uniqueLemma lemma=".">
        <tns:count token=".">3</tns:count>
      </tns:uniqueLemma>
      <tns:uniqueLemma lemma="haben">
        <tns:count token="hat">1</tns:count>
      </tns:uniqueLemma>
      <tns:uniqueLemma lemma="Pizza">
        <tns:count token="Pizza">3</tns:count>
      </tns:uniqueLemma>
    </tns:uniqueLemmas>
  </tns:TextCorpus>
</D-Spin>
```

3.1. WAR-files, Servlets, and URLs

Deploy WebLichtSampleService.war and go to: http://localhost:8080/WebLichtSampleService in your browser, replacing localhost:8080 with the actual host:port if necessary.

You should end up at the index.jsp page, which is entered as a welcome-file in WEB-INF/web.xml. No welcome page is needed for a WebLicht service because it is not intended to be user-interactive. Normally index.jsp can be safely deleted. However, our sample service's welcome page contains instructions on how to call the servlet using `wget`, so in this case it should not be deleted.

The name of the war-file (web archive) is WebLichtSampleService.war, and WebLichtSampleService is the context, so it is the first part of the path after the host name. The context name can be modified in META-INF/context.xml.

A servlet within the WebLichtSampleService context called lemmaServlet03 is in a source package called servlets. The mappings between servlet names and their corresponding class names can be specified in WEB-INF/web.xml. Our sample service is available at:

http://localhost:8080/WebLichtSampleService/lemmaServlet03

You can test the service with the sample input file from the command line:

```
wget --post-file=sampleInput.xml http://localhost:8088/WebLichtSampleService/lemmaServlet03 -O lemmaServlet03Out.xml
```

The output of lemmaServlet03 will be stored in the file lemmaServlet03Out.xml.

3.2. A Look at the Servlet Code

A servlet is just a Java class that extends HttpServlet, which contains `doGet` and `doPost` methods[1] that can be overridden to perform GET and POST requests, respectively. Each of these methods has a request and a response object as parameters.

The `doPost` method first sets the response's contentType to „text/xml" with „UTF-8" character encoding, which is TCF standard. Next, the document that has been posted is read from the request's input stream into a temporary file. An input stream from this temporary file, along with the response's output stream is then sent to the `LemmaSummarizer` object, which does the real work of the service. Finally, the temporary file is deleted and the response's output stream is closed. If any errors are encountered, an error is sent to the response with an appropriate message. Note that although it may seem unnecessary to store the input document in a file before processing, it is more reliable than reading directly from the request's input stream.

Our `doGet` method simply sends an error to the response because we are expecting data to be sent with the POST method.

[1] This pertains to Java EE 5. There are some simplifications in the upcoming Java EE 6.

3.3. A Look at the LemmaSummarizer

The `LemmaSummarizer` object does all of the real work of the service. It simultaneously parses the input stream (posted document) and writes all necessary data to the response's output stream. All elements, including their attributes, that are read from the input document are copied to the response. All information needed to perform the service's task are saved during parsing. After all of the TextCorpus layers have been parsed and copied to the response stream, but before the TextCorpus end tag is written, the new layer is calculated an written to the response. Then the remaining end tags are written.

The only public method in `LemmaSummarizer` is `summarizeLemmas`. It creates a parser for the input stream, a writer for the output stream, and a map to store Token objects with the token id as key. When the TextCorpus element is encountered, it is processed by the processTextCorpus method.

The `processTextCorpus` method parses and writes the TextCorpus layer, processing the lemmas layer with the `processLemmas` method and the tokens layer with the `processTokens` method. These two methods fill in the tokenIdMap, so that by the time the TextCorpus end element is reached, the map is filled with Token objects. For our sample input, the map looks like this:

t1	lemma: Karin token: Karin
t2	lemma: essen token: aß
t3	lemma: Pizza token: Pizza
t4	lemma: . token: .
t5	lemma: Max token: Max
t6	lemma: haben token: hat
t7	lemma: Pizza token: Pizza
t8	lemma: essen token: gegessen
t9	lemma: . token: .
t10	lemma: Sie|sie|sie token: Sie
t11	lemma: essen token: essen
t12	lemma: Pizza token: Pizza
t13	lemma: . token: .

Before the end TextCorpus element is written to the response, the new layer is added in the `writeLemmaSummary` method. This method manipulates the map above and creates a new map with the lemmas as keys and a list of TokenCount objects as values. The result for our sample data is:

essen	aß	essen	gegessen
	1	1	1
Karin	Karin		
	1		
Max	Max		
	1		
Sie|sie|sie	Sie		
	1		
.	.		
	3		
haben	hat		
	1		
Pizza	Pizza		
	3		

It can be seen from this map that the input document contains the lemma "essen" a total of three times: once as the token "aß", once as the token "essen" and once as the token "gegessen". The lemma "Pizza" appears three times as token "Pizza".

Once this map has been produced, it is simply iterated and written to the response as well-formed xml.

3.4. Why StAX?

If you are familiar with DOM parsing, you may ask why a StAX parser is being used in this sample service tutorial. DOM parsers hold an entire document in memory, so elements can be easily accessed and added. It is perfectly suited to our needs, except for its one major drawback – it cannot efficiently process large documents, and may even cause an OutOfMemoryException. A StAX parser on the other hand, is very fast and efficient, and generally will not cause memory problems even with very large documents because the programmer decides which pieces of information are saved and how they are saved.

4. Schemas

The structure of TCF files is defined in the schemas dspin_0_3.xsd and dspin_textcorpus_0_3.xsd. The file dspin_0_3.xsd is just a skeleton for defining the metadata and includes the file dspin_textcorpus_0_3.xsd. Besides the textCorpus format, other linguistic data formats can be specified, including lexicon or dictionary formats. Currently, TCF concentrates on the main layers of linguistic annotations.

5. Adding a Service to WebLicht

Once you have developed a service and made it available, you will need to register it in the WebLicht repository, which will make it available in the WebLicht web application. You may also need to have the schemas changed.

- If you are adding a layer that already exists in the schema (a “tokens” layer, for example), please use the same xml output format that is used by existing services producing that layer.

- If you are adding a layer that does not yet exist in the schema, the D-Spin development team will need to add the new layer to the schema.

For information about changing the schema and/or registering new services in WebLicht, please contact the D-Spin development group at:

weblicht@d-spin.org