Processing text

Bringing texts into good shape

DK-CLARIN WP 2.1 Technical Report
Jørg Asmussen, DSL, with input from other WP 2 members
Draft version of April 7, 2016[1]

Deliverables concerned

[...]

Outline of this document

[...]

[1] A more recent version may be available at:
http://korpus.dsl.dk/clarin/corpus-doc/text-processing.pdf.

1 Implementation

The text-conversion services are implemented in part as standalone Java programs, web-services based on eXist-db/XQuery,[2] as well as a combination of both approaches. The following description only gives an account of DSL's conversion software as DSN's never has been documented.

1.1 Web-services

Two equally popular web-service architectures are REST (Representational State Transfer) and SOAP (Simple Object Access Protocol). They each have their merits and drawbacks.

A major advantage of SOAP is that it can satisfy a wide range of non-functional requirements, in particular Quality of Service (QoS) requirements like secure, reliable and protocol-independent messaging. For SOAP to work there must be an HTTP body in which to place the SOAP envelope (containing the payload of the message and its metadata). This means that SOAP web-services always rely on the POST method even for so-called idempotent operations (i.e. operations which do not change anything on the server, e.g. simple requests for information). A drawback of SOAP is, besides relying on the POST transfer method only, that it involves the use of a quite verbose XML format (literally hundreds of different specifications) which may introduce overhead and needless complexity.

While REST does not allow for much QoS, its key merit is exactly its simplicity and transparency. According to the REST web-service design pattern, everything is considered a resource, and all resources are organized like a standard file system (which, in fact, resembles the structure of the early WWW). The operations which can be performed on resources are limited to the standard HTTP methods, i.e. POST, PUT, GET and DELETE.

Since the eXist open source XML database system is already being used in DK-CLARIN as a text-bank platform, see , and since eXist features an integrated REST-style HTTP server interface, REST was selected as the architectural style for the CTB web-services. Another reason for avoiding SOAP is that QoS aspects are not an issue, so using SOAP and WSDL would only introduce needless complexity.

[2] Services based on eXist-db will be gradually replaced by Java/Glassfish-based services in future development.

Having decided on a native XML database as the platform for the text-bank in DK-CLARIN WP2.1 and WP2.2, it seemed only logical to stay with the XML family and select the XQuery technology as the implementation language for the web-services. It only made the choice even more obvious that XQuery scripts which are stored in an eXist database collection can, in fact, be executed by simply pointing your web browser to the REST URL.[3]

As described in the online developer's guide for eXist,[4] eXist databases can be deployed in various ways, one of them being a stand-alone server process accessible through a REST-style API through HTTP. This is the simplest and quickest way to access the database because eXist features a built-in web server which conveniently treats all HTTP request paths as paths to a database collection.

The default listen address for the eXistServlet is

- `http://localhost:8080/exist/rest`

but when running as a stand-alone process – as is the case for DK-CLARIN – the server listens to port 8088, e.g.:

- `http://localhost:8088/ctb/xq`

XPath expressions or XQueries can either be added directly to the request string as values of the request parameter, `_query`, e.g.

- `http://localhost:8080/exist/rest/db/shakespeare?_query=//SPEECH[SPEAKER=%22JULIET%22]&_start=3&_howmany=5`

or they can be stored on the server in a database collection and called in a similar fashion using a simple HTTP GET request, e.g.:

- `http://localhost:8080/exist/rest/db/test/guess.xql`

In both cases the server returns raw XML to the browser (unless otherwise specified in the query).

In the current implementation of the web-services the public endpoint

- `http://ctbws.dsl.dk/[web-service]`

is mapped to an otherwise 'hidden' eXist database server via DNS.

All requests are performed via the POST method and the MIME type of the data must be application/xml. The XML content that is posted to the web-service, i.e. the request body, has the following basic structure for all web-services described below:

[3] See http://exist.sourceforge.net/devguide_xquery.html#storedxq for more details.

[4] See http://exist.sourceforge.net/deployment.html.

```
<request xmlns="http://ctbws.dsl.dk/ns/request">
  [request contents]
</request>
```

Please note, that the namespace `http://ctbws.dsl.dk/ns/request` is obligatory and always must be given as value of the *xmlns* attribute of the `<request>` element.

1.1.1 Demo Application

An application containing a collection of interactive demos[5] describing the structure of the input for each web-service and giving examples of the output returned by each web-service is available at:

- http://korpus.dsl.dk/clarin/demo/webservice/

Depending on your Flash Player version and your browser you may be able to view and download the Flash/Flex source code of the demo application by right-clicking on the demo application and choosing the menu option *View Source*.

Please contact Jørg Asmussen at `ja@dsl.dk` before modifying and re-using the code!

1.2 Web-services and Java

There are many ways of generating and dispatching an HTTP POST request programmatically (as opposed to using an HTML form with action and method attributes), and there are multiple programming languages which can be used to implement a simple client which takes an XML structure as input, builds and dispatches the POST request and prints the response received from the server.

The following code illustrates how one could implement a simple Java client which generates a full TEI WP2 header using the DK-CLARIN WP2.1 make-header web-service. It can easily be modified to be used with one of the other services described in this:

```
*** TokenizeText.java ***
package mystuff;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.URL;
import java.net.URLConnection;
public class TokenizeText {
  public static void main(String[] args) {
    if (args.length < 2) {
      System.err.println("Usage: <service URL> <file with XML request>");
    } else {
      StringBuilder sb = new StringBuilder();
```

[5] The demo application needs the Flash Player 10 plugin in your browser to be functional.

```
      String s = "";
      String input = "";
      URL url;
      URLConnection urlConn;
      try {
        //Read the XML document and store it in a String object
        BufferedReader fin =
          new BufferedReader(new FileReader(args[1])); //Req.file
        while ((s = fin.readLine()) != null) {
          sb.append(s);
        }
        input = sb.toString();
        fin.close();
        // URL of pretokenizer WS
        url = new URL(args[0]); //URL of the service
        // URL connection channel.
        urlConn = url.openConnection();
        // Let the run-time system (RTS) know that we want input.
        urlConn.setDoInput(true);
        // Let the RTS know that we want to do output.
        urlConn.setDoOutput(true);
        // Specify the content type.
        urlConn.setRequestProperty("Content-Type",
                                   "application/xml");
        // Send POST output. No need to use
        // setRequestMethod("POST") since only POST requests have
        // HTTP bodies with a particular content-type
        OutputStreamWriter oswr =
          new OutputStreamWriter(urlConn.getOutputStream(),
                                 "UTF8");
        oswr.write(input);
        oswr.flush();
        oswr.close();
        // Get response data.
        BufferedReader br2 =
          new BufferedReader(new InputStreamReader(
                             urlConn.getInputStream(), "UTF8"));
        String str = "";
        while ((str = br2.readLine()) != null) {
          // Output could also be printed to a file
          System.out.println(str);
        }
        br2.close();
      }
      catch (Exception e) {
        System.err.println(e.getMessage());
      }
    }
  }
}
```

2 Header constructor: `make-header`

2.1 Description

The header constructor web-service takes as input a simplified header only carrying the most relevant information about a text. The input (i.e. the request body) must comply with a specific XML structure, cf. the request body shown in Sec-

tion 2.3 and the demo application described in Section 1.1.1.[6] On the basis of this input, a complete TEI-P5-WP2 header (full header) is returned to the client. In case of missing information in the simplified input header, the web-service automatically fills in default information in the full header. Even an empty request returns a full and formally correct header with all values set to defaults.

2.2 Implementation

The header constructor is implemented as an XQuery script `make-header` that just works as an interface between the client and the function module `lib/make-header.xqm`.

Source code

The XQuery source code of the header constructor service and its function module can be downloaded from the following URLs respectively:

- http://ctbws.dsl.dk/make-header.zip
- http://ctbws.dsl.dk/lib/make-header.xqm.zip

Please contact Jørg Asmussen at `ja@dsl.dk` before modifying the code!

2.3 Use

The endpoint of the header constructor web-service is the following URI:

- `http://ctbws.dsl.dk/make-header`

Data upload is achieved by the POST method and the MIME type of the data must be *application/xml*. The XML content that is posted to the web-service, the request body, must comply with the following structure – the element contents given are for illustration purposes only. They are taken from an authentic DDOC sample.

Request body[7]

The following request contains a so-called simplified header to be converted into a full TEI-P5-WP2 header by the make-header web-service. The simplified header is considered an interface between user applications and the full header: User apps just need to transform information into the simple and flexible structure of the simplified header, the make-header web-service ensures that any simplified header is converted into a – at any time – correct full TEI-P5-WP2 header. Hence,

[6] The make-header request template can also be downloaded directly from http://ctbws.dsl.dk/request-templates/make-header.xml.

[7] A `make-header` request demo can also be downloaded directly from http://ctbws.dsl.dk/request-templates/make-header.xml.

validating against a dedicated TEI-P5-WP2 header schema should not be necessary. However, testing the validity of values from fixed inventories is still necessary. If the structure of the full header is changed, the make-header web-service will be changed accordingly; all other conversion tools that just convert to the simplified header structure will not be directly affected by such structural changes. So using the simplified header as an interface between conversion tools and the full TEI-P5-WP2 header makes it easier to maintain conversion tools and conversions themselves less error-prone.

The structure of the simplified header is deliberately flat with just one level beneath the outermost request node. However, some elements work as containers for any number of other elements, cf. further descriptions below. The order of the elements in a simplified header is arbitrary. The element names are (almost) identical to the corresponding variable and value-set names used in. A simplified header does not need to be complete: Elements may be left out and the corresponding elements or attributes in the full header will be filled in with default values. If a value-set exists for a certain type of information, the default value from this value-set is used if it is declared. If it is not declared, default is *nil* (or *99999999* in the case of dates and numbers). If a value-set does not exist, default is always *nil* (or *99999999*).

The following example shows how header data of a DDOC text can be expressed by means of the simplified header. It shows all elements possible in the the simplified header, even such which in this particular case could be left out as they just contain default values which would be added automatically by the service. Numbers in square brackets refer to comments further below in this description.

```
<request xmlns="http://ctbws.dsl.dk/ns/request">
  <textTitle>Skal vi have 35-timers arbejdsuge... nu?</textTitle> (1)
  <titleLevel>a</titleLevel> (2)
  <editionTitle>Bytinget</editionTitle> (3)
  <textIds> (4)
    <textId type="ctb">1200001003</textId> (5)
    <textId type="ddo">HRfX</textId> (6)
  </textIds> (7)
  <samplingDeclaration>CTB excerpt</samplingDeclaration> (8)
  <sponsorName>DDO</sponsorName> (9)
  <captureOrgName>dsl.dk</captureOrgName> (10)
  <captureMethod>keyed-proof</captureMethod> (11)
  <captureYear>1992</captureYear> (12)
  <numberOfWords>463</numberOfWords> (13)
  <numberOfParagraphs>2</numberOfParagraphs> (14)
  <distributorOrgName>dsl.dk</distributorOrgName> (15)
  <availStatus>restricted</availStatus> (16)
  <availDescAcademic>partial</availDescAcademic> (17)
  <availDescNonCommercial>partial</availDescNonCommercial> (18)
  <availDescAll>partial</availDescAll> (19)
  <anonymDescAcademic>0</anonymDescAcademic> (20)
  <anonymDescNonCommercial>0</anonymDescNonCommercial> (20)
  <anonymDescAll>0</anonymDescAll> (20)
  <notes> (21)
    <note type="dsl.dk" [lang="en"]> (22)
      DDOC text sample converted to TEI-P5-WP2 format
    </note>
```

```
  <note type="dsl.dk" [lang="xx"]>Grp:Bytinget1KK; Num:1</note> (23)
  <note type="dsl.dk" [lang="xx"]> (24)
    SpbId: LPG;
    FøS: Århus;
    Bop: ?;
    Reg: Øs;
    Udd: cand.scient.pol.;
    Erh: folketingsmedlem;
    SpV: i;
    Rol: de;
  </note>
  <note type="dsl.dk" [lang="xx"]> (24)
    SpbId: LIG;
    FøS: Horsens;
    Bop: ?;
    Reg: Ve;
    Udd: tekn. forb.eksamen;
    Erh: folketingsmedlem;
    SpV: i;
    Rol: de;
  </note>
</notes>
<authors> (25)
  <author id="LPG"> (26)
    <name>Gammelgaard, Lars P.</name> (27)
    <role>empty</role> (28)
    <age>adult</age> (29)
    <sex>1</sex> (30)
    <dob>1945</dob> (31)
    <dobCert>high</dobCert> (32)
  </author>
  <author id="LIG"> (33)
    <name>Gyldenkilde, Lilli</name>
    <role>empty</role>
    <age>adult</age>
    <sex>2</sex>
    <dob>1936</dob>
    <dobCert>high</dobCert>
  </author>
</authors>
<translators> (34)
  <translator id="nil"> (35)
    <name>nil</name>
  </translator>
</translators>
<editors> (36)
  <editor id="nil"> (37)
    <name>nil</name>
  </editor>
</editors>
<publId>10013</publId> (38)
<publHouse>DR</publHouse> (39)
<publDate>1983</publDate> (40)
<publDateCert>low</publDateCert> (41)
<edIssue>empty</edIssue> (42)
<edSect>empty</edSect> (43)
<edVolume>empty</edVolume> (44)
<edChapter>empty</edChapter> (45)
<edPages>empty</edPages> (46)
<textUri>empty</textUri> (47)
<textFileName>ja-korpus.dsl.lan:/DOT/textrepository/
              collections/ddoc/speech/BYTINGET.SGM</textFileName> (48)
```

```
  <relatedItems> (49)
    <relatedItem id="nil"> (50)
      <type>nil</type> (51)
      <title>nil</title> (52)
    </relatedItem>
  </relatedItems>
  <projectIdentifier>DDOC-spoken</projectIdentifier> (53)
  <applications> (54)
    <application id="nil">
      <appXmlId>nil</appXmlId>
      <appType>nil</appType>
      <appTask>nil</appTask>
      <appVersionNumber>99999999</appVersionNumber>
      <appScope>nil</appScope>
      <appDescription>nil</appDescription>
    </application>
  </applications>
  <textCreationYear>1983</textCreationYear> (55)
  <textCreationYearCert>low</textCreationYearCert> (56)
  <languageId>da</languageId> (57)
  <languageCharacterisation>empty</languageCharacterisation> (58)
  <tdChannelMode>s</tdChannelMode> (59)
  <tdChannel>122</tdChannel> (60)
  <tdConstitutionType>unknown</tdConstitutionType> (61)
  <tdDerivationType>original</tdDerivationType> (62)
  <tdOriginalLanguageId>da</tdOriginalLanguageId> (63)
  <tdDomainDiscourse>general</tdDomainDiscourse> (64)
  <tdDomain>331</tdDomain> (65)
  <tdFactualityType>fact</tdFactualityType> (66)
  <tdInteractActive>plural</tdInteractActive> (67)
  <tdInteractPassive>world</tdInteractPassive> (68)
  <tdInteractRole>basic-basic</tdInteractRole> (69)
  <tdInteractAge>adult-adult</tdInteractAge> (70)
  <tdPrepType>none</tdPrepType> (71)
  <tdPurposeType>persuade</tdPurposeType> (72)
  <catRefs> (73)
    <catRef
      type="http://ctb.dsl.dk/class/catRef/DDOC/RePr.xml">r</catRef>
    <catRef
      type="http://ctb.dsl.dk/class/catRef/DDOC/Medi.xml">tv</catRef>
    <catRef
      type="http://ctb.dsl.dk/class/catRef/DDOC/Genr.xml">kul</catRef>
    <catRef
      type="http://ctb.dsl.dk/class/catRef/DDOC/GnTy.xml">kul</catRef>
  </catRefs>
  <classCodes> (74)
    <classCode
      type="http://ctb.dsl.dk/class/classCode/CLARIN/demo.xml">
      demoValue
    </classCode>
  </classCodes>
  <revisions> (75)
    <revision>
      <revisionDate>2010-01-01</revisionDate>
      <revisionOrgName>dsl.dk</revisionOrgName>
      <revisionType>created</revisionType>
    </revision>
  </revisions>
</request>
```

1. `<textTitle>` contains the title of the source text. If the `<textTitle>` ele-

ment is missing, the default value *nil* is inserted into the corresponding elements in the full TEI-P5-WP2 header. The *lang* attribute indicates the language of he title, default is *nil*.

2. As this text is part of a collection, that is a series of broadcasts, its title level – given by the <titleLevel> element – has to be marked as analytic, indicated by the value 'a'. Default is monographic, 'm', which means that the text is a stand-alone text, not a member of a collection. If a text is a stand-alone text, the <titleLevel> element can be left out. The make-header web-service then automatically inserts the default value into the corresponding slot in the full TEI-P5-WP2 header.

3. <editionTitle> contains the title of the collection of which the text is a member. If a text is not member of a collection, the <editionTitle> element can be left out. Default is *nil*. If the title of the collection is irrelevant (e.g. because the text is monographic), <editionTitle> should be set to *empty*. The *lang* attribute indicates the language of he title, default is *nil*.

4. <textIds> is a container element which means that it may contain any number of related other elements, in this case various ids for the same text.

5. The <textId> of *type* 'ctb' is an invented example although the first two digits (the prefix) indicate that this is a text from the DDOC. CTB text ids should be derived from the make-id web-service devoted solely to dispatching valid ids, cf. Section 5 on page 23.

6. The <textId> of *type* 'ddo' is the original text id from the DDOC which we want to keep in the new TEI-P5-WP2 header.

7. </textIds> marks the end of the <textIds> container.

8. The text is an excerpt, that is, not a complete text, so <samplingDeclaration> is set to 'CTB excerpt'. Default is 'CTB sample' which means that it is not known whether the text is complete or an excerpt. If the <samplingDeclaration> element is left out, the make-header web-service assumes the default value.

9. Sponsor was the DDO project so <sponsorName> is set to 'DDO'. Sponsor means the intellectually supporting initiative behind the text capture. Default: 'DK-CLARIN'.

10. <orgName> contains the name of the organization responsible for creating the electronic version of the text. Default: *nil*.

11. <captureMethod> describes how the text was captured. In this case the text was manually keyed, i.e. transcribed from audio-tapes, and proof-read. Default: 'file'.

12. <captureYear> contains the year the text was captured. Default is the current year (which must be set in the corresponding value set file).

13. <numberOfWords> holds the approximate number of words (tokens) in the text sample. A word count can be made by the web-service count-units, see Section 6 on page 26. Default: *99999999*.

14. <numberOfParagraphs> holds the approximate number of paragraphs in the text sample. A paragraph count can be made by the web-service count-units, see Section 6 on page 26. Default: *99999999*.

15. <distributorOrgName> indicates the organization responsible for the distribution of this text (if it may be distributed). Default: *nil*.

16. <availStatus> indicates the availability of the text. In this case, the text is not available to everybody, thus <availStatus> is set to 'restricted'. Default is also 'restricted' so the <availStatus> element is actually unnecessary in this case and could be left out. The resulting full header would be the same anyway.

17. <availDescAcademic> describes the availability status for users from academic institutions affiliated with DK-CLARIN; 'partial' means that they may search and view text contents limited to what is specified in Danish citation law. Default is also 'partial', so this element could be left out without affecting the resulting full header.

18. <availDescNonCommercial> describes the availability status for non-commercial user; 'partial' means that they may search and view text contents limited to what is specified in Danish citation law. Default is 'partial' too, so this element could be left out without affecting the resulting full header.

19. <availDescAll> describes the availability status for all other users, again it is 'partial'. Default is also 'partial' again, so this element could be left out without altering the resulting full header.

20. No anonymisations required for any user group (elements <anonymDescAcademic>, <anonymDescNonCommercial>, and <anonymDescAll>). Default value is in all cases '0', so the anonymDesc elements could be left out.

21. The <notes> element is a container for any number of <note> elements each of which carries a *type* attribute telling which organization is responsible for this note and a lang attribute that denotes the language of the note. Valid notes are listed in . Notes may give information that cannot be expressed elsewhere in the TEI-P5-WP2 header. Default for both *type* and <note> content is *nil*.

22. The first <note> in this example gives some information on the corpus from which this text has been taken. The *lang* attribute of this note is "en" meaning "English". The *lang* attributes in this and other elements are not mandatory and can be left out. The make-header service described in (2) ignores them.

23. Another <note> gives some admin info that is contained in the original DDOC header but cannot be expressed by means of the TEI-P5-WP2 header. The *lang* attribute of this note is the non ISO-value "xx" which means "formalized", i.e. the language of the note is formally constructed to express certain properties of the text that cannot be expressed elsewhere in the header.

24. Further <note> elements give additional author/speaker information which is contained in the original DDOC header but cannot be expressed in the TEI-P5-WP2 header. Again, the *lang* attribute is set to "xx".

25. The <authors> element encapsulates all authors (or speakers) who have produced this text. It could be left out; however, as a text must have an author, the make-header web-service would create a dummy author *nil* (meaning the author has not yet been identified).

26. Each author/speaker carries a unique id (attribute *id* of the <author> element) which should be derived from the make-id web-service devoted solely to dispatching valid ids, cf. 5 on page 23. In this case, for illustration purposes, the id is the original one used in the DDOC. Default is *nil*.

27. The <name> of the author given as 'lastName, firstName' if possible. Default: *nil*.

28. The <role> element tells who has contributed most to the text. The role of the major author is 'major', all other authors are classified as 'minor'. However, in this text, both authors have contributed equally much which means that the role is undeterminable which is indicated by the *empty* value. Default: 'major'.

29. The <age> element indicates the age group to which the author belonged when he produced the text. Default is 'adult' so in this example the <age> element could be left out as well.

30. The <sex> element gives the sex of the author/speaker: '1' means male. Default: '0' meaning unknown.

31. Author's date of birth <dob> given in the pattern *yyyy[-mm[-dd]]*. Default is *99999999*.

32. Certainty of the date of birth is expressed in the <dobCert> element. Default is 'high' so in this case the <dobCert> element is actually unnecessary.

33. Another author (that is speaker in this example). OBS! Each <author> element comprises the following subelements: <name>, <role>, <age>, <sex>, <dob>, and <dobCert>. They can be left out which means that they are automatically filled in with default values.

34. The <translators> element encapsulates any number of possible translators of the text. The element can be left out if it is not relevant. The make-header web-service then inserts a placeholder dummy translator named *empty* in the full header. In contrary to the dummy author whose name value is *nil*, the dummy translator carries the value *empty*, meaning that this information is irrelevant, that there is no translator.

35. A dummy <translator> always has *id* attribute of *nil* and a <name> element of *empty*. In the example, for illustration purposes, the <translator> element explicitly creates a dummy translator in the full header. However, the whole <translators> structure could be left out in this case, the result would remain the same. Each <translator> element has the same child elements as has an <author> element. So additional info concerning the translator(s) could be given as well.

36. The <editors> block comprises information about editors, its children being <editor> elements. Apart from its different element name, it is structurally fully identical to the <authors> and <translators> blocks. If no editors were involved in producing/publishing the text, this block can be left out. In that case, the make-header web-service inserts a dummy editor in the full header.

37. In the case of the present text, which is a (transcribed) radio broadcast in a series of broadcasts, there should be an editor involved, i.e. the person responsible for this series. However, the DDOC header structure is not designed for that type of information so it is missing in the DDOC. Hence, editor is set to *nil* in the editor element. Default is *empty*.

38. <publId> contains the id of the publisher pointing to a data collection with further info on the publisher or distributor of the text source. Publisher ids are defined in value set documents. Default: *99999999*.

39. <publHouse> contains the name of the publisher/distributor. Default: *nil*.

40. <publDate> contains the date of publication. Default: *99999999*.

41. <publDayCert> indicates the certainty of publication date. Default: 'high'.

42. Imprint info <edIssue> indicates the issue of this publication. Default: *nil*.

43. Imprint info <edSect> gives the section. Default: *nil*.

44. Imprint info <edVolume> contains volume information. Default: *nil*.

45. Imprint info <edChapter>: the chapter. Default: *nil*.

46. Imprint info <edPages>: pages info. Default: *nil*.

47. <textUri> contains URI of online version of the source text. Default: *nil*.

48. <textFileName> contains the file name of the input version of the text. Default: *nil*.

49. Parallel versions of this text or texts otherwise related are listed within the <relatedItems> block. Defaults: *nil*. In this case there are no related texts, so the block containing pointers to related texts could be left out and the web-service would just insert a dummy with default values. For illustration purposes, an explicit default dummy is defined.

50. Attribute *id* of the <relatedItem> element refers to the CTB text id of the related text.

51. <type> of textual relationship, e.g. 'original', 'parallel'. Default: *nil*.

52. <title> gives the title of the related text. The *lang* attribute indicates the language of he title, default is *nil*.

53. <projectIdentifier> contains a unique identifier of the text collection project in which this electronic text was captured and prepared. Default: *nil*.

54. The <applications> container is used for listing applications that have processed the text. The default-segmented base version is the result of a pre-tokenizer having operated on it. However, this is never stated in the application info block. Thus, in most cases, the applications container can be left out and the make-header service just creates an empty placeholder in the output. In order to show all relevant elements of an application, here, an empty application is given explicitly. For a detailed description of these elements see .

55. <textCreationYear> contains the year of text creation. Default: *99999999*.

56. <textCreationYearCert> gives info on how sure it is that the text was created in that year. Default: 'high'.

57. <languageId> indicates the predominant language of the text. Default: *nil*.

58. <languageCharacterisation> may give some further description of the language used. Default: *nil*.

59. <tdChannelMode> tells whether the text is spoken or written. Default: 'w'.

60. <tdChannel> indicates the medium through which the text was experienced: '122' means television. Default: *99999999*.

61. <tdConstitutionType> holds a description of the internal composition of a text. In this case, the text is a fragment, but is unknown whether it is continuous or not, so <tdConstitutionType> is set to 'unknown'. Default: 'single'.

62. <tdDerivationType> gives info on whether the text is translated or original. Default: 'original'.

63. <tdOriginalLanguage> tells what was the original language of the text. This info is particularly relevant in case the text is a translation, otherwise the value is the same as in <languageId>. Default: *nil*.

64. <tdDomainDiscourse> describes whether the text is LSP or LGP. Default: 'general'.

65. <tdDomain> gives the DDOC domain code. '331' means business ('erhvervsliv'). Default: *99999999*.

66. <tdFactualityType> gives info on whether the text is imaginative or non-imaginative. Default: 'inapplicable'.

67. <tdInteractActive> indicates the number of addressors having produced the text. Default: 'singular'.

68. <tdInteractPassive> indicates the number of addressees to whom a text is directed. Default: 'world'.

69. <tdInteractRole> indicates the roles of addressor and addressee in terms of technical expertise concerning the topic of the text. Default: 'basic-basic'.

70. <tdInteractAge> indicates the age groups to which addressor and addressee belong. Default: 'adult-adult'.

71. <tdPrepType> indicates the extent to which a text may be regarded as prepared or spontaneous. Default: 'revised'.

72. <tdPurposeType> indicates the purpose or communicative function of the text, e.g. whether it is informative, expressive, etc. Default: 'inform'.

73. <catRefs> is a container with additional textual classifications in cases where the classification system follows a project-internal scheme. As the sample is from the DDOC, the additional classifications are DDOC-specific and the corresponding valuesets are given as vaules of the <catRef> attribute type. If no <catRefs> are given, the web-service generates one dummy <catRef> element with *nil* values.

74. <classCodes> is a container with classifications based on official text classification schemes. As no official classification scheme is used in the DDOC, the <classCodes> container gives just one single (superfluous)

<classCode> demo. If no <classCodes> are given, the web-service generates one dummy <classCode> element with 'nil' values.

75. The <revisions> block contains revision information on this text. If no revisions are given, the web-service generates a dummy <revision> element with a <revisonDate> of *99999999*, a <revisionOrgName> of *nil*, and a <revisionType> of 'created'.

The example given above shows all elements of the simplified header. However, as the make-header service employs defaults in all cases where corresponding information in the simplified header is missing, many elements of the example above would be left out in a real setting. The resulting response would be exactly the same. The reader is encouraged to experiment with this in the interactive demo application at http://korpus.dsl.dk/clarin/demo/webservice/.

Response

The response from the make-header service is a full TEI-CTB header formatted according to the specifications given in . What this header looks like can be seen by running the above example in the demo application at http://korpus.dsl.dk/clarin/demo/webservice/. The above example is the default example of the demo app.

Client development

The Java client example shown in Section 1.2 on page 4 can easily be adopted to be used with this web-service too.

3 Pre-tokenizer: pretokenize

3.1 Description

The pre-tokenizer web-service takes as input a raw text (preferably with <p> annotations), a prefix letter (for prefixing token ids) and finally a text id which has to be unique within the DK-CLARIN project. The input must comply with a specific XML structure (cf. the demo in Section 1.1.1). On the basis of this input, a pretokenized version of the text (no header) is returned to the client. The pretokenized format complies with TEI-P5 and will be referred to as TEI-P5-WP2 format in this document.

For some tasks the TEI-P5-WP2 format will constitute an adequate degree of tokenization, but for other tasks it may not. For example, common multiword expressions like *i går* ('yesterday') are not recognized and annotated as a single token. For this reason we refer to the tokenizer as a "pretokenizer" and encourage

users to produce separate annotation layers (span groups) on the basis of TEI-P5-WP2 in case they need more sophisticated or customized tokenization, see .

A key issue when implementing a tokenizer is which characters should be considered punctuation (i.e. non-word characters). Punctuation characters should be characterized by having no semantic impact on the words in the text. In other words, if a punctuation character is removed from the text, this should in no way change the meaning of the words in the text (denotational, connotational, or otherwise). If, for example, currency symbols, degree symbols and so on were to be deleted, it would cause a loss of meaning. For this reason such characters are not considered punctuation.

3.1.1 List of punctuation characters

On the basis of discussions in the WP2 project group, which agreed upon using the punctuation characters listed in Wikipedia, cf. http://da.wikipedia.org/wiki/Tegns\T1\aetning and http://en.wikipedia.org/wiki/Punctuation, the list of punctuation characters used in the pre-tokenizer is limited to the following:

Table .1: Punctuation characters

Character	Description	Code	Variants
’	apostrophe	0027	
(	bracket, round, opening	0028	
)	bracket, round, closing	0029	
[	bracket, square, opening	005B	
]	bracket, square, closing	005D	
{	bracket, curly, opening	007B	
}	bracket, curly, closing	007D	
:	colon	003A	
,	comma	002C	
-	dash/hyphen	002D	2013 en dash: – 2014 em dash: —
/	slash	002F	005C backslash: \
_	underscore	006F	2026 horizontal ellipsis: …
!	exclamation mark	0021	

Table continues on next page…

Character *(continued)*	Description *(continued)*	Code *(continued)*	Variants *(continued)*
.	full stop	002E	
“	upright double quotation mark	0022	00AB left guillemet: « 00BB right guillemet: » 2018 left single: ‘ 2019 right single: ’ 2039 left single guillemet: ‹ 203A right single guillemet: › 201C left double: “ 201D right double: ”
?	question mark	003F	
;	semicolon	003B	
¿	various punctuation	00BF	

Whitespace characters are captured by a regular expression which conflates whitespace sequences into a single whitespace character (`\s+`). The pretokenizer does not record the number or types of whitespace in the input.

3.2 Implementation

The pre-tokenizer is implemented as an XQuery script which in addition to built-in functions like `tokenize` and `normalize-space` makes use of the following five self-defined functions:

1. `tok:tokenize-doc(text,id-prefix)`
2. `local:isolate-punctuation(string)`
3. `local:punct(string,id)`
4. `local:space(id)`
5. `local:token(string,id)`

Four of the self-defined functions are local, but the `tokenize-doc` function resides in the `tok` namespace and is the main function which calls the other four functions to carry out subtasks of the tokenization process. All five functions are grouped into an XQuery module `pretokenize.xqm`, and this module is included

from a single XQuery script pretokenize which is invoked by requests addressed to the pretokenizer web-service at http://ctbws.dsl.dk/pretokenize.

The tokenize-doc function takes two arguments, namely the <p> annotated text and an id prefix. This function iterates through the <p> elements of the input document, calls the local:isolate-punctuation function on each <p> element (which inserts ^ characters around all punctuation characters and whitespace sequences as defined in Section 3.1.1) and then tokenizes the text string in the <p> element using the ^ character as delimiter.

For each token, tokenize-doc calls either local:punct, local:space, or local:token depending on the contents of the token. These three functions in turn insert <c> or <w> elements in the output with appropriate id numbers as attributes. The main XQuery script finally wraps everything in a <text> element and returns it to the client.

Source code

The XQuery source code of the tokenizer service and the tokenizer function module can be downloaded from the following URLs respectively:

- http://ctbws.dsl.dk/pretokenize.zip
- http://ctbws.dsl.dk/lib/pretokenize.xqm.zip

Please contact Jørg Asmussen at ja@dsl.dk before modifying the code!

3.3 Use

The endpoint of the pretokenizer web-service is the following URI:

- http://ctbws.dsl.dk/pretokenize

Data upload is via the POST method and the MIME type of the data must be application/xml. The XML content which is posted to the web-service, the request body, must comply with the following structure – the element contents given are for illustration purposes only.

Request body[8]

```
<request xmlns="http://ctbws.dsl.dk/ns/request">
  <idPrefix>A</idPrefix>
  <textId>2100000000</textId>
  <text>
    <p>A paragraph.</p>
    <p>Another paragraph.</p>
```

[8] A pretokenize request demo can also be downloaded directly from http://ctbws.dsl.dk/request-templates/pretokenize.xml.

```
  </text>
</request>
```

Response

The reponse from the service is the following TEI fragment:

```
<text xmlns:math="http://www.w3.org/1998/Math/MathML"
      xmlns="http://www.tei-c.org/ns/1.0"
      xmlns:xi="http://www.w3.org/2001/XInclude"
      xmlns:svg="http://www.w3.org/2000/svg"> >
  <body>
    <p>
      <w xml:id="A-2100000000-2-1">A</w>
      <c xml:id="A-2100000000-2-2" type="s"/>
      <w xml:id="A-2100000000-2-3">paragraph</w>
      <c xml:id="A-2100000000-2-4" type="p">.</c>
    </p>
    <p>
      <w xml:id="A-2100000000-4-1">Another</w>
      <c xml:id="A-2100000000-4-2" type="s"/>
      <w xml:id="A-2100000000-4-3">paragraph</w>
      <c xml:id="A-2100000000-4-4" type="p">.</c>
    </p>
  </body>
</text>
```

Client development

The Java client example shown in Section 1.2 on page 4 can easily be adopted to be used with this web-service too.

4 Text id registry: `register-text`

4.1 Description

The text id registry web-service takes as input an XML request body with 3 arguments (expressed by XML elements) of which the original id of the text is the most important. It checks whether this text id is already contained in the text id registry or not. As part of the response, a boolean is returned that is true if the text id is already a member of the registry or false otherwise. If the text id does not yet exist in the registry, it is inserted.

For each request made to the text id registry web-service, the result of the request is logged, i.e. whether the text may be inserted in the CTB or not.

4.2 Implementation

The text id registry web-service is implemented as an XQuery script `register-text` which is invoked by requests addressed to the corresponding web-service at

- http://ctbws.dsl.dk/registry/register-text.

The main functionality of this service is found in the function module `/lib/registry/register-text.xqm`.

The registry itself is an XML document located under `http://ctb.dsl.dk/registry/text/`. The name of the document is identical with the name of the text group, the exentsion of the document is `.xml`. An example is

- http://ctb.dsl.dk/registry/text/demo.xml.

The corresponding logfile is located at

- http://ctb.dsl.dk/registry/text/demo.log.xml.

The following example illustrates the structure of the registry document (with just one text id registered):

```
<textRegistry group="infomedia">
  <txt id="e18062c7"
       org="dsl.dk"
       ins="2009-11-05T18:09:35.578+01:00"/>
</textRegistry>
```

For each text id inserted there is one `<txt>` element. The attribute *id* contains the text id, *org* indicates the organisation responsible for handling this text, and *ins* tells when this text id was registered.

The following example shows the structure of a log document:

```
<textRequestLog group="infomedia">
  <txt id="e18062c7"
       insert="true"
       org="dsl.dk"
       reqTime="2009-12-30T18:09:35.601+01:00"/>
  <txt id="e18062c7"
       insert="false"
       org="dsn.dk"
       reqTime="2009-12-31T08:01:11.711+01:00"/>
</textRequestLog>
```

Each request made to the text id registry is logged in one `<txt>` element whose attributes give the result and further info of that request. Thus *id* contains the text id this request was made on, *insert* tells whether the text with the corresponding id

may be inserted in the CTB or not: if *insert* is 'true´ it may be inserted, if it is 'false', the text most likely already has been inserted in the CTB and should therefore be rejected this time. Attribute *org* gives the organisation having made the logged request and *reqTime* gives the date and time of that request.

Source code

The XQuery source code of the registry service and the registry function module can be downloaded from the following URLs respectively:

- http://ctbws.dsl.dk/registry/register-text.zip
- http://ctbws.dsl.dk/lib/registry/register-text.xqm.zip

Please contact Jørg Asmussen at ja@dsl.dk before making any modifications to the code!

4.3 Use

The endpoint of the textregistry web-service is the following URI:

- http://ctbws.dsl.dk/registry/register-text

Data upload is via POST and the MIME type of the data must be application/xml. The XML content which is posted to the web-service, the request body, must comply with the following structure – the element contents given are for illustration purposes only. **Do not send any requests to the 'infomedia' text group unless you have been explicitly authorized to do so by Jørg Asmussen, otherwise the Infomedia registry may be corrupted.** **OBS!**

Request body[9]

```
<request xmlns="http://ctbws.dsl.dk/ns/request">
  <textGroup>infomedia</textGroup>
  <textId>e18062c7</textId>
  <organisation>dsl.dk</organisation>
</request>
```

The element <textGroup> indicates the name of the group of texts; various groups – and thus various corresponding registry documents – may be defined. However, at the moment, there is only one 'real' registry document, that is the one used for Infomedia texts, the text group 'infomedia'. In addition, there is a group 'demo' for demo purposes.

Please **do not use the Infomedia group** as its registry document may be cor- **OBS!**

[9] A register-text request demo can be downloaded directly from http://ctbws.dsl.dk/request-templates/registry/register-text.xml.

rupted by improper use! Use the 'demo' text group instead as it has been created for testing purposes!

The element <textId> contains the original text id used by the text group in question. In the case of Infomedia texts, it is the original text id used by Infomedia. In the case of the 'demo' group, for testing the functionality of this service, any string may be given as input.

Finally, <organisation> gives the name of the organisation being responsible for the text in question.

New registry documents and logfiles must be created manually directly in the eXist-db.

Response

The following XML content illustrates the structure of the reponse from the service:

```
<response>
  <textId>e18062c7</textId>
  <exists>false</exists>
  <inserted>true</inserted>
</response>
```

Element <textId> is the text id from the request, <exists> indicates whether it already is in the registry, and <inserted> tells whether it has been inserted into the registry; this is always the case if it is not already registered.

Client development

The Java client example shown in Section 1.2 on page 4 can easily be adopted to be used with this web-service too.

5 id dispatcher: make-id

5.1 Description

The id dispatcher web-service takes as input an XML request body with 2 arguments (expressed by XML elements). It returns CTB-valid ids for texts and persons (authors, editors, and translators). It should be used when adding TEI-WP2 headers to texts to be included in the CTB, and it ensures that ids are correct according to the definitions.

5.2 Implementation

The id dispatcher web-service is implemented as an XQuery script make-id which is invoked by requests addressed to

- http://ctbws.dsl.dk/registry/make-id.

The main functionality of this service is found in the function module /lib/registry/make-id.xqm.

ids are dispatched according to the values recorded in an id registry. The registry itself is a couple of XML documents located under http://ctb.dsl.dk/registry/id/. There is one registry document for each of the 2 id classes 'text' and 'person'. In addition, there is a 'demo' class for demo purposes only that should be used with the online demo interface and the like. **Please do not use other types than 'demo' if you just want to test the service!** Otherwise the id counters would be unnecessarily altered. An example of an id registry document is **OBS!**

- http://ctb.dsl.dk/registry/id/demo.xml.

The following example illustrates the structure of the registry document that controls dispatching text ids – the example here is just an excerpt from the full document which is located at http://ctb.dsl.dk/registry/id/text.xml:

```
<idRegistry>
  <idClass prefix="10" mnemo="k2000">
    <first>0</first>
    <next>103506</next>
    <last>99999999</last>
  </idClass>
  <idClass prefix="21" mnemo="dkclarin21">
    <first>0</first>
    <next>117988</next>
    <last>99999999</last>
  </idClass>
  <idClass prefix="22" mnemo="dkclarin22">
    <first>0</first>
    <next>0</next>
    <last>99999999</last>
  </idClass>
</idRegistry>
```

There is one id class for each project indicated by a 2-digit *prefix* attribute and somewhat clarified by the 'mnemonic' value of the *mnemo* attribute. The <first> element contains the first legal id value in a particular <idClass>, <next> the next one to use, and finally, <last> gives the highest id to be used within that class.[10]

[10] The current implementation of this web-service does not check whether the highest legal value is exceeded.

Source code

The XQuery source code of the id dispatcher service and the registry function module can be downloaded from the following URLs respectively:

- http://ctbws.dsl.dk/register/make-id.zip
- http://ctbws.dsl.dk/lib/register/make-id.xqm.zip

Please contact Jørg Asmussen at ja@dsl.dk before making any modifications to the code!

5.3 Use

The endpoint of the id dispatcher web-service is the following URI:

- http://ctbws.dsl.dk/register/make-id

Data upload is via POST and the MIME type of the data must be application/xml. The XML content which is posted to the web-service, the request body, must comply with the following structure – the element contents given are for illustration purposes only. **Do not send any requests to the 'text' or 'person' id classes unless you have been explicitly authorized to do so by Jørg Asmussen.** OBS!

Request body[11]

```
<request xmlns="http://ctbws.dsl.dk/ns/request">
  <idClass>text</idClass>
  <idPrefix>21</idPrefix>
</request>
```

The element <idClass> indicates the id class, either 'text', 'person', or 'demo'. Please **do not use the 'text' or 'person' class** unless you have been explicitly allowed to do so! Use the 'demo' class instead as it has been created for testing purposes! OBS!

The element <idPrefix> contains the prefix to use for the texts in question. A complete list of prefixes can be found at

- http://korpus.dsl.dk/clarin/corpus-doc/text-header/vs_textId.xml.

[11] A make-id request demo can also be downloaded directly from http://ctbws.dsl.dk/request-templates/make-id.xml.

Response

The following XML content illustrates the structure of the response from the service:

```
<response>
  <idClass>text</idClass>
  <id>2100000719</id>
</response>
```

Element <idClass> gives the id class (taken from the request), <id> gives the dispatched id. Ids of classes 'text' and 'demo' consist of the prefix followed by 8 digits whereas 'person' ids have 10 digits following the prefix. In addition, 'demo' ids have the prefix *Demo-*.

Client development

The Java client example shown in Section 1.2 on page 4 can easily be adopted to be used with this web-service too.

6 Word and paragraph counter: count-units

Work in progress.

Document history

The most recent version may have some minor fixes that are not explicitly listed in the history.

20.01.2011:

- Modified description of the *make-header* service (and the service itself), cf. Section (2), in order to produce headers with *lang* attributes according to the newest header definition.