The Corpus Thread

"Reference corpus of general language"

The complete documentation of the DK-CLARIN WP 2.1 Project

Jørg Asmussen
ja@dsl.dk

Det Danske Sprog- og Litteraturselskab
Society for Danish Language and Literature
Dept. for Digital Dictionaries and Text Corpora
Christians Brygge 1, DK-1219 Copenhagen K
dsl.dk

March 5, 2014

Preface

This book documents the work carried out in Work Package 2.1 (WP 2.1) of the DK-CLARIN project: *Reference Corpus of general language*. The project was officially running three and a half years from 2008 – 2011. The work in WP 2.1 was carried out by Jakob Halskov (Dansk Sprognævn), Liisa Theilgaard (Society for Danish Language and Literature, DSL), and Jørg Asmussen (DSL). All documentation is by Jørg Asmussen with input from other members of DK-CLARIN.

The goal of the DK-CLARIN Project in general was to provide a research infrastructure for the humanities integrating written, spoken, and visual records into a coherent and systematic digital repository. This repository should be available at clarin.dk where further information on the project can be found as well.

The DK-CLARIN project has stopped, however the type of corpus work described in this documentation is continuing and further developing at the *Department for Digital Dictionaries and Text Corpora* at DSL. Thus parts of this documentation may have been updated and new tools and resources may have become available. Further information can be found at corest.dsl.dk.

This book is structured in three parts: The *Introduction* part gives an account of the goal of WP 2.1 and of the tasks that have been carried out to reach it. The *Design* part is a comprehensive description of the applied text format and metadata structure. The *Collecting* part sketches how to manage collecting and and processing the material. The *Markup* part deals primarily with POS tagging and discusses various approaches as well as the applied approach and its specific requirements. Finally, the *Deployment* part, gives some details on the composition of the corpus and describes show it can be accessed.

Contents

Part I

Introduction

Chapter 1

Aim and concepts

Sketching the world of corpora

Deliverables concerned

This chapter concerns the DK-CLARIN work package 2.1 *Reference corpus of general language* as well as corpus projects carried out by Det Danske Sprog- og Litteraturselskab, dsl.dk. An overview of project-specific deliverables, that is *project tasks*, is given in Section 1.2.

Outline

This chapter gives an overview of DK-CLARIN work package 2.1 *Reference corpus of general language* as a whole and sketches major concepts of this work package and corpus projects carried out by DSL in general.

1.1 Aim of the project

The aim of the project[1] is twofold:

Corpus: Gather a reference corpus of general Danish according to certain design principles.

To achieve this goal, it is necessary to develop a framework for managing the construction process of corpora. This leads to the second aim:

Framework: Establish a *Corpus Management and Retrieval System* (CMRS) as a framework for building and analyzing corpora. The result of this is an in-house (DSL) collection of methods, tools, and means of structured data storage, together with this comprehensive documentation. The CMRS concept established in WP 2.1 aims at being transferable to other corpus management and retrieval scenarios.

1.1.1 Reference corpus

The reference corpus of general Danish that is gathered as part of this DK-CLARIN project (WP 2.1) comprises 45 million running tokens and is textually mixed, although – as a matter of rather limited resources – texts from periodicals like newspapers and magazines are preferred as they are more straightforward to process especially if they come in some kind of XML format as is the case for texts from one of DSL's main sources, Infomedia.[2]

[1] Throughout the documentation *the project* refers to DK-CLARIN work package 2.1, i.e. building a reference corpus of general language.

[2] http://infomedia.dk/

Metadata The texts of the corpus are provided with metadata given as XML-formatted headers. These ensure that texts can be filtered according to different textual parameters. However, the level of detail is restricted to the general information attached to the material when it is received from the text supplier.

Markup The tokens of the material are tagged with information on lemma forms, part-of-speech, and inflectional markers.

Accessibility A copy of the corpus is included in the DK-CLARIN repository of resources and tools. It is made publicly accessible through a web-based concordancer as well.[3]

1.1.2 CMRS framework

As the *Corpus Management and Retrieval System* is an essential prerequisite for the accomplishment of the project – working as its "corpus factory" –, we will start with an overview of this system and give a brief description of its components.

Figure 1.1 shows the *compositional structure* or "architecture" of the CMRS. The figure follows largely the Fundamental Modeling Concepts framework, FMC. However, some minor modifications have been made to a few symbols of the notation framework, in particular, the symbol for read/write access to a storage (two rounded unidirectional arrows in FMC) has been replaced by a strong white bidirectional arrow that consumes less space and can be applied more flexibly. FMC's channel concept (usually denoted by a small circle) has been replaced by a somewhat broader concept of interfaces through which communication between compound components working together as a system and other systems or users outside take place. Figure 1.2 shows the types of interfaces used in this notation.[4]

As illustrated in Figure 1.1, the major data repository of the CMRS is the *text bank* that holds four different types of data: a collection of the *texts* themselves, a collection of *metadata* describing the general characteristics of each individual text, a collection of *annotations* providing various linguistic information on text token level, and finally *supplier details* providing information on contacts, text deliverances, and agreements. The text bank is fed with textual material from text suppliers through *import handlers*, that is, transducers that convert the various text formats into the specific one(s) needed in the CMRS. Texts may be manually complemented with metadata through a *metadata editor*, the same goes for supplier details. Annotating the text material with linguistic information is performed by the *corpus processing unit* that utilizes linguistic data drawn from external resources and processed by *lexical transducers*. The corpus processing unit also extracts *formatted corpora* according to different structural specifications that may

[3] The corpus will be included as a stand-alone corpus in KorpusDK and should be publicly available under ordnet.dk/korpusdk from winter 2013-14.

[4] See www.fmc-modeling.org for a quick introduction and pointers to further reading.

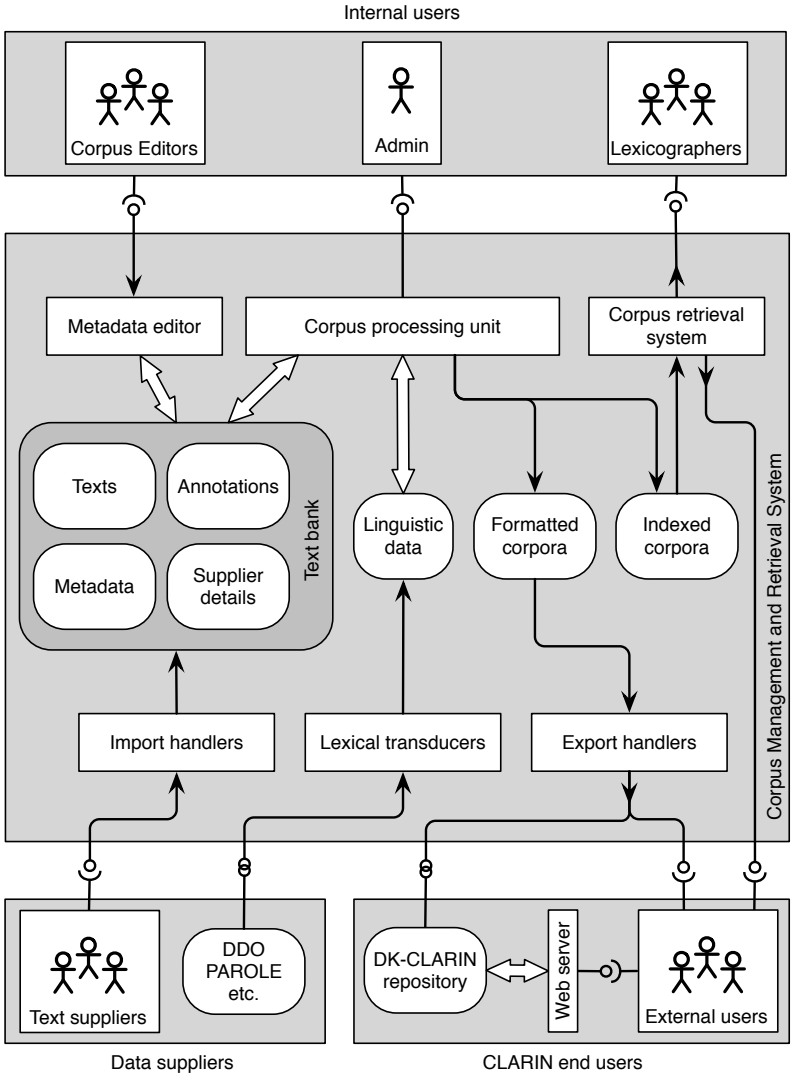

Figure 1.1: Major components and interfaces of the CMRS

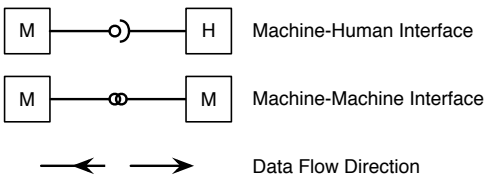

Figure 1.2: Symbols for various types of interfaces

be exported through export handlers. Finally, the corpus processing unit produces *indexed corpora* that can be deployed to external/internal *users* through a *corpus retrieval system* – a corpus search engine like a concordancer for example. The difference between formatted and indexed corpora is that formatted corpora are formatted in accordance with a (TEI) schema of some kind and contain text samples possibly together with metadata and some sort of linguistic annotations on token level. In contrast, indexed corpora are made searchable in some sort of search engine. Metadata and annotations may be searchable as well as may be additional lexical data derived from the material, e.g. different types of frequency lists, collocations, or other statistical material.

1.2 Project tasks and documentation outline

In order to achieve the major aims of the project, that is building a CMRS and use it for gathering a reference corpus, a number of steps to perform were defined in the original project plan[5] where the result of each of these steps is termed *deliverable*. Some of these steps are directly related to the design and construction of the corpus, others are mainly related to establishing the CMRS. So the list of steps to be taken constitutes an unordered to-do list rather than reflecting the sequential process of either building a CMRS or applying it to "assemble" corpora. As opposed to the list of deliverables, the present documentation is structured in order to reflect the *sequential process* of making a corpus from the *design* phase, over *collecting* and *markup*, to final *deployment*.

In the following, the steps/deliverables of the original plan are listed together with references to the chapters in this book where the corresponding documentation – that serve as *deliverable reports* – can be found. The documentation proper gives further pointers to the tools and resources.[6]

D1 Text registry DSL as well as DSN collect Infomedia text material, parts of which are likely to be included in the WP 2.1 corpus. Therefore, a way of registering texts needs to be established. A registry allows tracing and eliminating possible duplicate texts. The text registry functionality is part of the CMRS. **Outcome:** Report.

▷ Chapter 2

D2 Tokenizer A consistent and easy-to-use token concept needs to be defined. The token concept has important implications on the design of the tokenizer tool and the POS-tagger applied in WP 2.1. **Outcome:** Tool and report.

[5] http://korpus.dsl.dk/clarin/wp21/wp21-arbejdsplan-old.pdf

[6] The CMRS itself is not considered a DK-CLARIN deliverable as it is not possible to fully implement it due to limited resources. Instead accounts of its different components is given in this book. Based on the documentation, it should be straightforward for other, similar projects to reuse some of the design considerations applied.

▷ Chapter 4

▷ Chapter 5

D3 Decision on text bank system A text bank system is necessary for project-internal text administration. Investigations of different approaches to such a system will be carried out. Two general options seem viable – either one based on a relational db or on an XML db. The text bank system is the core component of the CMRS. **Outcome:** Report.

▷ Chapter 2

D4 Text supplier registry A registry of active and potential text suppliers needs to be designed as an integrated component of the CMRS. **Outcome:** Report.

▷ Chapter 2

D5 Implementation of text bank system The chosen text bank approach (see D3) implemented (possibly with a GUI) as component of the CMRS. **Outcome:** Report and a project-internal service.

▷ Chapter 2

D6 Processing of Infomedia text Conversion to the DK-CLARIN format and text bank import of Infomedia material collected by DSL and DSN in 2008 and 2009. **Outcome:** Report.

▷ Chapter 5

D7 Development of format transducers Design and development of transducers capable of transforming all supplier formats into the WP2.1 text format. **Outcome:** Report and project-internal services.

▷ Chapter 5

D8 Processing of other text Collected text material is converted and inserted into the text bank component of the CMRS. **Outcome:** Report.

▷ Chapter 5

D9 Full-form lexicon Development and/or configuration of a full-form lexicon for POS tagging. **Outcome:** Resource with documentation.

▷ Chapter 8

D10 Lemmatizer It is considered indispensable that corpus texts need to indicate the lemma form of each inflected word form in the corpus to let the user of the corpus perform more flexible queries. Therefore, it is necessary to either develop or configure a lemmatizer (that may be based on a full-form lexicon or a morphological analyzer). In the context of WP 2.1, a lemmatizer designed as an integral part of a POS tagger is the preferable solution. **Outcome:** Tool with documentation.

- ▷ Chapter 6
- ▷ Chapter 7
- ▷ Chapter 8

D11 POS tagger In order to tag tokens in corpus texts with part-of-speech information, it is necessary to either develop or configure a POS tagger (either based on a full-form lexicon or a morphological analyzer) and a suitable tag set. **Outcome:** Tool with documentation.

- ▷ Chapter 6
- ▷ Chapter 7
- ▷ Chapter 8

D12 Download service Implementation of download option for copyright-cleared or scrambled text material. **Outcome:** None. Cancelled due to project cutbacks.

D13 TEI transducer The original plan for WP 2.1 was based on the assumption that the repository of potential corpus texts – the corpus text bank – most likely would have a non-XML structure (relational db). In order to make interchange of texts easy and in order to make them fit into the intended resource repository of DK-CLARIN, the development of a transducer that could reshape the texts and metadata stored in the corpus text bank to valid TEI XML seemed necessary. However, during the course of the project, it became clear that the text bank itself should be implemented as an XML database so that the texts could be stored in their final TEI XML format. Therefore, the task of developing a transducer became a task of defining an appropriate subset of TEI in order to suit the metadata and text format needs of DK-CLARIN. **Outcome:** Report.

- ▷ Chapter 3
- ▷ Chapter 4
- ▷ Chapter 5

D14 Prototype of concordance tool A web-based concordance tool needs to be configured/implemented as a prototype for testing. **Outcome:** Report.

- ▷ Chapter 10

D15 Panel of test users Constitution of a panel of test users. **Outcome:** None. Cancelled due to project cutbacks.

D16 User tests Performing and evaluating user tests of web-concordancer. **Outcome:** None. Cancelled due to project cutbacks.

D17 Final version of concordance tool Web-based concordancer with public access. **Outcome:** Service with documentation.

- ▷ Chapter 10

D18 Final version of corpus Final version of POS-tagged corpus of 45 million words available for the DK-CLARIN repository and accessible through a web-based (or other) concordance tool. **Outcome:** Resource with documentation.

- ▷ Chapter 9

As already mentioned above, the chapters do not follow the order of steps/deliverables but instead a more general structure as the process of building a corpus can be subdivided into four phases:

1. **Design:** Textual metadata must be determined as well as annotations on other textual levels. A repository for storing the text material – a text bank – needs to be designed and implemented as part of the CMRS. Designing a text bank includes designing the representation of text data.

   The following chapters cover the design phase of the project:

   - ▷ Text metadata: Chapter 3
   - ▷ Text formatting: Chapter 4
   - ▷ Text bank: Chapter 2

   These chapters cover the following tasks/deliverables of the project:

   - ▷ D 1 – D 5 and D 13

2. **Collecting:** This phase covers negotiations with potential text suppliers, the conversion of text material gathered in a myriad of odd formats into the standard format defined during the design phase, and finally storing it in the text bank. It also covers the task of manually adding missing metadata.

   The following chapter covers the phase of collecting material:

- ▷ Text processing: Chapter 5

A chapter on text acquisition would be desirable as well, however, as this documentation focuses on the technical aspects of building a corpus, it has been left out.

This chapter covers the following tasks/deliverables of the project:

- ▷ D 2, D 6 – D 8, and D 13

3. **Markup:** Texts that have been converted to the standard format are ready to have various types of annotations added. In this project, all words in a text are tagged with the following types of annotations: A standardized orthographic form of the word, its lemma form, its part of speech as well as some inflectional information. In order to carry out these types of markup, certain tools need to be developed and/or configured as part of the markup phase.

   The following chapters cover the markup phase:

   - ▷ Survey of POS taggers: Chapter 6
   - ▷ Design of the jaPOS tagger: Chapter 7
   - ▷ The full-form lexicon: Chapter 8

   These chapters cover the following tasks/deliverables of the project:

   - ▷ D 9 – D 11

4. **Deployment:** Deploying the corpus means to make it accessible for end-users, either through a corpus retrieval system of some kind that needs to be developed/configured, or by distributing the text files, that make up the corpus, in a standard format.

   The following chapters cover the markup phase:

   - ▷ Corpus specifications: Chapter 9
   - ▷ Forthcoming *PAROLE version 2* documentation
   - ▷ Forthcoming *Corpus retrieval software (CoREST)* documentation
   - ▷ Corpus access: Chapter 10

   This chapter covers the following tasks/deliverables of the project:

   - ▷ D 14 and D 17

1.3 Text collection, text bank, corpus

This Section aims at giving some clarification on the concepts *text collection*, *text bank*, and *corpus*, of which the two latter play an important role in the CMRS.

1.3.1 Text collection/archive/repository

A text collection is a collection of complete or abridged texts of any kind collected by a project or institution/company. The purpose of collecting the material may be documentation or archiving in general for purposes not yet defined – and often, corpus construction is not considered an option at all. As a consequence of not having specified an explicit purpose for the text collection other than maybe documentation, the process of collecting is often opportunistic rather than guided by certain corpus-compositional principles. The texts of a text collection may carry a well-defined minimum of annotations on text level. Based on these annotations, a subset of text items may be exported from the the archive. An archive may be a structured means of storage, e.g. a database holding the texts in some generalized format.[7] However, there does not need to be any structured means of storage – the material may reside unordered in a file system and may be composed of texts with various incompatible formats. In relation to the CMRS, it may be considered text material that has not yet been imported into the CMRS but is stored in other accessible locations and formats.

1.3.2 Text bank

Whereas a text collection still may be rather unstructured, a text bank is an implementation geared to storing and retrieving texts to be potentially included in linguistic corpora. So, in contrast to a text collection, the explicit purpose of the texts gathered in a text bank is to be able to build corpora from them. It allows to better process and organize potential corpus text material.[8] Therefore, a text bank requires an elaborate structure: Texts of a text bank must carry well-defined meta-information (e.g. expressed as an XML-formatted header), they must follow a well-defined format, that is, they should be tokenized and each token should have a unique reference ID in order to be addressed unambiguously. A text bank provides furthermore interfaces/handlers through which texts and metadata can be added and through which texts can be selected for export as a corpus in an appropriate format. Corpus query functions for linguistic investigation are not part of the text bank, but are features of separate corpus query and statistics tools. The text bank is first and foremost a tool for text and corpus administration.

1.3.3 Corpus

A corpus is a group of text items from a text bank that have been selected due to explicit criteria based on the information given in the meta-data part of a text item. The purpose of a corpus is to allow certain linguistic investigations as it is assumed that the corpus (text items as whole) constitutes a representative sample of the sort

[7] Examples of text collections are The Oxford Text Archive and Arkiv for Dansk Litteratur.

[8] The text bank must not be confused with the general DK-CLARIN repository developed in WP 5 that is supposed to support various data types (e.g. texts, images, lexicons) and various formats to be used in various contexts, not just corpus construction.

of language to be investigated. Each text item in a corpus either carries the same meta-data that are used in the text bank or a subset of them, e.g. only those that are relevant for the corpus in question. A corpus can be supplemented with meta-data describing its characteristics, e.g. the purpose of it and the selection criteria for the texts it is comprised of. Corpus texts usually carry several token annotation layers, e.g. an orthographically normalized version of the token, a lemmatized form of it, POS information, inflectional information. A corpus of this type can be made accessible for queries in a concordancer or it may be used to be processed by other corpus tools, e.g. for statistical purposes.

Part II

Design

Chapter 2

The text bank

A platform for managing text collections
DRAFT VERSION

Deliverables concerned

D1 Text registry DSL as well as DSN collect Infomedia text material, parts of which are likely to be included in the WP 2.1 corpus. Therefore, a way of registering texts needs to be established. A registry allows tracing and eliminating possible duplicate texts. The text registry functionality is part of the CMRS. **Outcome:** Report.

D3 Decision on text bank system A text bank system is necessary for project-internal text administration. Investigations of different approaches to such a system will be carried out. Two general options seem viable – either one based on a relational db or on an XML db. The text bank system is the core component of the CMRS. **Outcome:** Report.

D4 Text supplier registry A registry of active and potential text suppliers needs to be designed as an integrated component of the CMRS. **Outcome:** Report.

D5 Implementation of text bank system The chosen text bank approach (see D3) implemented (possibly with a GUI) as component of the CMRS. **Outcome:** Report and a project-internal service.

Outline of this chapter

This chapter gives an account of the text bank, that is, of its intended functions as text repository and administrative registry as well as its implementation. The db software should be able to store and give multi-user access to XML text documents, manage text supplier data, and facilitate the detection of duplicate text material. This chapter also serves as documentation for WP 1.10 and 1.21 as described in Asmussen (2008).

2.1 Introduction

Crucial for composing corpora is a repository of texts from which texts according to a specific profile can be selected. The corpus itself is considered a separate resource. This repository is intended to contain corpus-suitable texts with certain textual metadata expressed in a systematic way that can be expressed as XML, see Chapter 3. A repository of this kind is called a *text bank*.[1] A text bank may provide some administrative functions as well like handling text supplier information.

In the following, the implementation of the text bank is described in further detail. This is followed by a description of the text registry and the text supplier registry functionality, see Figure 1.1.

2.2 Implementation

2.2.1 XML vs. relational db systems

The fundamental units of a corpus are *text items* drawn from a text repository. A repository containing potential corpus texts in a standardized text and metadata format only is called a *text bank*. As corpus texts to be included in the text bank are XML-formatted according to a specific schema, see chapter 3 and chapter 4, it

[1] See also Section 1.3.2.

seems obvious to store them in an XML database. Moreover, it is necessary to be able to access and edit these fundamental text units with a viewer/editor.

Storing the text material in a relational database would require substantial processing of the material prior to database import and export. As such, conversion procedures introduce an extra amount of resources and complexity, they must be considered error-prone in terms of appliance and maintenance. Therefore, it appears obvious to apply an XML-based db solution. An overview of XML databases can be found on Wikipedia.

To minimize software expenditures, only open source products have been examined. Among these, projects with low or none obvious development activities were rejected, that is Xindice, myXMLDB, ozone. Sedna could be a candidate, but it is obviously not supported by the oXygen XML editor, the same seems to be the case for MonetDB and BaseX. After these exclusions the only candidate left is eXist.

2.2.2 eXist – the text bank system by choice

2.2.2.1 Advantages

- Native XML db
- Easy to install and maintain
- Built-in indexing (automatic and user-defined) which means quick searches
- XQuery is used to manipulate data
- Theoretically unlimited document size. So far, the 13,5 MB DK-PAROLE Corpus has been the largest document uploaded
- Can store up to 2^{31} documents
- Accessible from the oXygen editor
- Easy to set up as a web service. This should make it straightforward to
  - have the eXist-based text bank to fit into the envisaged DK-CLARIN infrastructure
  - develop a stand-alone text bank interface and skip the oXygen editor

2.2.2.2 Disadvantages

So far, the following disadvantage could be identified:

Non-commercial project: As for most software development projects of this kind, there is no guarantee for continuous development as well as support may be unreliable.

This disadvantage should be taken into account for future development. In particular, web services probably should not be based entirely on eXist's XQuery interface but should be encapsulated by a self-developed web service interface that accesses eXist but could be changed to access other db

2.2.2.3 Current implementation and set-up

The CTB is located in the eXist XML document collection `/db/ctb` as data repository and an oXygen editor as a basic user interface which communicates with the data repository by means of a oXygen-eXist db connection. The oXygen editor will be replaced by a dedicated viewer/editor during the project period. The eXist service is installed on the host `ja-korpus.dsl.lan` until a more convenient solution is available.

2.2.2.4 User Interfaces

2.3 Features

2.3.1 Text repository

2.3.2 Text registry

2.3.3 Text supplier registry

2.4 Alternative approaches

Chapter 3

Text metadata

What the header of a text item looks like[1]

Deliverables concerned

D13 TEI transducer The original plan for WP 2.1 was based on the assumption that the repository of potential corpus texts – the corpus text bank – most likely would have a non-XML structure (relational db). In order to make interchange of texts easy and in order to make them fit into the intended resource repository of DK-CLARIN, the development of a transducer that could reshape the texts and metadata stored in the corpus text bank to valid TEI XML seemed necessary. However, during the course of the project, it became clear that the text bank itself should be implemented as an XML database so that the texts could be stored in their final TEI XML format. Therefore, the task of developing a transducer became a task of defining an appropriate subset of TEI in order to suit the metadata and text format needs of DK-CLARIN. **Outcome:** Report.

[1] A header/text template can be downloaded from:
http://ctb.dsl.dk/templates/formatsample.xml
The corresponding XML schema is available at:
http://dkclarin.dk/schemas/WP2
Schema read-me file available at:
http://dkclarin.dk/schemas/WP2/README_TEIP5DKCLARIN_validation.pdf

Outline of this chapter

This chapter describes how the metadata part of text items can be expressed by means of a TEI P5 header whereas Chapter 4 describes the text part proper. One major aim of the header design described in this chapter is to integrate header information from text items in existing corpora of Danish language, i.e. the Corpus of the Danish Dictionary and PAROLE-DK, KORPUS 2000, other corpus-relevant material from DOT/DSL, as well as the LGP and LSP corpora of written Danish which are compiled as part of DK-CLARIN.

Guide to reading this chapter

The structure of the header is oriented towards that one used by the BNC Burnard (2007) and PAROLE-DK Keson (1998b) but tries to avoid idiosyncrasies not covered by TEI P5 as well as modifications of the TEI header schema.

Section 3.1 summarizes some corpus linguistic concepts used throughout the DK-CLARIN project, which are described in further detail in Chapter 1.

Section 3.2 gives a general account of the header structure of headers of text items to be included in the *Corpus Text Bank*, CTB.[2] The description of the CTB header structure is in its starting point strongly inspired by that one given in Burnard (2007). This section constitutes the major part of this chapter.

Section 3.3 starts with a complete header template and describes in detail the sets of values that have to be used to fill in the header. It can be used as a manual for those who have to fill in text headers with appropriate information, either manually or automatically by converting and mapping existing material. This section

[2] The CTB is a text repository of written texts that are candidates to be included in a linguistic corpus. The CTB has been developed by WP 2.1 in order to better process and organize potential corpus text material, see III. It must not be confused with the general DK-CLARIN repository developed in WP 5 that is supposed to support various data types (e.g. texts, images, lexicons) and various formats.

is probably too detailed for those readers who just want the more general lines of how the CTB header is composed and may therefore be skipped by most readers.

3.1 Concepts

A *text item* consists of a *text* potentially to be included in a corpus, and of some metadata about the text. The metadata is typically contained in a *header* which precedes the text proper.[3] A text item is the smallest chunk of text plus metadata in a repository of potential corpus texts – a *corpus text bank* – from which text items are selected for inclusion in a specific corpus. Thus, a text item is the smallest corpus-compositional unit. The text part of a text item is either a complete text (usually a shorter one) or a sample taken from a longer text, e.g. a chapter from a book, see Chapter 1. Longer texts, e.g. novels, are divided into smaller parts, e.g. chapters, before they are included in a corpus text bank. A corpus text bank may be considered as a somewhat more specialized kind of text archive, intended to contain all kinds of corpus-relevant text chunks. The reason why longer texts are chopped into smaller chunks is that this subsequently makes corpus composition more precise as text-typological fine-tuning becomes easier – a novel, for instance, is less likely to skew the intended balance of a corpus if it can be selected from the text bank in smaller quantities, e.g. chapter-wise.

This chapter describes the header structure of text items collected in the *Corpus Text Bank* (CTB) – a corpus text bank for all kinds of written corpus-relevant texts collected as part of the DK-CLARIN project's work package 2.1: "Basic written language resources — Reference corpus of general language". Text items from the CTB may be included in one or more specific corpora intended for linguistic research. A *corpus* is a more organized collection of texts compiled on the basis of the text bank for a specific – i.e. linguistic – purpose. Text material being collected for literary purposes or as part of an electronic library (archive) may stress other features of the TEI header proposal. Here, the header structure is adopted to the specific needs of *corpus* texts.

Text item headers are structured by means of TEI P5. In the following, this structure adapted to the needs of structurally integrating various existing corpora or text collections is described in detail. The collections to be structurally integrated are the *Corpus of the Danish Dictionary* (DDOC, Norling-Christensen and Asmussen (1998)), *PAROLE-DK* (Keson (1998a) and Keson (1998b)), *KORPUS 2000* (Andersen et al. (2002)), other corpus-relevant material from DOT/DSL and Dansk Sprognævn (DSN), as well as the LGP and LSP corpora of written Danish which are compiled as part of DK-CLARIN.[4]

[3] Another solution would be to store the metadata in a separate database and establish a link between text and metadata.

[4] Text material from the *Arkiv for Dansk Litteratur* (ADL) and other archives may at a later stage be integrated as well, if the header structure of their texts can be mapped to that one described here.

The TEI header structure provides extremely flexible means of expressing textual metadata. A wealth of information can be given in a more or less fine-grained way. The following Section 3.2 describes a header that exactly accommodates the needs of potential corpus texts. In many cases, TEI allows the header to be modified either by augmenting or simplifying it. However, a header with more or less information is still compatible with the model described here as long as its structure does not conflict with TEI P5 syntax (and semantics) requirements.

Therefore, the following section does not describe a TEI header in general, but the specific header of a potential corpus text in the Corpus Text Bank of WP 2.1, expressed by means of TEI.[5]

3.2 Header structure

The header of a text item provides a structured description of the text contents, analogous to the title page and front matter of a book. Every separate text item in the Corpus Text Bank has its own header `<teiHeader type="text">`. In addition, a corpus itself may have a header `<teiHeader type="corpus">` containing information which is applicable to the whole corpus. The corpus header is not part of this description. To a large extent, a corpus header would be an abridged and slightly modified version of a text header. Furthermore the corpus header should contain the declaration of value sets for various elements (e.g. a domain taxonomy for LSP texts). The Corpus Text Bank contains value declarations in form of a collection of certain value set files which may be referenced by the CTB header. The content structure of the Corpus Text Bank is described in detail in Chapter 2 The value set files proper are described in detail in Section 3.3.2.

The remainder of this section describes the components of the `<teiHeader type="text">` element as used within the Corpus Text Bank. A TEI header contains a file description (Section 3.2.1), an encoding description (Section 3.2.2), a profile description (Section 3.2.3), and a revision description (Section 3.2.4), represented by the following four elements:

`<fileDesc>` (file description) contains a full bibliographic description of an electronic text as well as the source from which it was derived.

`<encodingDesc>` (encoding description) documents the relationship between an electronic text and the source from which it was derived.

`<profileDesc>` (text-profile description) provides a detailed description of non-bibliographic aspects of a text, specifically the languages and sublanguages used, the situation in which it was produced, the participants and their setting.

[5] The header design has been adopted for text resources to be included in the DK-CLARIN repository developed by WP 5.

<revisionDesc> (revision description) summarizes the revision history for a file.

3.2.1 The file description

The file description <fileDesc> is the first of the four main constituents of the header. It is intended to document a digital file. It contains the following four subdivisions:

<titleStmt> (title statement) groups information about the title of a work represented in the electronic text sample and those responsible for its intellectual content.

<extent> specifies the size of the electronic text sample in number of words and paragraphs (and other countable units).

<publicationStmt> (publication statement) groups information concerning the publication or distribution of the electronic text sample.

<notesStmt> (notes statement) collects together any notes providing information about a text additional to that recorded in other parts of the bibliographic description.

<sourceDesc> (source description) supplies a description of the source text from which the digital text sample was derived.

Further detail for each of these is given in the following subsections.

3.2.1.1 The title statement

The title statement <titleStmt> element of a text item contains one <title> element, followed by one <sponsor> and one <respStmt> element as shown in the following pattern:

```
<titleStmt>
  <title>
    samplingDeclaration textTitle
  </title>
  <sponsor>sponsorName</sponsor>
  <respStmt>
    <resp>Data capture</resp>
    <name>organizationName
      <note type="method">captureMethod</note>
      <date when="captureYear"/>
    </name>
  </respStmt>
</titleStmt>
```

The content of the <title> element is an initial caption (*samplingDeclaration*), e.g. "CTB version of:",[6] followed by the title of the source text (*textTitle*). Thus, the contents of the title element resemble that one used in PAROLE-DK: "Tagged sample of: '*textTitle*' ". As the CTB virtually can contain both tagged (even differently tagged) and untagged text, any statements about whether the text is tagged in some respect or not must not be made in the <title> element but should be given as *application information*, see Section 3.2.2.3.

The <title> element is followed by a <sponsor> element indicating the name of the sponsoring organization or institution.[7] According to the TEI guidelines, sponsors give their intellectual authority to a project; they are to be distinguished from funders, who provide the funding but do not necessarily take intellectual responsibility. The <sponsor> content of material captured as part of the DK-CLARIN project is "DK-CLARIN". Texts which were captured in other projects (and made available to DK-CLARIN) have their own specific <sponsor> content.

A <respStmt> element is used to indicate each institution responsible for any significant effort in the creation of the electronic text sample. The CTB header has only one responsibility statement indicating the responsibility for original data capture. The name of the responsible institution is given as an constant string for each institution in a <name> element. The <note> element of type "method", subordinate to <name> gives an indication of how the text was captured, e.g. by scanning or typing. Finally, the year of data capture is given as a four-digit date (or a complete date) as the value of the *when* attribute in the <date> element subordinate to <name>.

PAROLE-DK's header does neither include sponsor nor responsibility information, whereas the BNC uses lots of <respStmt> elements with great verbosity. In PAROLE-DK, this information instead is virtually part of the <publicationStmt> assuming that the distributor is always the same as the organization responsible for data capture (and is the sponsor). Here, it is assumed that the sponsor, the collector, and the distributor are of central importance and that it cannot be taken for granted that these decisive roles are played by one organization only. However, it is assumed that these roles are fully sufficient to describe the institutional background of a potential corpus text. Additional roles may come into play for a whole corpus or text collection and must be part of the headers of these resources.

OBS! Author and editor information for the source from which a text is derived (e.g. the author of a book) is not included in the <titleStmt> element but in the <sourceDesc> element discussed below in Section 3.2.1.5.

[6] Other *samplingDeclaration* captions are acceptable as well. A complete list is given in Section 3.3. The chosen caption must always be identical to the string value given in the <samplingDecl> element, see Section 3.2.2.1. In the example given, *CTB* stands for *Corpus Text Bank*.

[7] An alternative (and probably more appropriate) expression instead of *sponsor* would be *initiative*.

3.2.1.2 The extent statement

The <extent> element is used in each text header to specify the size of the text to which it is attached. The size is given as the number of words in the <num> element, the *n* attribute is set to "words". In another <num> element with the n attribute set to "paragraphs" the number of paragraphs is stated.[8] Other <num> elements measuring extent in other units may be added, but must be registered as part of the legal inventory described in Section 3.3:

```
<extent>
  <num n="words">numberOfWords</num>
  <num n="paragraphs">numberOfParagraphs</num>
</extent>
```

The count given does not include the size of the header itself. The number of words and paragraphs must be mechanically computed prior to insertion of the text into the text bank.

3.2.1.3 The publication statement

The <publicationStmt> element is used to specify publication and availability information for an electronic text. It contains the following three elements:

<distributor> supplies the name of a person or agency responsible for the distribution of a text.

<availability> supplies information about the availability of a text, for example any restrictions on its use or distribution, its copyright status, etc.

<idno> (identifying number) supplies an identifying code for a text.

```
<publicationStmt>
  <distributor>organizationName</distributor>
  <idno type="textIdType">textId</idno>
  <availability status="availStatus">
    <ab type="availGroup">availDesc anonymisationDesc</ab>
    <ab type="availGroup">availDesc anonymisationDesc</ab>
    <ab type="availGroup">availDesc anonymisationDesc</ab>
  </availability>
</publicationStmt>
```

The <distributor> element contains the name of the organization[9] responsible for the distribution of the electronic text sample. Usually there can only be

[8] This is a necessary extent information particularly for texts which are to be included in parallel corpora.

[9] In DK-CLARIN this will typically be a member of the DK-CLARIN consortium.

one distributor for each text even though TEI allows to repeat this element as often as needed. The inventory of strings denoting distributors should be invariant, i.e. one name only per distributor.

The obligatory CTB text id is given as contents of an `<idno type="ctb">` element. Some dialects of TEI introduce an attribute *id* of the `<TEI>` element which is illegal according to strict TEI. Other types of text, project-, or institution-internal identifications may be given in additional `<idno>` elements whose type attributes indicate the specific type of id.

The text strings in `<ab>` ('anonymous block')[10] elements given under `<availability>` for both restricted (attribute *status* is set to "restricted") and free (attribute *status* is set to "free") give availability information for three fixed user categories: academic users, non-commercial users, and all types of users.

Academic users are defined as users who are affiliated with the DK-CLARIN consortium.

Non-commercial users are academic users not affiliated with the DK-CLARIN consortium, users from educational or governmental institutions.

All users are any type of users including commercial users.

The DK-CLARIN license committee has finally, i.e. at the end of the project, concluded that the types of licenses should be employed: public, academic and restricted and that licenses are to be managed outside text headers. However, WP 2.1 will stick to the categories and values described above.

The following pattern shows the substructure of the `<availability>` element:[11]

```
<availability status="restricted">
  <ab type="academic">
    <seg type="availDesc">availDesc</seg>
    <seg type="anonymDesc">anonymDesc</seg>
  </ab>
  <ab type="nonCommercial">
    <seg type="availDesc">availDesc</seg>
    <seg type="anonymDesc">anonymDesc</seg>
  </ab>
  <ab type="all">
    <seg type="availDesc">availDesc</seg>
```

[10] This type of elements is preferred to the alternative `<p>` which is semantically misleading – these are no paragraphs but blocks of information.

[11] The `<availability>` element requires subordinate `<p>` or `<ab>` elements thus inhibiting more meaningfully structured availability information. The cumbersome typed `<ab>` and `<seg>` elements thus seem to be the only way of expressing structured availability information, unless TEI P5 is modified.

```
      <seg type="anonymDesc">anonymDesc</seg>
    </ab>
  </availability>
```

The various values are defined in Section 3.3. Two types of values are given in two subordinate <seg> elements: The availability description *availDesc* and a description of how to anonymize private information associated with the text, *anonymDesc*. If availability for any user category is other than "full" or any kind of anonymization is required, that is if *anonymDesc* is other than "nothing" (i.e. value "0"), the availability *status* attribute is set to "restricted", otherwise it is set to "free".

TEI allows a <date> element as part of <publicationStmt>; however, it is left out here, as the CTB version of a text cannot be said to having been published at a given time. Text bank texts may undergo changes (e.g. annotations are modified, more detailed info is given in the header) some of which are time-stamped in the revision description of the header, see Section 3.2.4, so the texts can never be said to be final, but they are available at all times in the shape they have at a given point in time. However, they may be published as part of a corpus, hence the <date> element under <publicationStmt> should be part of the corpus header.

3.2.1.4 The notes statement

The <notesStmt> contains one or more <note> elements, each containing a single piece of descriptive information, which does not fit into other parts of the header. Each <note> element carries an obligatory *xml:lang* attribute indicating the language of the note as well as a *resp* attribute denoting the organization responsible for this note, that is, the organization that has authored this note:

```
<notesStmt>
  <note xml:lang="languageId"
  resp="organizationName">note</note>
</notesStmt>
```

3.2.1.5 The source description

The <sourceDesc> element is used to supply bibliographic details for the original source material from which an electronic text sample derives. In the case of DK-CLARIN corpus texts, this may be a book, pamphlet, newspaper, etc. or an electronic source of some (non-TEI) format. Within the <sourceDesc> element several sub-structures are available according to TEI. Here, the <biblStruct> sub-structure is used in almost the same way as in PAROLE because it imposes a fixed structure on the bibliographic description and, most importantly, because it allows to distinguish between information

concerning the text proper and information concerning the edition (e.g. book, newspaper) from which the text was drawn:

```
<sourceDesc>
  <biblStruct>
    [...]
  </biblStruct>
</sourceDesc>
```

The <biblStruct> element contains the following three elements:

<analytic> (analytic level) contains bibliographic elements describing an item (e.g. an article or poem) published within a monograph or journal and – according to the TEI guidelines – not as an independent publication. In the CTB headers, though, it is used for independent publications as well, see below.

<monogr> (monographic level) contains bibliographic elements describing an item (e.g. a book or journal) published as an independent item (i.e. as a separate physical object).

<idno> (identifying number) supplies any standard or non-standard number used to identify a bibliographic item.

<relatedItem> may contain a reference to some other bibliographic item related to the present one in some specified manner, for example as a translation of it. However, the use of this element is deprecated as the quality and quantity of relationships between texts may vary depending on the perspective of the user, therefore they should not be treated as a fixed information in the header of a text. Instead, various relation files should be introduced that relate any number of texts to each other in any way. The format of these relation files should be defined in a technical report. The substructure of the deprecated <relatedItem> is:

```
<relatedItem type="relatedType">
  <bibl>
    <title xml:lang="languageId">relatedTitle</title>
    <idno type="ctb">relatedId</idno>
  </bibl>
</relatedItem>
```

It must be placed as last element in <biblStruct> and it may be repeated as many times as necessary.

The complete substructure of <biblStruct> looks as follows:

```
<biblStruct>
  <analytic>
```

```
      <title xml:lang="languageId"
        level="titleLevel">textTitle</title>
      <author>
        <name ref="#personId">surname, forename</name>
      </author>
      <respStmt n="translators">
        <resp>Translated by</resp>
        <name ref="#personId">
          surname, forename
        </name>
      </respStmt>
    </analytic>
    <monogr>
      <title xml:lang="languageId">editionTitle</title>
      <editor>
        <name ref="#personId">surname, forename</name>
      </editor>
      <imprint>
        <publisher n="publId">publHouse</publisher>
        <date when="publDate" cert="certainty"/>
        <biblScope type="issue">edIssue</biblScope>
        <biblScope type="sect">edSect</biblScope>
        <biblScope type="vol">edVolume</biblScope>
        <biblScope type="chap">edChapter</biblScope>
        <biblScope type="pp">edPages</biblScope>
      </imprint>
    </monogr>
    <idno type="uri">textUri</idno>
    <idno type="file">textFileName</idno>
    <relatedItem type="relatedType">
      <bibl>
        <title xml:lang="languageId">relatedTitle</title>
        <idno type="ctb">relatedId</idno>
      </bibl>
    </relatedItem>
  </biblStruct>
```

According to the TEI guidelines,

> [in] common library practice a clear distinction is usually made between an individual item within a larger collection and a free-standing book, journal, or collection. Similarly a book in a series is distinguished sharply from the series within which it appears. An article forming part of a collection which itself appears in a series thus

> has a bibliographic description with three quite distinct levels of information: the analytic level, giving the title, author, etc. of the article; the monographic level, giving the title, editor, etc. of the collection; the series level, giving the title of the series, possibly the names of its editors, etc. and the number of the volume within that series.[12]

The aim of the bibliographic information for texts which are intended to be included in a corpus, that is the type of texts collected in the Corpus Text Bank, is not to imitate the precision of a librarian but to give an easy way of referring to texts and to probably use bibliographic information in some corpus searches as well. This requires a rather fixed and to some extent rigid structure of the bibliographic part of the header which is the reason why the <biblStruct> structure is used here and not one of the other (less structured) possibilities of TEI. The <biblStruct> structure can be used to distinguish between the three information levels discussed above in the TEI guideline snippet. Here, only two of the levels are used, namely the analytic and the monographic level. The <monogr> element in the <biblStruct> structure is obligatory. According to TEI, it seems that in the case of a text being monographic, the <analytic> part of the structure should be left out and the text title and author information should be given within the <monogr> part of the structure. However, in the CTB headers, the <analytic> part is considered *obligatory*, no matter whether the text is part of a collection of some kind, i.e. analytic, or a stand-alone publication, i.e. monographic. This is to ensure that all <biblStruct> elements in CTB headers have the same structure, that text title and author information is always found in the same place, that is in the obligatory <analytic> part of the structure.

Within the <analytic> structure, <title> always gives the title of the text. If the text is part of a collection, e.g. a newspaper article which is part of a newspaper, the *level* attribute of <title> is set to "a" which means *analytic*, whereas the <title> element in <monogr> gives the title of the collection, e.g. the name of a newspaper. If the text is a free-standing book, e.g. a novel, the *level* attribute is set to "m" meaning *monographic*; in such cases the <title> element in the <monogr> part is left empty. All <title> elements carry the obligatory attribute *xml:lang* indicating the language of the title.

The author of a text is always given in <author> in the <analytic> part of <biblStruct>. There is one <author> element for each author who has contributed to the document. The name of the author is given in a <name> element. If the name has been decomposed into forename and surname, the information is given as *surname, forename(s)*, otherwise the comma is left out. If the name of the author is unknown, the <name> element is filled in with an *unknown* symbol (see Section 3.3), if it for some reason is anonymous, the <name> element is filled in with the string "anonymous". A <name> element should have a *ref* attribute giving an XML reference to a corresponding <person> element in the

[12] See http://www.tei-c.org/release/doc/tei-p5-doc/en/html/CO.html.

<profileDesc> part of the header where additional info concerning the author(s) is given, see Section 3.2.3.5.[13]

PAROLE has no participant description as part of the profile description. Instead, PAROLE augments TEI by adding two arguments (*gender* and *born*) to the <author> element. In contrast to PAROLE, the CTB header defers from altering the TEI proposal.

The <author> element is followed by a <respStmt> with an obligatory attribute *n* carrying the constant value "translators" that contains the name(s) of the person(s) who has/have translated this text if it is a translation, otherwise <respStmt> is filled in with the *empty* symbol, see Section 3.3. The <respStmt> element contains an obligatory <resp> element with the fixed string "Translated by" and a subsequent <name> element of *type* "translator" gives the name of the translator. If there is more than one translator, additional <name> elements are used.[14] If the translation has been carried out by a company or the like, the name of the company is given. The <name> elements may carry a *ref* attribute giving a reference to a corresponding <person> element in the <profileDesc> part of the header where additional info concerning the translator(s) may be given. This <name> element is of special relevance to texts which may be included in parallel corpora. More on translated texts can be found under the description of the <derivation> element in Section 3.2.3.3.

In the <monogr> part, the title of the collection is given if the text is part of a collection, otherwise it is left empty. The name of the editor is given in a <name> element as *surname, forename(s)*; if it is undeterminable how to decompose the name into forename(s) and surname, the comma is left out. If there are more than one editor, each of them is given in its own <editor> element. If there is no editor, the <name> element of <editor> carries an *empty* symbol, see Section 3.3. The <name> elements may carry a *ref* attribute giving a reference to a corresponding <person> element in the <profileDesc> part of the header where additional info concerning the editor(s) may be given.

In the <imprint> part of <monogr>, the name of the publishing house is given in the element <publHouse>,[15] the obligatory date of publishing as value of the *when* attribute of <date>, either the year or – in the case of newspapers – the year, month, and day according to the pattern *yyyy-mm-dd*. The *cert* attribute

[13] It may seem odd that the *ref* attribute is given on the <name> element and not on the <author> element which would have been an option. However, as *ref* attributes also are used with translators and editors and neither the <respStmt> element used for translators nor the <editor> element are allowed to carry a *ref* attribute, it is instead attached to the <name> element in all these cases.

[14] It may seem inconsequent to repeat the <name> element for each translator whereas in case of the author and editor, the corresponding <author> and <editor> elements are repeated. However, as there obviously is no <translator> element in TEI, and as <respStmt> cannot carry a *type* attribute, repetition of the semantically rather empty <respStmt> element with its obligatory subordinate <resp> element (giving the semantics) seems much too awkward and would furthermore increase the complexity of queries.

[15] This element may be repeated if more publishers are to be listed.

of <date> tells the certainty of the date which can either be “high” or “low”. If the exact date is not known, an estimate is given and the *cert* attribute is set to “low”. <imprint> includes five <biblScope> elements of different types which have to be filled in with the appropriate types of information, see Section 3.3. If a certain type of information does not apply to the publication described, it is left empty.

The <monogr> part of the structure is followed by an <idno> element of type “uri”[16] where a web pointer to the text can be given, i.e. the location from which it can be or has been downloaded. Other possible types are “isbn” and “issn”. If it for some reason seems necessary to register the ISBN or ISSN, <idno> elements of the corresponding types can be added as well.

Another <idno> element of type “file” follows. As texts in most cases are delivered as electronic files, a back-reference to this source file is made by stating its filename and if necessary the path to it in this element. The file itself should be kept in an archive maintained by the organization which collected that particular text.[17] It may be necessary to leave out some information from material delivered, e.g. formatting, figures, tables, etc. In other cases, one single source file may contain a longer text that has to be chopped into smaller chunks. Being able to locate the source file ensures that certain completions or corrections can be made to the CTB file at a later point in time, if necessary.

3.2.2 The encoding description

The second major component of the TEI header is the encoding description <encodingDesc>. This contains information about the relationship between an encoded text and its original source.

The CTB <encodingDesc> element has the following sub-elements:

<samplingDecl> (sampling declaration) contains a description of the method used in sampling the text.

<projectDesc> (project description) describes the aim or purpose for which an electronic file was encoded.

<appInfo> (application information) records information about the applications which have processed the text of the TEI file.

3.2.2.1 The sampling declaration

The <samplingDecl> element gives an indication of how the text was sampled, the indication is put in an <ab> element. The indication is a string from a fixed set.

[16] It might seem weird to place the URI of a text here. However, as there does not seem to be another adequate element to put this information, common practice obviously is to do it in this manner, see http://colab.mpdl.mpg.de/mediawiki/TEI_Bibliographic_Information.

[17] In the case of DK-CLARIN WP 2.1 all original texts are kept on the ja-korpus.dsl.lan server under /Volumes/Data/textrepository.

It must always be completely identical to the initial caption given in the <title> of <titleStmt>, see Section 3.2.1.1.

```
<samplingDecl>
  <ab>CTB version of:</ab>
</samplingDecl>
```

3.2.2.2 The project description

The <projectDesc> element gives an indication of the aim of collecting and encoding that particular text, i.e the corpus or text collection project or process:

```
<projectDesc>
  <ab>projectIdentifier</ab>
</projectDesc>
```

In the case of new texts captured by WP 2.1 of the DK-CLARIN project, the value of *projectIdentifier* is "DK-CLARIN-WP2.1". Similar fixed contents are defined for other relevant DK-CLARIN projects and for other finished projects like DDOC or KORPUS 2000, see Section 3.3.

3.2.2.3 Application information

The <appInfo> element gives information about all applications or other (manual) procedures by which the text sample has been enriched with markup. The header itself may also be manipulated by such applications or procedures, but this is not registered in the <appInfo> element – this may however be recorded under <revisionDesc>, see Section 3.2.4. The application information helps determining whether texts are structurally comparable, i.e. texts that have been processed by the same bundle of applications and procedures should be structurally identical.

The <appInfo> element should be filled in with one empty dummy-application if the file just contains the default-segmented (i.e. pre-tokenized) version of the text, the so-called *base version*, however the whole <appInfo> structure may be left out in this case as well.[18] The following example shows an <appInfo> with one empty dummy-application. The values given are explained further in Section 3.3.2.

```
<appInfo>
  <application xml:id="app_nil"
    type="nil"
    subtype="nil"
    ident="nil"
```

[18] Leaving <appInfo> out is recommended by DK-CLARIN WP 5.

```
      version="99999999"
      n="nil"
      when="99999999">
      <desc>nil</desc>
      <ptr target="#app_nil"/>
      <ref target="#opt_nil"/>
    </application>
  </appInfo>
```

Otherwise, there is one `<appInfo>` element for each *annotation layer* belonging to the text in the file, see 4. The general structure is as follows:

```
<appInfo>
  <application xml:id="appXmlId"
    type="appType"
    subtype="appTool"
    ident="appId"
    version="appVersionNumber"
    n="appMode"
    when="appDate">
    <desc>appDesc</desc>
    <ptr target="#appXmlId"/> (may be left out)
    <ref target="#appOptionFile"/> (optional)
  </application>
</appInfo>
```

The `<application>` element has the following attributes:

xml:id unique XML identifier which is referenced by the corresponding annotation layer in the text.

type specifies both the task (segmentation, annotation) and whether it was performed by an automatic application or a manual procedure (or a combination of both).

subtype gives a further description of the applied tool taken from a fixed list of options.

ident supplies a unique identifier for the application/procedure.

version supplies a version number for the application/procedure. The version specification may contain other characters than digits, however it must match the following regular expression:
`[\d]+[a-z]*[\d]*(\.[\d]+[a-z]*[\d]*){0,3}`.[19]

[19] It may seem weird to apply version numbers to manual procedures. However, the *version* attribute is mandatory in TEI and also manual procedures may alter over time and should in any case be thoroughly documented – that is versioned.

n gives supplementary info about the applied tag set or tokenization mode.

when gives the date when the application was executed on the text.

The <application> element contains an element <desc> giving a free-text description of the application.

The element <ptr> within <application> references that/those application/applications whose output has been used as input for the application in question as annotations can be added as layers on each other, cf. Chapter 4. This element is left out if an annotation refers to the base version of the text and not to another annotation layer.

Finally, the optional <ref> element may reference certain resources a given tool has been using in cases where this is important.

3.2.3 The profile description

The third component of a TEI header is the profile description <profileDesc>. In the CTB, this is used to provide the following elements:

<creation> contains information about the creation of a text.

<langUsage> (language usage) describes the languages, sublanguages, registers, dialects etc. represented within a text.

<textDesc> (text description) provides a description of a text in terms of its situational parameters.

<textClass> (text classification) groups information which describes the nature or topic of a text in terms of a standard classification scheme, thesaurus, etc.

<particDesc> (participation description) describes the identifiable speakers, voices, or other participants in a linguistic interaction.

3.2.3.1 Text creation

The element <creation> is provided to record details of a text's creation, in the CTB header just the date it was *composed*, i.e. writing on it was finished; it should not be confused with the <imprint> element, where the date of the publication of the (source) text is recorded. In many cases the date, that is the year when a text was finished, is not known: in these cases the date is set to the same as under <imprint> and the value of the attribute *cert* is set to "low" instead of "high". Here is the patten:

```
<creation>
  <date when="textCreationYear" cert="certainty"/>
</creation>
```

3.2.3.2 Language usage

The <languageUsage> element contains the element <language> where the (dominant) language of the text is indicated by the attribute *ident*. Language codes are constructed as defined in BCP 47[20], the language notation standard to use should be ISO 639-1[21].[22] Particularly for sublanguages, an informal prose characterization should be supplied as content for the element. Language usage is expressed by the following XML pattern:

```
<langUsage>
  <language ident="languageId">
    languageCharacterization
  </language>
</langUsage>
```

3.2.3.3 Text description

The overall intention of using this part of the TEI proposal is to establish a structure that can contain text descriptions which can be applied to *every* potential corpus text. The structure is considered general and mandatory for every text in the CTB and information from this structure can be used to extract corpora from the CTB. Specialized textual information, which only may apply to *some* texts, is gathered in the <textClass> part of the header, see Section 3.2.3.4. Also, the amount of specialized textual information may vary from text to text.

The <textDesc> element characterizes each text according to the following eight situational parameters, each represented by one of the following eight elements:

<channel> (primary channel) describes the medium or channel by which a text is delivered or experienced. For a written text, this might be print, manuscript, e-mail, etc.; for a spoken one, radio, telephone, face-to-face, etc. The *mode* attribute describes the mode of the channel with respect to speech or writing.

<constitution> describes the internal composition of a text or text sample, for example as fragmentary, complete, etc.

[20] http://tools.ietf.org/html/bcp47

[21] http://www.sil.org/iso639-3/codes.asp. OBS! Select *View by 639-1*.

[22] At first glance, ISO 639-3 may seem a better choice as it provides more than 6900 language codes, also for dialects and historic languages. However, Danish seems only weakly represented in this standard. Danish authorities should probably get more involved in this standardization work. For DK-CLARIN purposes some of the private areas of this standard could be utilized. Maybe an issue for DK-CLARIN WP 1? Therefore, in the current headers, additional linguistic information may be given in a private BCP 47 extension with regional and historical tags (which needs to be defined).

<derivation> describes the nature and extent of originality of this text, that is, in the CTB header, just an indication of whether it has been translated from another language.

<domain> (domain of use) describes the most important social context in which the text was realized or for which it is intended, for example education, religion, business etc.

<factuality> describes the extent to which the text may be regarded as imaginative or non-imaginative, that is, as describing a fictional or a non-fictional world.

<interaction> describes the number of those producing and experiencing the text.

<preparedness> describes the extent to which a text may be regarded as prepared or spontaneous

<purpose> characterizes a single purpose or communicative function of the text, e.g. whether it is informative, expressive, etc.

By default, a text description will contain each of the above elements, supplied in the order specified. In the CTB, the <textDesc> pattern looks as follows:

```
<textDesc>
  <channel mode="tdChannelMode">tdChannel</channel>
  <constitution type="tdConstitutionType"/>
  <derivation type="tdDerivationType">
    <lang>languageId</lang>
  </derivation>
  <domain type="tdDomainDiscourse">tdDomain</domain>
  <factuality type="tdFactualityType"/>
  <interaction active="tdInteractActive"
    passive="tdInteractPassive">
    <note type="interactRole">tdInteractRole</note>
    <note type="interactAge">tdInteractAge</note>
  </interaction>
  <preparedness type="tdPrepType"/>
  <purpose type="tdPurposeType"/>
</textDesc>
```

Some of the elements given in the <textDesc> pattern contain further specified information:

The <derivation> element has a subordinate element <language> which indicates the original language of the text; if the text is not translated,

the original language is identical to that indicated under <langUsage>, see Section 3.2.3.2.

The <interaction> element contains two subordinate <note> elements, one of them indicating the roles of the participants in the communication, that is, whether they are experts or laymen; the other <note> element gives the ages of addressor and addressee. Using a <note> element for giving further interaction-related information is not an optimal solution. A straighter way is to use special elements for the needed purposes or to augment the attribute list of the <interaction> element. However, this would require a modification of the TEI grammar.

More info on this part of the header can be found in Section 3.3.

3.2.3.4 Text classification

Texts may be described along many dimensions, according to many different taxonomies. No generally accepted consensus as to how such taxonomies should be defined has yet emerged. To accommodate special needs, TEI allows to express more specialized text characteristics by the following elements:

<catRef> (category reference) provides either a list of codes or one single code identifying the categories to which the text has been assigned, each code referencing a category element declared in the corpus header or under a separate, invariant URL. In CTB, there is one <catRef> element for each dimension, the type of dimension is indicated by the (referencing) value of the attribute *scheme*. CTB does not use lists of codes.

<classCode> contains the classification code used for the text in some standard classification system. There is one <classCode> element for each classification system.

Using <catRef> is the preferred way to give additional textual classifications in all cases where the classification system follows a CTB-internal standard. The pattern to be applied is as follows:

```
<textClass>
  <catRef scheme="myClassification" target="myValue"/>
</textClass>
```

The <catRef> element is repeated for each classification dimension used. If several values are given within the same classification dimension, <catRef> elements with the same classification scheme are repeated.

In cases where an official classification system is applied, the <classCode> element is used instead. More values within the same scheme are given by repeating <catRef> elements. The <catRef> and <classCode> elements should be used according to the following, invented, example:

```
<textClass>
  <catRef scheme="dk-clarin.eu/ctb/agerel" target="#a-c"/>
  <catRef scheme="dk-clarin.eu/ctb/domain" target="#med"/>
  <catRef scheme="dk-clarin.eu/ctb/domain" target="#bio"/>
  <catRef scheme="dk-clarin.eu/ctb/genre" target="#ad"/>
  <classCode scheme="official.classfication.eu">xyz</classCode>
</textClass>
```

3.2.3.5 The participant description

The participant description (<particDesc>) element is used to provide additional information about authors (or speakers) of texts. The element itself is considered obligatory in the CTB header, however, its contents may just be an empty <person> element which is given as a placeholder to ensure that the header has a valid TEI structure. If additional personal info is given, one <person> element for each participant having been involved in creating the text is inserted into <particDesc>.[23] The <person> element carries a number of attributes which are used to provide encoded values for some key aspects of the person concerned, see the following example:[24]

```
<particDesc>
  <person xml:id="personId"
    role="creatorRole"
    age="creatorAge"
    sex="creatorSex">
    <birth>
      <date when="creatorBirth" cert="certainty"/>
    </birth>
  </person>
</particDesc>
```

The DDOC material mentioned in Section 3.1 has a lot more information on each text creator, e.g. his place of birth which could be expressed as an element <placeName> under <birth>, his place of residence which could be put into an element <residence> as sibling to <birth>, and so on. However, corpus-linguistic practice has shown that this type of information hardly ever is used (nor useful if it is not given according to clear-cut classification schemes). Therefore, new material should not be marked-up with this kind of information that is also extremely costly to gather. For DDOC (and other material) which already carries this type of information, appropriate structural elements of <person> should be included into the header to allow keeping this information for possible future investigation, see Chapter **??**.

[23] A possible empty placeholder <person> element may then be deleted.

[24] More details of which values to fill in can be found in Section 3.3.

3.2.4 The revision description

A list of typical revisions which a document will undergo should be created, i.e. values for *revisionType*. At least the revision type "Document created" seems important. Others, which deal with the completeness of the header may be useful as well. The pattern of the revision description is as follows:

```
<revisionDesc>
  <change when="revisionDate"
    who="organizationName">revisionType
  </change>
</revisionDesc>
```

The revision description must not be confused with the application information discussed in Section 3.2.2.3.

3.3 Filling in the header

3.3.1 Full header template

In the following, a complete version of the CTB header template is shown. Its four main constituents and their subdivisions are separated by horizontal lines to facilitate orientation:

```
<teiHeader type="text">
```

<fileDesc>

```
  <fileDesc>
```

<titleStmt>

```
    <titleStmt>
      <title>samplingDeclaration textTitle</title>
      <sponsor>sponsorName</sponsor>
      <respStmt>
        <resp>Data capture</resp>
        <name>organizationName
          <note type="method">captureMethod</note>
          <date when="captureYear"/>
        </name>
      </respStmt>
    </titleStmt>
```

<extent>

```
    <extent>
      <num n="words">numberOfWords</num>
      <num n="paragraphs">numberOfParagraphs</num>
    </extent>
```

<publicationStmt>

```
    <publicationStmt>
      <distributor>organizationName</distributor>
      <idno type="textIdType">textId</idno>
      <availability status="availStatus">
```

```
      <ab type="academic">
        <seg type="availDesc">availDesc</seg>
        <seg type="anonymDesc">anonymDesc</seg>
      </ab>
      <ab type="nonCommercial">
        <seg type="availDesc">availDesc</seg>
        <seg type="anonymDesc">anonymDesc</seg>
      </ab>
      <ab type="all">
        <seg type="availDesc">availDesc</seg>
        <seg type="anonymDesc">anonymDesc</seg>
      </ab>
    </availability>
  </publicationStmt>
```

<notesStmt>

```
  <notesStmt>
    <note xml:lang="languageId"
      resp="organizationName">note</note>
  </notesStmt>
```

<sourceDesc>

```
  <sourceDesc>
    <biblStruct>
      <analytic>
        <title xml:lang="languageId"
          level="titleLevel">textTitle</title>
        <author>
          <name ref="#personId">surname, forename</name>
        </author>
        <respStmt n="translators">
          <resp>Translated by</resp>
          <name ref="#personId">surname, forename</name>
        </respStmt>
      </analytic>
      <monogr>
        <title xml:lang="languageId">editionTitle</title>
        <editor>
          <name ref="#personId">surname, forename</name>
        </editor>
        <imprint>
          <publisher n="publId">publHouse</publisher>
          <date when="publDate" cert="certainty"/>
          <biblScope type="issue">edIssue</biblScope>
          <biblScope type="sect">edSect</biblScope>
          <biblScope type="vol">edVolume</biblScope>
          <biblScope type="chap">edChapter</biblScope>
          <biblScope type="pp">edPages</biblScope>
        </imprint>
      </monogr>
      <idno type="uri">textUri</idno>
      <idno type="file">textFileName</idno>
      <relatedItem type="relatedType">
        <bibl>
          <title xml:lang="languageId">relatedTitle</title>
```

```
          <idno type="ctb">relatedId</idno>
        </bibl>
      </relatedItem>
    </biblStruct>
  </sourceDesc>
</fileDesc>
```

<encodingDesc>

```
<encodingDesc>
```

<samplingDecl>

```
  <samplingDecl>
    <ab>samplingDeclaration</ab>
  </samplingDecl>
```

<projectDesc>

```
  <projectDesc>
    <ab>projectIdentifier</ab>
  </projectDesc>
```

<appInfo>

```
  <appInfo>
    <application xml:id="appXmlId"
      type="appType"
      subtype="appTool"
      ident="appId"
      version="appVersion"
      n="appMode"
      when="appDate">
      <desc>appDesc</desc>
      <ptr target="#appXmlId"/>
      <ref target="#appOptionFile"/>
    </application>
  </appInfo>
</encodingDesc>
```

<profileDesc>

```
<profileDesc>
```

<creation>

```
  <creation>
    <date when="textCreationYear" cert="certainty"/>
  </creation>
```

<langUsage>

```
  <langUsage>
    <language ident="languageId">
      languageCharacterization
    </language>
  </langUsage>
```

<textDesc>

```
  <textDesc>
    <channel mode="tdChannelMode">tdChannel</channel>
    <constitution type="tdConstitutionType"/>
    <derivation type="tdDerivationType">
      <lang>languageId</lang>
    </derivation>
    <domain type="tdDomainDiscourse">tdDomain</domain>
    <factuality type="tdFactualityType"/>
```

```
      <interaction active="tdInteractActive"
        passive="tdInteractPassive">
        <note type="interactRole">tdInteractRole</note>
        <note type="interactAge">tdInteractAge</note>
      </interaction>
      <preparedness type="tdPrepType"/>
      <purpose type="tdPurposeType"/>
    </textDesc>
```

<textClass>

```
    <textClass>
      <catRef scheme="myClassification" target="myValue"/>
      <classCode scheme="theirClassification">theirValue</classCode>
    </textClass>
    <particDesc>
      <person xml:id="personId"
        role="creatorRole"
        age="creatorAge"
        sex="creatorSex">
        <birth>
          <date when="creatorBirth" cert="certainty"/>
        </birth>
      </person>
    </particDesc>
  </profileDesc>
```

<revisionDesc>

```
  <revisionDesc>
```

<change>

```
    <change when="revisionDate"
      who="organizationName">revisionType
    </change>
  </revisionDesc>
</teiHeader>
```

3.3.2 Value sets for header standard information

When filling in the header with standard information about the text, some types of information may be undetermined or non-existent, e.g. the name of an author may be simply missing in the header for some reason, that is, it is *undetermined*, or a text may not have a title, that is, its title is *non-existent*. Such incomplete parts of the header could be left out in these cases if permitted by TEI, however, leaving out such parts would obscure whether the information is missing because it is undetermined or because it is non-existent. If the information is undetermined, efforts should be undertaken to occasionally add it, otherwise, if it is non-existent, such efforts would be waste of time. In order to distinguish these two cases, it is recommended to always explicitly state non-existent information by filling in *empty* for string and symbol values, *0* (= zero) for integers, and *1000* in the case of years (and dates),[25] in other words never to leave these parts of a header out. However,

[25] The value *1000* for dates is necessary in order to comply with the TEI data type `date` that does not allow a value of *0*.

if the information is undetermined, these parts of a header may be left out indicating that the missing information occasionally should be added or be marked as non-existent if that is the case.

So in the case of undetermined information, it is legal to skip the respective part of the header if allowed by TEI; however, for the sake of completeness, it is strongly recommended to state *nil* in case of string values and *99999999*[26] in the case of integers and dates to indicate that this particular information obviously is missing and should be added if it does exist or, if it turns out that the information definitely does not exist, it should be marked as non-existent. To sum up, the following constant symbols are used as values for header elements and attributes, unless otherwise stated further below in this section:[27]

Symbol	Type	Meaning
empty	String	Info is non-existent
anonymous	Names	Person is unknown
0	Integer	Info is non-existent
1000	Date/Year	Info is non-existent
nil	String	Info has not been determined yet
99999999	Integer and Date/Year	Info has not been determined yet

In all other cases, that is in cases where the desired information is available, the values listed in Section 3.3.2.1 are used replacing the header variables indicated in the full header template above. For each of these variables a description is given followed by an overview of its properties and – in the case of enumerated sets – a list of legal values. In cases where these lists are too comprehensive, they are replaced by a link to an XML version of them. All value sets are also accessible as XML files and may be referenced automatically or manually when filling in headers. All value set files are found under the path `http://korpus.dsl.dk/clarin/corpus-doc/text-header/`. The filenames themselves are given below.[28] The structure of the XML value set files is as shown in the following extract. The structure has been designed for this specific purpose (i.e. it is not TEI) and it should be fairly self-explanatory:

[26] In former versions of the documentation the 'undetermined' value was *1* (minus one). However, TEI does not always allow a negative value for some of its integer datatypes which is the reason why it has been replaced.

[27] In cases where TEI does not allow the undetermined/non-existent values defined here, the elements of the value sets are restricted to those that are accepted by TEI. This is the case for the following attributes: `cert` in `<date>`, `sex` in `<person>`, `mode` in `<channel>`, `type` in `<factuality>`, `level` in `<title>`.

[28] As these are XML files, a web browser may not show them well formatted. Viewing them as HTML *source* may help though.

```
<?xml version="1.0" encoding="UTF-8"?>
<valuesetCollection
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation=
        "http://korpus.dsl.dk/clarin/corpus-doc/
        text-header/valuesetCollection.xsd">
    <set name="captureMethod" type="symbol">
        <element>
            <value>nil</value>
            <desc>Info has not been determined yet</desc>
        </element>
        <element>
            <value>empty</value>
            <desc>Info is irrelevant, non-existent, or undeterminable</desc>
        </element>
        <element default="true">
            <value>file</value>
            <desc>The source of the text is an electronic file</desc>
        </element>
        <element>
            <value>ocr-raw</value>
            <desc>The text is OCR-scanned but not proof-read</desc>
        </element>
        <element>
            <value>ocr-proof</value>
            <desc>The text is OCR-scanned and proof-read</desc>
        </element>
        <element>
            <value>keyed-raw</value>
            <desc>The text is manually keyed but not proof-read</desc>
        </element>
        <element>
            <value>keyed-proof</value>
            <desc>The text is manually keyed and proof-read</desc>
        </element>
        [...]
    </set>
</valuesetCollection>
```

The following properties are given for each value set:

1. The ***value set type*** gives an indication of whether the set of values is meant to be augmented or not. It may be

   enumerated, closed, which means that no further values should be added to it

   enumerated, open, meaning that one can add further values if necessary

   Open and *closed* is a distinction only relevant to enumerated, i.e. extensionally defined sets, whereas sets whose contents are intentionally defined, i.e. by description, as a matter of fact always are open:

descriptive can contain any description that observes the definition of the set

2. The ***XML URL*** is a URL that points to an XML version of the value set (only applicable for extensional value sets)

In some cases, properties are indicated as "undetermined" which means that this information still is missing for some reason and should be added in a future version of this chapter.

In other cases, properties are indicated as "n/a" as not applicable.

3.3.2.1 Alphabetical list of value sets

Note that some value sets are still empty as the properties they describe have not been relevant meta-info yet. Many others may still be augmented with additional values. Please refer to the most recent version of this chapter which can be downloaded as a technical report from http://korpus.dsl.dk/clarin/corpus-doc/text-header.pdf.

▷ ***anonymDesc***

Indicator specifying what type(s) of private text information must be made anonymous (= must not be shown).

Properties

Value set type	enumerated, closed
XML name	vs_anonymDesc.xml

Legal values

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
0	Nothing in the text or associated with the text must be made anonymous**. Default**
I	Names of individuals must not be shown
P	Names of places must not be shown
A	Name(s) of the author(s) must not be shown
T	Text title must not be shown

The values can be combined if more of them apply to a specific user group, e.g. "IA" means that names of individuals and of the author(s) must be made anonymous.

▷ ***appDate***

The date a particular markup application/procedure was applied to the text.

Properties

Value set type	descriptive
XML name	n/a

Legal values Dates must follow the pattern *yyyy-mm-dd*.

▷ ***appDesc***

Free-text description of the application/procedure that has operated on the text.

Properties

Value set type	descriptive
XML name	n/a

Legal values Any string.

▷ ***appId***

Unique version name-independent identifier of an application/procedure that has operated on the text.

Properties

Value set type	enumerated, open
XML name	vs_appId.xml

Legal values

Value	Description
nil	Info has not been determined yet. **Default**
empty	Info is irrelevant, non-existent, or undeterminable
LocalInfoMediaConverter	Converts Infomedia text to CTB base format with simple headers

▷ ***appMode***

Info about the applied tag set, tokenization mode, or configuration.

Properties

Value set type	enumerated, open
XML name	vs_appMode.xml

Legal values

Value	Description
99999999	Info has not been determined yet
0	Info is irrelevant, non-existent, or undeterminable
da-xxx	Danish language model version applied

▷ ***#appOptionFile***

XML pointer to information on the setup of the tool that has processed the text.

Properties

Value set type	descriptive
XML name	n/a

Legal values Any string that can be used for unique XML-referencing.

▷ ***appTool***

Describes the (automatic or manual) tool that has operated on the text.

Properties

Value set type	enumerated, closed
XML name	vs_appTool.xml

Legal values

Value	Description
nil	Info has not been determined yet**. Default**
empty	Info is irrelevant, non-existent, or undeterminable
pretokenizer	Splits a text into word-like segments. A pretokenizer is only applied once, all other applications are based on the pretokenized version of the text
tokenizer	Splits a text into word-like segments
s-splitter	Sentence splitter. Splits the text into sentences, i.e. a segment between two full stops or some similar type of punctuation. Inserts <s> and </s> tags around sentence-like text segments
p-splitter	Paragraph splitter. Splits the text into paragraphs. Inserts and tags around paragraph-like text segments
regularizer	Tags a token with a regularised version of its surface representation, i.e. its orthography
lemmatizer	Tags a token with its lemma form
pos-tagger	Tags a token with part-of-speech info
morph-tagger	Tags a token with morphological/inflectional info
term-tagger	Tags a token with some indication of whether it is a term (in texts to be included in LSP corpora)
multi-processor	Multifunctional tool that performs various tasks like tokenizing, lemmatizing, tagging as one complex process
other	Tool performing tasks not yet listed

▷ ***appType***

Specifies whether an application or procedure that operated on the text was automatic (or a combination of both) as well as the type of task of the application/procedure in terms of segmentation or annotation.

Properties

Value set type	enumerated, closed
XML name	vs_appType.xml

Legal values

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
a_segmentation	Text split into smaller segments (e.g. sentences, tokens) by an automatic process. **Default**
c_segmentation	Text split into smaller segments (e.g. sentences, tokens) by a combined automatic-manual process
m_segmentation	Text split into smaller segments (e.g. sentences, tokens) by a manual process
a_annotation	Text segments annotated with info by an automatic process
c_annotation	Text segments annotated with info by a combined automatic-manual process
m_annotation	Text segments annotated with info by a manual process

▷ ***appVersion***

Version specification of an application/procedure that has operated on the text.

Properties

Value set type	descriptive
XML name	n/a

Legal values The version specification must start with at least one digit but may contain other characters than digits. It must match the following regular expression :
`[\d]+[a-z]*[\d]*(\.[\d]+[a-z]*[\d]*){0,3}.`

▷ ***appXmlId***

Unique XML identifier which is referenced by the corresponding annotation layer (`<spanGrp>` element, see 4.2.3.2) in the text.

Properties

Value set type	descriptive
XML name	n/a

Legal values Valid XML IDs are constructed by concatenating the *appId*, an underscore, and the *appVersion* where dots are replaced by underscores.

▷ ***availDesc***

Tells how this text may be used in terms of copyright and other restrictions.

Properties

Value set type	enumerated, closed
XML name	vs_availDesc.xml

Legal values

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
full	The user has free access to the complete material, but is not allowed to redistribute it
partial	The user can search and view text contents limited to what is specified in Danish citation law**. Default**
limited	Access only upon written agreement between the DK-CLARIN consortium and the user. Details of this agreement are to be further specified
none	No acces for users not affiliated with the DK-CLARIN consortium

▷ ***availStatus***

Attribute of the `<availability>` element indicating whether the text is freely available for all user categories (cf. the header template above) or not.

Properties

Value set type	enumerated, closed
XML name	vs_availStatus.xml

Legal values

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
free	The text is freely available for all user categories
restricted	The text is not freely available for at least one user category**. Default**

▷ ***captureMethod***

The method of data capture.

Properties

Value set type	enumerated, closed
XML name	vs_captureMethod.xml

Legal values

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
file	The source of the text is an electronic file**. Default**
ocr-raw	The text is OCR-scanned but not proof-read
ocr-proof	The text is OCR-scanned and proof-read
keyed-raw	The text is manually keyed but not proof-read
keyed-proof	The text is manually keyed and proof-read
double-keyed	The text is double-keyed, i.e. keyed in two versions by two individual typists, both versions are automatically compared and manually corrected
pdf-converted-acrobat9	Converted from PDF by Acrobat 9
pdf-converted-pdf2xml	Converted from PDF by pdf2xml

▷ ***captureYear***

The year of data capture.

Properties

Value set type	descriptive
XML name	n/a

Legal values Four-digit years which may be extended to full dates following the pattern *yyyy-mm-dd*.

▷ ***certainty***

The degree of certainty of how precise some data, typically dates, are.

Properties

Value set type	enumerated, closed
XML name	vs_certainty.xml

Legal values

Value	Description
high	The given dates are definitely correct. **Default**
low	The given dates are an estimate

▷ ***creatorAge***

The age group to which a particular author belonged at the time he/she produced the text.

Properties

Value set type	enumerated, closed
XML name	vs_creatorAge.xml

Legal values The age intervals are inevitably arbitrary. The "teen" interval is consciously extended to the age of 25 to be able to better indicate young people's language in general. See also TEI P5.[29]

[29] http://www.tei-c.org/release/doc/tei-p5-doc/html/ref-person.html

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
infant	A person aged 0–5
child	A person aged 6–12
teen	A person aged 13–25
adult	A person aged 26–60**. Default**
senior	A person aged 61 and above

▷ ***creatorBirth***

The year a particular author was born.

Properties

Value set type	descriptive
XML name	n/a

Legal values Four-digit date following the pattern *yyyy*.

▷ ***creatorRole***

The role of a particular author in terms of his or her influence on the language of the text.

Properties

Value set type	enumerated, closed
XML name	vs_creatorRole.xml

Legal values For written texts:[30]

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
major	Assigned to one single autor, translator, or editor who is assumed to have had major impact on the language of the text**. Default**
minor	Assigned to all other textual contributors

[30] The list may be augmented with values for spoken texts from the DDOC.

There should only be one author, translator, or editor with “major” influence. All other contributors should be classified “minor”.

▷ ***creatorSex***

The sex of a particular author.

Properties

Value set type	enumerated, closed
XML name	vs_creatorSex.xml

Legal values From ISO 5218:1977 Representation of Human Sexes to comply with TEI, see `http://www.tei-c.org/release/doc/tei-p5-doc/html/ref-data.sex.html`. OBS! The values for *undetermined* (“nil”) and *n/a* (“empty”) differ from the CTB standard values.

Value	Description
0	Unknown. **Default**
1	Male
2	Female
9	Not applicable

▷ ***edChapter***

The chapter of a book or similar edition from which the text sample is taken.

Properties

Value set type	descriptive
XML name	n/a

Legal values Any integer.

▷ ***edIssue***

The issue of a newspaper or journal from which the text sample is taken.

Properties

Value set type	descriptive
XML name	n/a

Legal values Any string.

▷ ***edPages***

The range of pages the text sample spans over in the edition from which it is taken.

Properties

Value set type	descriptive
XML name	n/a

Legal values Any integer or an interval of integers according to the pattern: *x*–*y* where $y > x$. Groups of intervals are not allowed. Each text sample in the CTB must be coherent. If several samples are taken from the same text source, each of them has to be put into a CTB file of its own.

▷ ***edSection***

The section of a newspaper from which the sample is taken.

Properties

Value set type	descriptive
XML name	n/a

Legal values Any string.

▷ ***edVolume***

The volume of a book from which the text sample is taken.

Properties

Value set type	descriptive
XML name	n/a

Legal values Any integer.

▷ ***editionTitle***

The title of the edition (e.g. book, newspaper) in which the text appeared.

Properties

Value set type	descriptive
XML name	n/a

Legal values Any string.

▷ ***fileCreationYear***

The year the electronic text sample was created.

Properties

Value set type	descriptive
XML name	n/a

Legal values Four-digit date which may be extended to a full date following the pattern *yyyy-mm-dd*.

▷ ***forename***

First name(s) of a text's author/editor/translator.

Properties

Value set type	descriptive
XML name	n/a

Legal values Any string. Names are always given as a string of pattern *surname, forename* in `<name>` elements. If the name cannot be decomposed into forename and surname, the name is stated without a comma. If the text has been written/translated/edited by a company or organization, the name of that company/organization is stated. If the name for some reason is anonymous, the `<name>` element is filled in with the string "anonymous".

▷ ***languageCharacterization***

Prose description of the language indicated by *languageId*.

Properties

Value set type	descriptive
XML name	n/a

Legal values Comma-separated list of the descriptions associated with the values applied in *languageId*, e.g. “Danish” if *languageId* is “da”. See *languageId*.

▷ ***languageId***

Code that identifies the language used in the text sample or in a `<note>` or `<title>` tag.

Properties

Value set type	enumerated, open
XML name	vs_langSubId.xml

Legal values Values follow BCP 47[31] and ISO 639-1.[32] The language code is constructed according to BCP 47 as follows:

langSubId [- x [- *langSubHist*] [- *langSubRegion*]]

It consists of an obligatory part with a language code *langSubId* according to ISO 639-1 and an optional private extension, prefixed by the BCP 47 subtag *x* that holds a code *langSubHist* for the historic period of the language in question, and another optional part with a regional code *langSubRegion*. If both optional parts are present, they must come in the order specified.

Legal values for *langSubId* are defined in the following subset of the ISO 639-1 standard, however the non-standard value “xx” has been added to indicate formalized language that may occur in the content of `<note>` elements, see an example in Chapter **??**.

Value	Description
nil	Info has not been determined yet (not part of ISO 639-1). **Default**
empty	Info is irrelevant, non-existent, or undeterminable (not part of ISO 639-1)
da	Danish
de	German
en	English
es	Spanish
fr	French
xx	Formalized or constructed (not part of ISO 639-1)

[31] http://tools.ietf.org/html/bcp47
[32] http://www.sil.org/iso639-3/codes.asp. OBS! Select *View by 639-1*.

For each *langSubId*, that is for each language, a set of *langSubHist* and *langSubRegion* codes can be defined; for each language the name of the *langSubHist* and *langSubRegion* variables is extended with the ISO 639-1 code of the language in question, e.g. *langSubHistDa* or *langSubRegionDa* for Danish. Legal values must be defined according to the pattern "hCode" for historic codes and "rCode" for region codes, the "h" and the "r" indicating *historic* and *region* respectively, whereas the "Code" part contains the code to be used for a certain period or region. Currently no such codes are defined for any language within the CTB framework.

▷ ***myClassification***

URL of a user-defined text classification.

Properties

Value set type	enumerated, open
XML name	vs_myClassification.xml

Legal values Any valid URL pointing to a classification scheme. Currently, the following classification scheme URLs are defined:

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
http://ctb.dsl.dk/class/catRef/DDOC/RePr.xml	Synsvinkel (produktion, reception)
http://ctb.dsl.dk/class/catRef/DDOC/Medi.xml	Medium, channel
http://ctb.dsl.dk/class/catRef/DDOC/Genr.xml	Genre, text type
http://ctb.dsl.dk/class/catRef/DDOC/GnTy.xml	Genre type (simplified genre classification)
http://ctb.dsl.dk/class/catRef/infomedia/PSIN.xml	Infomedia PSIN topic labels

▷ ***myValue***

Value given in a user-defined text classification.

Properties

Value set type	enumerated, open
XML name	n/a

Legal values Legal values according to the user-defined classification.

▷ ***note***

Any note giving additional information about the text which cannot be expressed by other elements in the header.

Properties

Value set type	descriptive
XML name	n/a

Legal values Any string.

▷ ***numberOfParagraphs***

The number of paragraphs in the text.

Properties

Value set type	descriptive
XML name	n/a

Legal values Any integer.

▷ ***numberOfWords***

The number of word-like units, i.e. <w> elements, in the text.

Properties

Value set type	descriptive
XML name	n/a

Legal values Any integer.

▷ ***organizationName***

The name of an organization that has carried out some particular piece of work or had some particular responsibility related to the electronic text sample.

Properties

Value set type	enumerated, open
XML name	vs_organizationName.xml

Legal values

Value	Description
nil	Info has not been determined yet. **Default**
empty	Info is irrelevant, non-existent, or undeterminable
cst.ku.dk	Center for Sprogteknologi, KU
dsl.dk	Det Danske Sprog- og Litteraturselskab
dsn.dk	Dansk Sprognævn
dsl-dsn.dk	DSL og DSN i fællesskab
duds.nordisk.ku.dk	Digitale Undersøgelser af Dansk Sprog, INSS, KU

▷ ***#personId***

Id linking between the name of an author and the `<person>` element in `<textDesc>` giving additional author information.

Properties

Value set type	descriptive
XML name	n/a

Legal values Any string that can be used for unique XML-referencing. The string should contain a sequence of digits.

▷ ***publDate***

The publishing date of the edition in which the text appeared.

Properties

Value set type	descriptive
XML name	n/a

Legal values Values are given either as the year as a four-digit number, or the year, month, and day given according to the pattern *yyyy-mm-dd*.

▷ ***publHouse***

The name of the publisher (company, or if self-published, the author) of the edition in which the text appeared, or the name of the text supplier.

Properties

Value set type	enumerated, open
XML names	vs_publId.xml

Legal values String denoting a publisher/supplier taken from the description part of the lists referred to under *publId* below.

▷ ***publId***

Unique identifier of either publisher or text supplier pointing to an external database of publishers.

Properties

Value set type	enumerated, open
XML names	vs_publId.xml

Legal values Integer according to specified lists maintained by WP 2.1.

Additional publisher/supplier info is found in the resource

- `/db/ctb/suppliers/ctb-suppliers.xml`

in the eXist-db on the `ja-korpus.dsl.lan` server. The *publId*s given in the list above can be seen as pointers to the records with additional supplier info.

▷ ***projectIdentifier***

Unique identifier of the text collection project in which this electronic text was captured and prepared.

Properties

Value set type	enumerated, open
XML name	vs_projectIdentifier.xml

Legal values

Value	Description
nil	Info has not been determined yet**. Default**
empty	Info is irrelevant, non-existent, or undeterminable
DK-CLARIN-WP2.1	LGP corpus project under DK-CLARIN, 2008-2010
DK-CLARIN-WP2.2	LSP corpus project under DK-CLARIN, 2008-2010
DK-CLARIN-WP2.3	Renaissance corpus project under DK-CLARIN, 2008-2010
DK-CLARIN-WP2.4	JVJ/ADL corpus project under DK-CLARIN, 2008-2010
DK-CLARIN-WP2.5	Nationalmuseet's corpus project under DK-CLARIN, 2008-2010
DK-CLARIN-WP2.6	Parallel corpus project under DK-CLARIN, 2008-2010
DSL-DOT	Ongoing DSL-DOT gathering
DSL-DOT-IM	Ongoing DSL-DOT gathering via InfoMedia
DDOC-spoken	Corpus of The Danish Dictionary, transcribed speech
DDOC-written	Corpus of The Danish Dictionary, written
K2000	Material collected in the Korpus 2000 project

▷ ***relatedTitle***

Title of a text related to the current one.

Properties

Value set type	descriptive
XML name	n/a

Legal values Any string denoting a text title.

▷ ***relatedType***

Value stating how the text possibly is related to another text.

Properties

Value set type	enumerated, closed
XML name	

Legal values

Value	Description
nil	Info has not been determined yet**. Default**
empty	Info is irrelevant, non-existent, or undeterminable
noRelated	No related text exists
original	The related text is the original from which the current text has been translated
parallel	It is not known whether the related text is the original or the translation, as may be the case for texts from the EU

▷ ***revisionDate***

Date when a revision was performed on the text item.

Properties

Value set type	descriptive
XML name	n/a

Legal values Year, month, and day given according to the pattern *yyyy-mm-dd*.

▷ ***revisionType***

Standardized type of revision applied to the text item.

Properties

Value set type	enumerated, open
XML name	vs_revisionType.xml

Legal values

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
created	First version of CTB file created**. Default**

▷ ***samplingDeclaration***

Indicates the amount of original text included in the CTB version.

Properties

Value set type	enumerated, closed
XML name	vs_samplingDeclaration.xml

Legal values

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
CTB sample	It is unknown whether the text is complete or abridged**. Default**
CTB version	Complete text is included
CTB excerpt	Continuous excerpt from the original text

▷ ***sponsorName***

The name of the initiative (or organization) that intellectually has supported or initiated the collection of a particular text.

Properties

Value set type	enumerated, open
XML name	vs_sponsorName.xml

Legal values

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
DK-CLARIN	The DK-CLARIN Consortium, 2008-2010**. Default**
ordnet.dk	The Ordnet.dk Project at dsl.dk, 2006-2013
Korpus 2000	The Korpus 2000 Project at dsl.dk, 2000-2002
DDO	Den Danske Ordbog at dsl.dk, 1991-2005

▷ ***surname***

Last name of a text's author/editor/translator.

Properties

Value set type	descriptive
XML name	n/a

Legal values Names are always given as a string of pattern *surname, forename* in `<name>` elements. If the name cannot be decomposed into forename and surname, the name is stated without a comma. If the text has been written/translated/edited by a company or organization, the name of that company/organization is stated. If it for some reason is anonymous, the `<name>` element is filled in with the string "anonymous".

▷ ***tdChannel***

The primary channel/medium by which a text is delivered or experienced.

Properties

Value set type	enumerated, open
XML name	vs_tdChannel.xml

Legal values Generally, a text can either be written or spoken. If it is written, it can either be distributed electronically, e.g. on the Internet, or on paper, e.g. as a book. The following table is only rudimentary, but shows the principle of coding: The first digit from the left indicates the general channel which can be further specified by adding further digits, e.g. "2" means written, "22" means written using an electronic channel, "221" might mean email, etc.

Value	Description
99999999	Info has not been determined yet**. Default**
0	Unknown channel
1	Spoken
121	Radio
122	TV
2	Written
21	Paper
22	Electronic

▷ ***tdChannelMode***

Describes the channel/medium of a text with respect to speech or writing.

Properties

Value set type	enumerated, closed
XML name	vs_tdChannelMode.xml

Legal values Values follow the TEI specifications:[33]

Value	Description
w	Written. **Default**
s	Spoken
sw	Spoken recorded by writing it down
ws	Written meant to be spoken
m	Mixed
x	Unknown or inapplicable. OBS! TEI mixes two cases which usually are kept apart in CTB

▷ ***tdConstitutionType***

Describes the internal composition of a text or text sample, for example as fragmentary or complete.

Properties

Value set type	enumerated, closed
XML name	vs_tdConstitutionType.xml

Legal values Legal values make up a subset of the TEI specifications:[34]

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
single	A single complete text. **Default**
frags	The text is a continuous fragment, e.g. a chapter from a novel
unknown	It is unknown whether the text is complete or fragmentary

[33] http://www.tei-c.org/release/doc/tei-p5-doc/html/ref-channel.html

[34] http://www.tei-c.org/release/doc/tei-p5-doc/html/ref-constitution.html

▷ ***tdDerivationType***

Describes whether the text is translated or original.

Properties

Value set type	enumerated, closed
XML name	vs_tdDerivationType.xml

Legal values Legal values follow the TEI specifications:[35]

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
original	Original, un-translated version of the text. **Default**
translation	The text is a translation

▷ ***tdDomain***

The domain the text is associated with.

Properties

Value set type	enumerated, closed
XML name	vs_tdDomain.xml

Legal values The full set of 66 DDOC domain values is used, as experiments using it for automatic domain classification were promising, see Asmussen (2005).[36] The 66 values can be looked up in the following XML document: DDOC domain values.

▷ ***tdDomainDiscourse***

Describes whether the discourse is domain-specific or not, i.e. if the type of language used in the text can be categorized as language for general or specific purposes.

Properties

Value set type	enumerated, closed
XML name	vs_tdDomainDiscourse.xml

[35] http://www.tei-c.org/release/doc/tei-p5-doc/html/ref-derivation.html

[36] http://korpus.dsl.dk/staff/ja/papers/cl2005_asmussen.latex.pdf

Legal values

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
general	No domain-specific discourse. Language for general purposes used. **Default**
specific	Domain-specific discourse. Language for specific purposes used

▷ ***tdFactualityType***

Tells whether a text is imaginative or non-imaginative.

Properties

Value set type	enumerated, closed
XML name	vs_tdFactualityType.xml

Legal values Values must conform with the TEI specifications[37] given in the following list:[38]

Value	Description
fiction	The text is to be regarded as entirely imaginative
fact	The text is to be regarded as entirely informative or factual
mixed	The text contains a mixture of fact and fiction
inapplicable	The fiction/fact distinction is not regarded as helpful or appropriate to this text. **Default**

▷ ***tdInteractActive***

The number of addressors having produced the text.

Properties

Value set type	enumerated, closed
XML name	vs_tdInteractActive.xml

[37] http://www.tei-c.org/release/doc/tei-p5-doc/html/ref-factuality.html

[38] TEI does not allow to distinguish between “unknown” and “inapplicable”.

Legal values Values conform to the suggestions made in the TEI specifications.[39]

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
singular	A single addressor**. Default**
plural	Many addressors
corporate	A corporate addressor

▷ ***tdInteractAge***

The age group to which addressor and addressee belong.

Properties

Value set type	enumerated, closed
XML name	vs_tdInteractAge.xml

[39] http://www.tei-c.org/release/doc/tei-p5-doc/html/ref-interaction.html

Legal values

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
infant-infant	A person aged 0–5 addressing another infant
infant-child	A person aged 0–5 addressing a child
infant-teen	A person aged 0–5 addressing a teen
infant-adult	A person aged 0–5 addressing an adult
infant-senior	A person aged 0–5 addressing a senior
child-infant	A person aged 6–12 addressing an infant
child-child	A person aged 6–12 addressing another child
child-teen	A person aged 6–12 addressing a teen
child-adult	A person aged 6–12 addressing an adult
child-senior	A person aged 6–12 addressing a senior
teen-infant	A person aged 13–25 addressing an infant
teen-child	A person aged 13–25 addressing a child
teen-teen	A person aged 13–25 addressing another teen
teen-adult	A person aged 13–25 addressing an adult
teen-senior	A person aged 13–25 addressing a senior
adult-infant	A person aged 26–60 addressing an infant
adult-child	A person aged 26–60 addressing a child
adult-teen	A person aged 26–60 addressing a teen
adult-adult	A person aged 26–60 addressing another adult**. Default**
adult-senior	A person aged 26–60 addressing senior
senior-infant	A person aged 61 and above addressing an infant
senior-child	A person aged 61 and above addressing a child
senior-teen	A person aged 61 and above addressing a teen
senior-adult	A person aged 61 and above addressing an adult
senior-senior	A person aged 61 and above addressing another senior

▷ ***tdInteractPassive***

The number of addressees to whom a text is directed.

Properties

Value set type	enumerated, closed
XML name	vs_tdInteractPassive.xml

Legal values Values are taken from the TEI suggestions.[40]

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
self	Text is addressed to the originator e.g. a diary
single	Text is addressed to one other person e.g. a personal letter
many	Text is addressed to a countable number of others e.g. a conversation in which all participants are identified
group	Text is addressed to an undefined but fixed number of participants e.g. a lecture
world	Text is addressed to an undefined and indeterminately large number e.g. a published book**. Default**

▷ ***tdInteractRole***

Describes the roles of addressor and addressee in terms of technical expertise concerning the topic of the text. This information is usually only interesting if *tdDomain* has a value other than its default. Otherwise *tdInteractRole* will default to “basic-basic”.

Properties

Value set type	enumerated, closed
XML name	vs_tdInteractRole.xml

[40] http://www.tei-c.org/release/doc/tei-p5-doc/html/ref-interaction.html

Legal values

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
basic-basic	A person with basic knowledge of the topic, i.e. a layperson, addresses another person with basic knowledge**. Default**
basic-advanced	Somebody with basic knowledge addressing somebody with advanced knowledge
basic-expert	Somebody with basic knowledge addressing somebody with expert knowledge
advanced-basic	Advanced addressing basic
advanced-advanced	Advanced addressing advanced
advanced-expert	Advanced addressing expert
expert-basic	Expert addressing basic
expert-advanced	Expert addressing advanced
expert-expert	Expert addressing expert

▷ ***tdPrepType***

Describes the extent to which a text may be regarded as prepared or spontaneous.

Properties

Value set type	enumerated, closed
XML name	vs_tdPrepType.xml

Legal values A subset from the TEI suggestion:[41]

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
none	The text is spontaneous or unprepared
revised	Polished or revised before presentation**. Default**

[41] http://www.tei-c.org/release/doc/tei-p5-doc/html/ref-preparedness.html

▷ ***tdPurposeType***

Characterizes a single purpose or communicative function of the text, e.g. whether it is informative, expressive, etc.

Properties

Value set type	enumerated, closed
XML name	vs_tdPurposeType.xml

Legal values Following the TEI suggestions:[42]

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
persuade	Didactic, advertising, propaganda, etc.
express	Self expression, confessional, etc.
inform	Convey information, educate, etc.. **Default**
entertain	Amuse, entertain, etc.

▷ ***textCreationYear***

The year in which the text was authored.

Properties

Value set type	descriptive
XML name	n/a

Legal values Four-digit date. If the year of text creation is not known, *textCreationYear* is set to the same value as *publDate*.

▷ ***textFileName***

Name of the source file from which this text is drawn, that is usually the name of the file the text was delivered in. The organization having collected the text is responsible for keeping a copy of its source file in an archive if it wants to enable future corrections or modifications of the CTB version of the text with regard to certain information only contained in the source file.

[42] http://www.tei-c.org/release/doc/tei-p5-doc/html/ref-purpose.html

Properties

Value set type	descriptive
XML name	n/a

Legal values Any legal (path and) filename pointing to the source file in the archive.

▷ ***textId***

Unique text identifier.

Properties

Value set type	system: descriptive
	prefixes listed below: enumerated, open
XML name	system: n/a
	prefixes: vs_textId.xml

Legal values Values for *textId* of *textIdType* "ctb" (cf. below): Specified 10-digit integer. Identifiers of this type are composed as follows: The first two digits (from the left) indicate the project framework within which the texts were collected (which can be some other than DK-CLARIN). Thus, the first two digits can be viewed as a kind of prefix. The following set of prefixes of *textIdType* "ctb" is used:

Value	Description
99999999	Info has not been determined yet
0	Info is irrelevant, non-existent, or undeterminable
10	Korpus 2000 material from 'Politiken', 'Jyllands-Posten' and 'fyldepennen.dk'
11	Other Korpus 2000 material
120	PAROLE (OBS! PAROLE comprises some material from DDOC)
121	Material from the Corpus of The Danish Dictionary (DDOC)
13	Material collected by DSL's ordnet.dk project
14	Infomedia material collected by DSL's ordnet.dk project
20	Infomedia material collected by DK-CLARIN WP2.1, LGP Corpus
2009	Infomedia magazines 2010-11 collected by DK-CLARIN WP2.1, LGP Corpus
21	Material collected by DK-CLARIN WP2.1, LGP Corpus
22	Material collected by DK-CLARIN WP2.2, LSP Corpus
23	Material collected by DK-CLARIN WP2.3, Renaissance Corpus
24	Material collected by DK-CLARIN WP2.4, ADL/JVJ
25	Material collected by DK-CLARIN WP2.5, Nationalmuseet
26	Material collected by DK-CLARIN WP2.6, Parallel Corpus
90000	DiaKo - optegnelser af dialekter, NFI/ØMO

However, depending on the actual id system (see *textIdType* below), strings are acceptable as well.

▷ ***textIdType***

Identifies the type of *textId* given.

Properties

Value set type	enumerated, open
XML name	vs_textIdType.xml

Legal values Default type is "ctb", but other project- or institution-internal types can be added.

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
ctb	Text id according to the id system specified for the Clarin Text Bank. **Default**
ddo	Text id according to the id system specified for the Corpus of The Danish Dictionary
dsst	Text id according to the id system of Dansk Sprog- og Stilhistorisk Tekstbase (WP2.3)
im	Text id according to the id system used by Infomedia (WP2.1)
wiki	Wikipedia ID found in Wikipedia export documents at /mediawiki/page/id/text()
extUri	External URI/URL of the text resource

▷ ***textTitle***

Title of the text from which the sample is taken.

Properties

Value set type	descriptive
XML name	n/a

Legal values Any string denoting a text title.

▷ ***textUri***

Resource identifier locating the text source.

Properties

Value set type	descriptive
XML name	n/a

Legal values Any valid URI pointing at a source instance of the text.

▷ ***theirClassification***

URL of an official text classification scheme.

Properties

Value set type	enumerated, open
XML name	vs_theirClassification.xml

Legal values Any valid URL pointing to a classification scheme. Currently, the following official classification scheme URLs are defined:

Value	Description
nil	Info has not been determined yet
empty	Info is irrelevant, non-existent, or undeterminable
http://ctb.dsl.dk/class/classCode/CLARIN/demo.xml	Classification containing some demo values

▷ ***theirValue***

Value given in an official text classification system.

Properties

Value set type	n/a
XML name	n/a

Legal values Legal values according to official classification.

▷ ***titleLevel***

Indicates the level of the title within a publication, whether the title is on analytic level, i.e. the text is part of a collection (e.g. a newspaper), or whether it is on the monographic level, i.e. a stand-alone publication (e.g. a novel).

Properties

Value set type	enumerated, closed
XML name	vs_titleLevel.xml

Legal values

Value	Description
m	Monographic title**. Default**
a	Analytic title

3.3.3 Additional value sets for text classification

Text classification outside the scope of standard TEI header semantics is achieved by using a number of <catRef> schemes inside the <textClass> element. This special information is needed to enable older corpus material like the DDOC and KORPUS 2000 to be easily integrated in the new structure. The following types of information are inherited from these two corpora, the general structure for the <catRef> element being

```
<catRef
  scheme="http://ctb.dsl.dk/class/catRef/textGroup/scheme"
  target="#target"/>
```

where the *schemes* are in use can be seen under *myClassification*, see 3.3.2.1 on page 64.

In CTB, there is no <catRef> scheme for genre information. Instead, the <factuality> element under <textDesc> is used. DDOC and KORPUS 2000 genre values (as well as other obsolete values in an CTB context) should be mapped to the CTB header, see Chapter **??**.

Chapter 4

Text formatting

What an annotated text should look like

Deliverables concerned

D2 Tokenizer A consistent and easy-to-use token concept needs to be defined. The token concept has important implications on the design of the tokenizer tool and the POS-tagger applied in WP 2.1. **Outcome:** Tool and report.

D13 TEI transducer The original plan for WP 2.1 was based on the assumption that the repository of potential corpus texts – the corpus text bank – most likely would have a non-XML structure (relational db). In order to make interchange of texts easy and in order to make them fit into the intended resource repository of DK-CLARIN, the development of a transducer that could reshape the texts and metadata stored in the corpus text bank to valid TEI XML seemed necessary. However, during the course of the project, it became clear that the text bank itself should be implemented as an XML database so that the texts could be stored in their final TEI XML format. Therefore, the task of developing a transducer became a task of defining an appropriate subset of TEI in order to suit the metadata and text format needs of DK-CLARIN. **Outcome:** Report.

Outline of this chapter

This chapter gives an overview of the *general* principles of text formatting and word level markup for corpus texts. For more *specific* information, consult the documentation of applications and procedures for segmenting and annotating text, see Chapter 5 and Chapter 6.

4.1 Basic considerations

4.1.1 Motivation

The main motivation of defining a *general text format* is to establish a joint basis for all tools that operate on CTB texts. Thus, tools do not need to be configured for a multitude of formats which means that they will be easier and less error-prone to develop and maintain.

4.1.2 Format requirements

1. The format has to be expressed by means of TEI P5.
2. Annotations should not interfere with the basic format of the text proper.
3. The basic format of the text proper should not be biased by interpretations.
4. It must be possible to annotate one single text with various (possibly mutually exclusive) types of annotations, each type appearing as a group of annotations that conceptually belong together.
5. Each annotation in an annotation group must be able to refer to either the text proper or to another annotation group which means that layers of annotations, i.e. annotations on annotations, become feasible.
6. It should be possible to store annotations separate from the text proper.
7. Several versions of the text proper should be avoided.

4.1.3 Consequences

Pre-tokens: The text has to be mechanically segmented into rather primitive tokens which we call *pre-tokens* in the following as they do not reflect any linguistic word conceptualizations.

Reference: It must be possible to unequivocally refer to these tokens.

Transformation: A generalized, multi-purpose format that needs to be transformed in order to be legible for humans which means that specific viewers and editors must be developed in order to interact with the text.

4.2 Formatting text

4.2.1 A source sample to be formatted

The following snippet shows a paragraph taken from the DDOC[1] source files:

```
<p><s>To kendte russiske historikere Andronik Mirganjan og
Igor Klamkin tror ikke, at Rusland kan udvikles uden en
"jernnæve".</s></p>[2]
```

A text version like this one is called the *source version* of a text. The source version of a text must comply with the TEI P5 specifications in order to be formatted. If it does not, it must be converted into TEI P5 prior to further formatting. The excerpt above conforms to TEI.

4.2.2 Bad: Formatting against the requirements

In the DK-PAROLE Corpus, cf. Keson (1998a), this same paragraph/sentence is formatted like this:

```
<p>
  <s>
    <W lemma="to" msd="AC---U=--">To</W>
    <W lemma="kendt" msd="ANP[CN]PU=[DI]U">kendte</W>
    <W lemma="russisk" msd="ANP[CN]PU=[DI]U">russiske</W>
    <W lemma="historiker" msd="NCCPU==I">historikere</W>
    <W lemma="Andronik" msd="NP--U==-">Andronik</W>
    <W lemma="Mirganjan" msd="NP--U==-">Mirganjan</W>
    <W lemma="og" msd="CC">og</W>
    <W lemma="Igor" msd="NP--U==-">Igor</W>
    <W lemma="Klamkin" msd="NP--U==-">Klamkin</W>
    <W lemma="tro" msd="VADR=----A-">tror</W>
    <W lemma="ikke" msd="RGU">ikke</W>
```

[1] DDOC = Corpus of The Danish Dictionary, cf. Norling-Christensen and Asmussen (1998).

[2] Original source: Leon Nikulin: Jeltsins skæbnetime, Det Fri Aktuelt, 1.12.1992, p. 14. Actually, the original paragraph is longer than the single sentence reproduced here.

```
      <W lemma="," msd="XP">,</W>
      <W lemma="at" msd="CS">at</W>
      <W lemma="Rusland" msd="NP--U==-">Rusland</W>
      <W lemma="kunne" msd="VADR=----A-">kan</W>
      <W lemma="udvikle" msd="VAF-=----P-">udvikles</W>
      <W lemma="uden" msd="SP">uden</W>
      <W lemma="en" msd="PI-CSU--U">en</W>
      <W lemma="&quot;" msd="XP">"</W>
      <W lemma="jernnæve" msd="NCCSU==I">jernnæve</W>
      <W lemma="&quot;" msd="XP">"</W>
      <W lemma="." msd="XP">.</W>
    </s>
  </p>
```

Even if the format is easy to decode, at least for humans, it has certain shortcomings running counter to the requirements defined in Section 4.1.2 above:

1. The format is not expressed by means of TEI P5 as <W> is not a legal element in TEI P5 (whereas <w> would be, but *msd* is not a legal attribute of <w>).
2. Annotations interfere with the format of the text proper (as attributes of the <W>-element).
3. The text format is affected by interpretation: Punctuation characters are considered as words that again carry a lemma tag and a morphosyntactic tag.
4. New annotation layers can hardly be added without further interfering with the already existing format (e.g. by adding further attributes to the <W> element).
5. It is not possible to refer to the basic tokens of the text.
6. Annotations cannot easily be separated from the text proper.
7. Other interpretations of the text expressed by alternative annotations may require new versions of the text.

4.2.3 Good: Formatting according to the requirements

4.2.3.1 From source version to base format

Two of the consequences emerging from the requirements are that the text has to be mechanically segmented into *basic tokens*[3] and that it must be possible to unequivocally refer to these tokens, cf. Section 4.1.3. Mechanical text segmentation, or pre-tokenization, is carried out by certain textual surface items, i.e. characters, only. For segmentation purposes characters fall into three categories:

[3] The segmentation process is called pre-tokenization, cf. Chapter 5.

- ▷ Letters and numbers, i.e. alpha-numeric characters
- ▷ Whitespace characters
- ▷ Punctuation characters

Continuous sequences of alpha-numeric characters are considered 'words' even if these segments are not necessarily in accordance with a linguistic definition of a word. Linguistic interpretations are deliberately avoided at this point. 'Words' are put into <w> elements.

Whitespace and punctuation is put into <c> elements – character by character – that can be of *type* "s" (space) or "p" (punctuation). The non-obligatory *subtype* attribute may specify some other characteristics of the character in question, e.g. the length of a whitespace. Specifications of the possible inventory of the subtype attribute are not given before it turns out that this attribute is really needed. Standard space characters (ASCII 32) are not explicitly denoted in the <c> elements (i.e. they remain empty) whereas other whitespace characters such as tabs (coded as 	) can be given in the element.

<w> and <c> elements are the smallest segments (i.e. *basic tokens*) of a text. Each of them carries a unique *xml:id* that allows referencing to it from elsewhere.[4] The source example given in Section 4.2.1 would look like this after segmentation:

```
<p>
  <s>
    <w xml:id="x002">To</w>
    <c xml:id="x003" type="s"/>
    <w xml:id="x004">kendte</w>
    <c xml:id="x005" type="s"/>
    <w xml:id="x006">russiske</w>
    <c xml:id="x007" type="s"/>
    <w xml:id="x008">historikere</w>
    <c xml:id="x009" type="s"/>
    <w xml:id="x010">Andronik</w>
    <c xml:id="x011" type="s"/>
    <w xml:id="x012">Mirganjan</w>
    <c xml:id="x013" type="s"/>
    <w xml:id="x014">og</w>
    <c xml:id="x015" type="s"/>
    <w xml:id="x016">Igor</w>
    <c xml:id="x017" type="s"/>
    <w xml:id="x018">Klamkin</w>
    <c xml:id="x019" type="s"/>
    <w xml:id="x020">tror</w>
    <c xml:id="x021" type="s"/>
    <w xml:id="x022">ikke</w>
    <c xml:id="x023" type="p">,</c>
    <c xml:id="x024" type="s"/>
```

[4] Assigning IDs requires some sort of control that every ID is unique.

```
      <w xml:id="x025">at</w>
      <c xml:id="x026" type="s"/>
      <w xml:id="x027">Rusland</w>
      <c xml:id="x028" type="s"/>
      <w xml:id="x029">kan</w>
      <c xml:id="x030" type="s"/>
      <w xml:id="x031">udvikles</w>
      <c xml:id="x032" type="s"/>
      <w xml:id="x033">uden</w>
      <c xml:id="x034" type="s"/>
      <w xml:id="x035">en</w>
      <c xml:id="x036" type="s"/>
      <c xml:id="x037" type="p">"</c>
      <w xml:id="x038">jernnæve</w>
      <c xml:id="x039" type="p">"</c>
      <c xml:id="x040" type="p">.</c>
    </s>
  </p>
```

This formatted version of the source text is called the text's *base format*. The base format is the standard input format for all tools like tokenizers, sentence splitters, lemmatizers, and taggers of all kinds, see the motivation for a fixed text format in Section 4.1.1.

As can be seen, markup above <w> and <c> level that is already present in the source version text, may be kept as long as the source version complies with the TEI specifications. In this case, the <p> and <s> tags were kept; <p> tags may carry an *xml:lang* attribute that indicates the language of the paragraph by using a value from the languageId value set described in 3.3.2. Even though tags other than <c>, <w>, <s>, and <p> may be used as long as they are TEI-compliant,[5] this type of markup should be avoided and added as span groups instead, see the following section.

4.2.3.2 Annotations

Annotations are given separately from the base format version of the text by a number of elements enclosed in <spanGrp> elements. The elements contain the annotations themselves that are either attached to one single basic token or a number of continuous basic tokens. Attachment is achieved by referencing the *xml:id* units from the obligatory *from* attribute of the element and – in case continuous basic tokens are referenced and not only a single one – the facultative *to* attribute. Every <spanGrp> contains one type of annotations only. The *ana* attribute of the <spanGrp> element refers to the application or method that has produced the annotations, listed in the <appInfo> element of the header. Some annotation examples follow.

[5] Among such elements is <lb> (line break) whereas the <hl> element, which was introduced by some other WP 2 projects, is not allowed in the body of a text. See also Section 4.2.3.4 on further examples.

Sentences In the base format version given in Section 4.2.3.1 <p> and <s> tags from the source version were kept as independent tags as they occurred above the level of the basic tokens and met the TEI specifications. The <p> tags are an obligatory part of the structure: Raw text as well as <w> and <c> elements must be encapsulated by <p> elements or equivalent elements, e.g. <ab>. However, the <s> tags could alternatively be expressed as <spanGrp> annotations. The following example shows how sentences can be tagged in this alternative way making <s> tags in the the base format version unnecessary.

```
<spanGrp ana="#sentences">
  <span from="#x002" to="#x040">s</span>
</spanGrp>
```

Lemmas The following example shows what the PAROLE lemma annotation expressed by the *lemma* attributes as shown in Section 4.2.2 looks like when expressed by the <spanGrp> annotation.

```
<spanGrp ana="#paroleLemma">
  <span from="#x002">to</span>
  <span from="#x004">kendt</span>
  <span from="#x006">russisk</span>
  <span from="#x008">historiker</span>
  <span from="#x010">Andronik</span>
  <span from="#x012">Mirganjan</span>
  <span from="#x014">og</span>
  <span from="#x016">Igor</span>
  <span from="#x018">Klamkin</span>
  <span from="#x020">tro</span>
  <span from="#x022">ikke</span>
  <span from="#x023">,</span>
  <span from="#x025">at</span>
  <span from="#x027">Rusland</span>
  <span from="#x029">kunne</span>
  <span from="#x031">udvikle</span>
  <span from="#x033">uden</span>
  <span from="#x035">en</span>
  <span from="#x037">"</span>
  <span from="#x038">jernnæve</span>
  <span from="#x039">"</span>
  <span from="#x040">.</span>
</spanGrp>
```

The linguistic interpretation expressed by the PAROLE lemma annotation is exactly the same as in the example shown in Section 4.2.2, including that punctuation characters are treated as lemmas, but this interpretation no longer imposes a certain formatting on the base format of the text. Base format and interpretation are kept apart.

POS and inflection In the same manner, the morphosyntactic annotation of the PAROLE corpus can be expressed by a <spanGrp>:

```
<spanGrp ana="#paroleMsd">
  <span from="#x002">AC---U=--</span>
  <span from="#x004">ANP[CN]PU=[DI]U</span>
  <span from="#x006">ANP[CN]PU=[DI]U</span>
  <span from="#x008">NCCPU==I</span>
  <span from="#x010">NP--U==-</span>
  <span from="#x012">NP--U==-</span>
  <span from="#x014">CC</span>
  <span from="#x016">NP--U==-</span>
  <span from="#x018">NP--U==-</span>
  <span from="#x020">VADR=----A-</span>
  <span from="#x022">RGU</span>
  <span from="#x023">XP</span>
  <span from="#x025">CS</span>
  <span from="#x027">NP--U==-</span>
  <span from="#x029">VADR=----A-</span>
  <span from="#x031">VAF-=----P-</span>
  <span from="#x033">SP</span>
  <span from="#x035">PI-CSU--U</span>
  <span from="#x037">XP</span>
  <span from="#x038">NCCSU==I</span>
  <span from="#x039">XP</span>
  <span from="#x040">XP</span>
</spanGrp>
```

Again, punctuation characters are treated as independent units carrying their own morphosyntactic annotation (“XP”).

Alternative POS markup If the morphosyntactic PAROLE annotation is considered inadequate for certain purposes, a new annotation group with another tag set and another treatment of punctuation easily can be added, for example:

```
<spanGrp ana="#parolePOS">
  <span from="#x002">NUM</span>
  <span from="#x004">ADJ</span>
  <span from="#x006">ADJ</span>
  <span from="#x008">S</span>
  <span from="#x010">S/span>
  <span from="#x012">S</span>
  <span from="#x014">KON</span>
  <span from="#x016">S</span>
  <span from="#x018">S</span>
  <span from="#x020">V</span>
  <span from="#x022">ADV</span>
  <span from="#x025">SUB</span>
  <span from="#x027">S</span>
  <span from="#x029">V</span>
  <span from="#x031">V</span>
  <span from="#x033">PRP</span>
```

```
    <span from="#x035">ART</span>
    <span from="#x038">S</span>
  </spanGrp>
```

Names, manually annotated In the same manner e.g. names could be marked up, for example as result of a manual procedure:

```
<spanGrp ana="#paroleNames">
  <span from="#x010" to="#x012">person</span>
  <span from="#x016" to="#x018">person</span>
  <span from="#x027">place</span>
</spanGrp>
```

4.2.3.3 Putting base format and annotation layers together

The base format version from Section 4.2.3.1 and all annotation groups are structurally combined as shown in the following sketch. The text in base format is enclosed by <body> tags whereas the <spanGrp> elements are siblings of the <body> element, following it in arbitrary order:

```
<text>
  <body>[6]
    - Text in base format (with obligatory paragraph markup)
  </body>
    - <spanGrp> with sentence markup[7]
    - <spanGrp> with lemma annotations
    - <spanGrp> with POS and inflectional annotations
    - <spanGrp> with alternative POS markup
    - <spanGrp> with name annotations
</text>
```

4.2.3.4 Additional information in the base version

According to TEI 5, only a few elements may occur as siblings to the <w> and <c> elements. The use of such elements to give additional textual or graphical formatting information should be generally avoided. This type of information should be placed in <spanGrp> elements if it cannot be entirely eliminated.[8]

However, as putting additional information into <spanGrp> elements may complicate the process of converting text from original versions to the base version, some exceptions are the following tags which occur in some forum texts gathered in WP 2.1:

[6] TEI expects the text to be subdivided into front matter, text body, and back matter. For corpus texts, a subdivision of this kind is unnecessary. However, TEI demands at least the <body> subdivision. Therefore, all CTB <text> elements contain one single <body> element encapsulating the text body.

[7] If the <s> markup is not already contained in the base format version of the text.

[8] As concerns linguistic corpus texts, layout information is dispensable in most cases and therefore can be removed.

<quote> may occur as sibling of <w> and <c> tags and may embed them as well, i.e. have them as children. This tag is used for surrounding text material that is quoted in forum posts; in these cases it always carries the *type* attribute 'forum'.

<add> may occur as sibling of <w> and <c> tags but cannot contain them. This element gives additional information on extra-textual resources like images (mandatory *type* attribute is 'img', mandatory *source* attribute is the URI of the resource) or pointers (*type* is 'url', *source* as before), or video (*type* 'video', *source* as before).

4.2.3.5 What happens to the source version of a text?

When converting a TEI P5 source version of a text into base format, all information is kept either as additional markup in the base format version like the <p> and <s> markup in the example shown in Section 4.2.3.1 or as independent span groups as shown in the sentences example in Section 4.2.3.2. As all necessary textual and extra-textual information contained in the source version can be expressed in base format in conjunction with a number of span groups, the source version proper becomes obsolete, and thus is not kept as a member of the CTB files. However, it should be stored independently in some location that can be referenced from the CTB header through one of the <idno> elements within <biblStruct>, see Section 3.2.1.5. The same applies to other source versions like URLs from which the TEI P5 base versions may have been derived. In the case of the WP 2.1 corpus project, all original texts are stored in the text file repository in the /Data volume on the server ja-korpus.dsl.lan – the same server where the eXist-db is installed, see Section 2.2.2.3.

4.2.3.6 Format requirements revisited

So far, the format requirements 1 – 4 and 7 have been highlighted by the example given above in Section 4.1.2. Regarding requirement 5, it has been shown how an annotation group refers to the text proper, namely through *from* and optional *to* attributes referencing the *xml:id* attributes of the basic textual units. What has not been shown yet is how to layer annotation groups by letting them reference other annotation groups. This will be part of the following examples section. Requirement 6, the possibility of storing annotations separate from the text proper, may be illustrated in a separate document.

4.2.4 Example

4.2.4.1 Tokenization and layers of annotations

The base format of a text may to some extent resemble a tokenized version of it. However, 'real' tokenization normally requires a certain amount of language-

specific linguistic knowledge on how to identify words, but the segmentation procedure applied to the source version of a text in order to transform it into base format does not possess this kind of knowledge; in fact, the segmentation procedure is entirely ignorant on delicate linguistic considerations. This means, that in some cases it may be desirable to apply a more intelligent tokenization procedure in addition to the mere segmentation of the source text as the following example shows.

Source version To keep this example as simple as possible, markup above the level of the basic textual units is kept to a minimum, i.e. the obligatory `<p>` tags:

```
<p>De staar over for et PROBLEM i dag.</p>
```

Base format The source version is converted into base format:

```
<p>
  <w xml:id="y01">De</w>
  <c xml:id="y02" type="s"/>
  <w xml:id="y03">staar</w>
  <c xml:id="y04" type="s"/>
  <w xml:id="y05">over</w>
  <c xml:id="y06" type="s"/>
  <w xml:id="y07">for</w>
  <c xml:id="y08" type="s"/>
  <w xml:id="y09">et</w>
  <c xml:id="y10" type="s"/>
  <w xml:id="y11">PROBLEM</w>
  <c xml:id="y12" type="s"/>
  <w xml:id="y13">i</w>
  <c xml:id="y14" type="s"/>
  <w xml:id="y15">dag</w>
  <c xml:id="y16" type="p">.</c>
</p>
```

Tokenization and regularization The linguistically ignorant segmentation mechanism that converts the source version into base format, treats the two word pairs *over for* and *i dag* as four separate words even if each pair reasonably may be considered as one single word, once in a while also with substandard spelling *overfor* and *idag*. In order to express this linguistically enlightened view on which textual units are to be treated as tokens, a token annotation layer is introduced as a span group. The tokenization algorithm applied furthermore regularizes the spelling of words according to some predefined norm of some kind, in the present case *De* is regularized as *de*, *staar* as *står*, and *PROBLEM* as *problem*. The `<span>` elements of this span group all carry an additional *xml:id* attribute giving each `<span>` a unique ID which can be referenced from elsewhere, i.e. from other annotation groups.

```
<spanGrp ana="#tokenRegular">
  <span xml:id="t1" from="#y01">de</span>
  <span xml:id="t2" from="#y03">står</span>
  <span xml:id="t3" from="#y05" to="#y07">over for</span>
  <span xml:id="t4" from="#y09">et</span>
  <span xml:id="t5" from="#y11">problem</span>
  <span xml:id="t6" from="#y13" to="#y15">i dag</span>
</spanGrp>
```

Lemmatization The following lemma annotations no longer address the basic textual units but instead the elements of the annotation group above:

```
<spanGrp ana="#lemma">
  <span from="#t1">de</span>
  <span from="#t2">stå</span>
  <span from="#t3">over for</span>
  <span from="#t4">en</span>
  <span from="#t5">problem</span>
  <span from="#t6">i dag</span>
</spanGrp>
```

POS annotation Finally, the following POS annotations address the same token annotation layer as does the lemma annotation group:

```
<spanGrp ana="#lemma">
  <span from="#t1">PRON</span>
  <span from="#t2">V</span>
  <span from="#t3">PRP</span>
  <span from="#t4">ART</span>
  <span from="#t5">S</span>
  <span from="#t6">ADV</span>
</spanGrp>
```

Part III

Collecting

Chapter 5

Processing text

Bringing texts into good shape

Deliverables concerned

[...]

Outline of this chapter

[...]

5.1 Implementation

The text-conversion services are implemented in part as standalone Java programs, web-services based on eXist-db/XQuery,[1] 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.

5.1.1 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.

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

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

stored in an eXist database collection can, in fact, be executed by simply pointing your web browser to the REST URL.[2]

As described in the online developer's guide for eXist,[3] 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:

```
<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.

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

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

5.1.1.1 Demo Application

An application containing a collection of interactive demos[4] 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!

5.1.2 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();
      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);
        }
```

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

```
            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());
          }
        }
      }
    }
```

5.2 Header constructor: make-header

5.2.1 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 Section 5.2.3 and the demo application described in Section 5.1.1.1.[5] 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.

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

5.2.2 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!

5.2.3 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[6]

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, 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.

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

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>`** element 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 5.5 on page 121.

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 5.6 on page 123. 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 5.6 on page 123. 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 (5.2) 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. 5.5 on page 121. 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 5.1.2 on page 101 can easily be adopted to be used with this web-service too.

5.3 Pre-tokenizer: pretokenize

5.3.1 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 5.1.1.1). 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.

5.3.1.1 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 5.1: 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: —

Table continues on next page...

Character *(continued)*	Description *(continued)*	Code *(continued)*	Variants *(continued)*
/	slash	002F	005C backslash: \
_	underscore	006F	2026 horizontal ellipsis: …
!	exclamation mark	0021	
.	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.

5.3.2 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 5.3.1.1) 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!

5.3.3 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[7]

```
<request xmlns="http://ctbws.dsl.dk/ns/request">
  <idPrefix>A</idPrefix>
  <textId>2100000000</textId>
  <text>
    <p>A paragraph.</p>
    <p>Another paragraph.</p>
  </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 5.1.2 on page 101 can easily be adopted to be used with this web-service too.

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

5.4 Text id registry: register-text

5.4.1 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.

5.4.2 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!

5.4.3 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[8]

```
<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 corrupted by improper use! Use the 'demo' text group instead as it has been created for testing purposes! **OBS!**

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.

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

Client development

The Java client example shown in Section 5.1.2 on page 101 can easily be adopted to be used with this web-service too.

5.5 id dispatcher: make-id

5.5.1 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.

5.5.2 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.[9]

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!

5.5.3 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!

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

Request body[10]

```
<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.

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 5.1.2 on page 101 can easily be adopted to be used with this web-service too.

5.6 Word and paragraph counter: count-units

Work in progress.

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

Part IV

Markup

Chapter 6

Survey of POS taggers

Approaches to making words tell who they are

Deliverables concerned

D10 Lemmatizer It is considered indispensable that corpus texts need to indicate the lemma form of each inflected word form in the corpus to let the user of the corpus perform more flexible queries. Therefore, it is necessary to either develop or configure a lemmatizer (that may be based on a full-form lexicon or a morphological analyzer). In the context of WP 2.1, a lemmatizer designed as an integral part of a POS tagger is the preferable solution. **Outcome:** Tool with documentation.

D11 POS tagger In order to tag tokens in corpus texts with part-of-speech information, it is necessary to either develop or configure a POS tagger (either based on a full-form lexicon or a morphological analyzer) and a suitable tag set. **Outcome:** Tool with documentation.

Outline of this chapter

This chapter describes WP 2.1's requirements to part-of-speech (= POS) tagging and provides a survey of existing POS approaches and their suitability. The survey is based on the requirements defined in Section 6.1. The chapter finishes with some conclusions on which approach to choose for tagging WP 2.1 corpus texts.

6.1 Requirements

Within a narrow DK-CLARIN context, the process of POS tagging could be reduced to just letting some kind of black box perform what is needed in order to get the text material of this corpus work-package marked-up with appropriate morphosyntactic info. Seen in this is light, the major requirement would be precision, therefore the only relevant answer allowed to ask on this topic may seem: How close to 100 % do we get? However, precision is not a quality in itself but the result of other properties.

As a black box is not configurable, and as requirements definitely will change over time and beyond the narrow scope of the DK-CLARIN project, "close to 100 %" may not be that close once some of the prerequisites of the tagging scenario have been modified. Therefore, an open configurable solution seems to be a more forward-looking approach. *Open* means that both the tagger software itself and its linguistic 'knowledge' must be open source and thus configurable, and available for free for everybody. Open source and free availability is considered a major requirement and crucial in order to achieve a permanent level of high precision.

A secondary requirement is that the tagger should not just apply a model of the language in question, i.e. Danish, and assign appropriate POS tags to words, but also perform lemmatization, i.e. assign the base forms of all (inflected) words. As a consequence of that, the tagger should apply an open-source full form lexicon (that is available already, but definitely needs to be enhanced).[1]

[1] See http://korpus2000.dk/e-resurser/boejningsformer_download.php?lang=uk.

Another requirement is that the tagger should be written in a widely-used, platform-independent programming language that also is used as major coding language in at least one of the DK-CLARIN corpus work-packages and thus provides a comprehensive API. As WP 2.1 uses Java and probably has the largest bulk of text to tag, Java is considered the programming language of choice for the tagger.

The tagger must be adaptive to various needs. For small occasional tagging purposes, a web-based solution seems optimal. In order to make a web-based tagger fit into specific text processing lines, it should come as a web service as well. Finally, the software should be executable on a stand-alone PC or workstation in order to process vast amounts of text quickly and without the need to access remote services.

Moreover, the tagger should be well-documented and continuously maintained (and enhanced) by a community rather than one single developer. It should be user-friendly to set up and get running.

To sum up, the following requirements given in prioritized order are considered crucial:

Availability: Free open-source tagger code and linguistic resources

Features: Capable of performing both POS tagging and lemmatization

Code: Tagger coded in Java or at least providing a Java API

Architecture: Flexible architecture adaptive to various usage scenarios

Usability: Well-documented and continuously maintained, user-friendly

6.2 Survey

In this section, various taggers are presented in arbitrary order and described according to the requirements given in the section above. The section is subdivided in a listing of universal taggers, i.e. taggers that in principle are language-independent, and taggers, specifically designed for or adopted to Danish.

6.2.1 Universal taggers

A comprehensive list of taggers can be found on Stanford University's NLP site[2]. Evert and Giesbrecht (2009) give an evaluation of the performance on German of some of these taggers, i.e. TreeTagger, TnT, SVMTagger, Stanford tagger, and the Apache UIMA Tagger.

1. IMS's TreeTagger – a language-independent POS tagger[3]

[2] http://nlp.stanford.edu/links/statnlp.html
[3] http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/

Availability: Free for academic use, easy download, comes with free language models for approximately 10 languages, but not Danish. Not open source. Assessment: **poor**.

Features: HMM[4] tagger using decision trees for smoothing. Performs lemmatization if a full form lexicon is supplied. Assessment: **good**.

Code: Comes as precompiled binaries, a Java API is available, but no java source files. Assessment: **fair**.

Architecture: The Java wrapper makes it possible to adopt the tagger to various usage-scenarios. Assessment: **good**.

Usability: The tagger was developed as part of the project *Textcorpora und Erschließungswerkzeuge* (1993-1996) at the IMS (same project as Corpus Workbench, CWB/CQP) and has not changed substantially since then. Documentation is sparse (included as a *read me* in the download package) but probably enough to both use the software for training and tagging purposes. The tagger is maintained by one single person (Helmut Schmid). However, it seems to have a fairly large user community that partly overlaps with those who use CQP as well. Assessment: **fair**.

2. TnT[5]

Availability: Free of charge for non-commercial research purposes. Download requires signed license agreement. Comes with language models for German and English. Not open source. Assessment: **poor**.

Features: HMM tagger with standard smoothing. Does not perform lemmatization. Assessment: **fair**.

Code: Comes as precompiled binaries, no wrappers available. Assessment: **poor**.

Architecture: May work in an I/O pipeline setting. Assessment: **poor**.

Usability: The tagger was developed by Thorsten Brants at Saarland University 1993-1999 and does not seem to have changed substantially since then. Documentation[6] is OK. The tagger does not seem to be actively maintained any longer. Assessment: **poor**.

3. SVMTool[7]

[4] Hidden Markov Models.

[5] http://www.coli.uni-saarland.de/~thorsten/tnt/

[6] http://www.coli.uni-saarland.de/~thorsten/publications/Brants-TR-TnT.pdf

[7] http://www.lsi.upc.es/~nlp/SVMTool/

Availability: Open source with models for Catalan, English, and Spanish. However, it must be trained by using the non open-source SVMlight software which can be used for free for academic purposes. Assessment: **fair**.

Features: Based on Support Vector Machines. Configurable in many ways. No lemmatization. Assessment: **fair**.

Code: C++ and Perl versions, Perl API. Assessment: **fair**.

Architecture: Can work in an I/O pipeline setting. Assessment: **poor**.

Usability: The SVMTool has been developed at the TALP Research Center NLP group at Universitat Politècnica de Catalunya. Latest version seems to be from 2006. Documentation[8] is OK. Assessment: **fair**.

4. Stanford Log-linear Part-Of-Speech Tagger[9]

Availability: Open source. Models for English, Arabic, Chinese, and German. Assessment: **good**.

Features: Based on the Maximum Entropy framework. It can be trained on any language on a POS-annotated training text for the language. No lemmatization. Assessment: **fair**.

Code: Java implementation. Assessment: **good**.

Architecture: Open source, thus easy integration in other environments. Assessment: **good**.

Usability: Comes with good documentation and seems well-maintained and up-to-date. Literature pointers on up-to-date website. Java NLP user lists are available for further information. However, applying the tagger to other languages than those with pre-compiled models, seems rather challenging. Assessment: **fair**.

5. Apache UIMA Tagger[10]

Availability: Open source. Comes with models for English and German. Assessment: **good**.

Features: HMM tagger as part of the *Apache Unstructured Information Management Architecture* (UIMA) framework. No lemmatization. Assessment: **fair**.

Code: Java. Assessment: **good**.

Architecture: Flexible. Web service integration as component of the framework. Assessment: **good**.

[8] http://www.lsi.upc.edu/~nlp/SVMTool/SVMTool.v1.3.pdf
[9] http://nlp.stanford.edu/software/tagger.shtml
[10] http://uima.apache.org/sandbox.html

Usability: Website and documentation seems OK even if latest website updates are from 2009. However, the UIMA framework has reached a degree of complexity (obscure code interdependencies) that makes the use of the tagger component rather cumbersome. Assessment: **poor**.

6. Chris Biemann's unsupos – unsupervised POS tagging[11]

Availability: Open source. Models for a number of languages available including Danish. It is not clear what type of material the Danish model is based on. Assessment: **good**.

Features: Unsupervised POS tagging. Does not require an annotated training corpus. Instead, word categories are determined by analyzing a large sample of monolingual, sentence-separated plain text. The tag set can probably not be determined by the user/linguist. No lemmatization. Assessment: **poor**.

Code: Java implementation. Assessment: **good**.

Architecture: Probably easy to integrate in various environments. Assessment: **good**.

Usability: Documentation is sparse, homepage and maintenance do not seem to be quite up-to-date. Assessment: **poor**.

[11] http://wortschatz.uni-leipzig.de/~cbiemann/software/unsupos.html

7. Eric Brill's simple rule-based part of speech tagger[12]

   Availability: Source code accessible at Plymouth Tech.[13] Assessment: **good**.

   Features: Based on rules derived from a training corpus. No lemmatization. However, absence of lemmatization may be resolved by modifying the code and implementing a lexicon (if the tagger is open source). Assessment: **fair**.

   Code: Originally implemented in C. Also implemented in Python as part of NLTK[14]. An interesting implementation of Brill's ideas combined with an HMM approach is the Erlangen-Tagger[15] though documentation of this approach seems poor and it is not open source. The same approach however is applied by Sujit Pal[16] whose Java code is available as open source (see also the next tagger reviewed here). Assessment: **good**.

   Architecture: Depending on the implementation, the tagger can be easily adopted to various conditions. Assessment: **good**.

   Usability: Especially the Python and the Java implementations come with good documentation. Assessment: **good**.

8. Sujit Pal's HMM-based tagger[17]

   Availability: Source code available from Sujit Pal's blog. Comes with a model for English derived from the Brown Corpus. Assessment: **good**.

   Features: HMM tagger based on Konchady (2006). No lemmatization. However, as the code is well-structured and not too complex, other features may be added. Sujit Pal is a software developer, not a linguist. His example makes some linguistic simplifications that may conceal the actual capabilities of his implementation. Assessment: **fair**.

   Code: Java. Assessment: **good**.

   Architecture: The code is well-documented and can easily be adopted to various needs. Assessment: **good**.

   Usability: Documentation is OK, code is clear and easy to modify. Assessment: **good**.

[12] http://en.wikipedia.org/wiki/Brill_tagger

[13] http://www.tech.plym.ac.uk/soc/staff/guidbugm/software/RULE_BASED_TAGGER_V.1.14.tar.Z

[14] http://www.nltk.org/

[15] http://www8.informatik.uni-erlangen.de/en/demosdownloads.html

[16] http://sujitpal.blogspot.com/2008/11/ir-math-in-java-rule-based-pos-tagger.html

[17] http://sujitpal.blogspot.com/2008/11/ir-math-in-java-hmm-based-pos.html

9. alias-i's LingPipe[18] toolkit

   Availability: Commercial. Assessment: **poor**.

   Features: HMM tagger. No lemmatization. Assessment: **fair**.

   Code: Java API. Assessment: **fair**.

   Architecture: Because of the Java API, integration in various settings seems feasible. Assessment: **fair**.

   Usability: Seems well-documented. POS tutorial[19] available on homepage. Assessment: **good**.

10. Jitar[20] is a simple trigram HMM POS tagger

    Availability: Open source. However, the code provided on the project homepage is incomplete.[21] Assessment: **poor** (as code is incomplete).

    Features: Simple trigram HMM tagger. No lemmatization. Assessment: **fair**.

    Code: Java. Assessment: **good**.

    Architecture: In principle, easy integration. Assessment: **fair**.

    Usability: Does not seem too complex which probably would make it fairly easy to use. However, it is maintained by just one person who already announced that Jitar development will be discontinued in favor of Jitar's C++ counterpart Citar.[22] Assessment: **poor**.

6.2.2 Taggers for Danish

Only two established taggers seem to be available although some others may be around as well. However, they may be narrowly tied to certain (closed) projects or companies.

1. CST's POS tagger

   Availability: Brill's allegedly modified code can be downloaded from the site of the *Centre for Language Technology* (CST) as "open source". CST has trained the tagger on DSL's publicly accessible PAROLE Corpus and thus derived a language model for Danish which they have decided not to give open public access to. Access is only given to an online version of the tagger after prior agreement. Neither conditions nor contents of the agreement are accessible on-site. Assessment: **poor**.

[18] http://alias-i.com/lingpipe/index.html
[19] http://alias-i.com/lingpipe/demos/tutorial/posTags/read-me.html
[20] http://github.com/danieldk/jitar
[21] Classes `LanguageModel.java` and `LinearInterpolationLM.java` have no contents.
[22] http://langkit.org/

Features: Based on Brill's rule-based framework. POS tagging only. However, CST provides a non-free lemmatizer with restricted access. Assessment: **fair**.

Code: C/C++. Assessment: **fair**.

Architecture: Restricted access to a web version only, not really suited for huge amounts of text. Assessment: **poor**.

Usability: A moderate amount of additional info and a demo is found on the tagger homepage. Assessment: **fair**.

2. VISL's Constraint Grammar parser

Availability: The VISL CG-3 software has been developed by GrammarSoft and is distributed as open source under the GNU General Public License. However, the Danish grammar DanGram is not publicly available and tagging/parsing of Danish can only be performed via text-by-text upload[23] or through a paid-for remote interface (accessed via the web). Conditions and prices have to be negotiated with GrammarSoft in advance. Assessment: **poor**.

Features: Based on the Constraint Grammar framework (Karlsson et al. (1995)), performs POS tagging, lemmatization and syntactic parsing. It is claimed to have a particularly high precision. Assessment: **good**.

Code: C++. Assessment: **fair**.

Architecture: Restricted access to web-based versions only, not really suited for larger amounts of text. Assessment: **poor**.

Usability: A comprehensive manual, a tutorial, examples, demos, and additional info is found on the homepage. Assessment: **good**.

6.2.3 Conclusions

As availability is considered a major requirement, the following taggers are of particular interest to the WP 2.1 project: Stanford, Apache UIMA, unsupos, Brill, and Sujit Pal's tagger implementations. Common to all these taggers is that they derive their language model from a training corpus and that they principally work as POS taggers only. The disadvantage of this is that lemmatization comes in as a separate process that requires specific tools or extensions to the existing implementations. In addition, unknown words, i.e. words not seen in prior training material, seem to be a problem for all taggers based on learning algorithms that produce language models.

However, some of the mentioned taggers may be modified to also take into account lexical knowledge and perform lemmatization as well, in particular Brill

[23] Uploaded texts will be added to VISL's own corpora if their copyright status permits it.

and Sujit Pal. Stanford and UIMA may be extendable as well, but their code is rather complex which probably makes the development of extensions difficult. As for unsupos, the unsupervised learning approach probably is not suitable for the needs of WP 2.1. Thus, it emerges that Sujit Pal's HMM and Brill implementations may be the most attractive solutions to start with. Stanford may be an alternative whereas UIMA seems far too complex for the needs of POS tagging only. It is a pity that both taggers specifically designed for handling Danish have severe usage restrictions, otherwise they might have been worth giving a try as well.

The conclusion is to conduct a case study with Sujit Pal's HMM implementation where it will be trained on the Danish Parole Corpus to evaluate the potential of his HMM approach. If it fails, Stanford can be considered as a fallback option.

6.3 Case study

The starting point of the case study is the Java code of Sujit Pal's HMM implementation[24] including the Java HMM library Jahmm by Jean-Marc François[25], University of Liège. Sujit Pal's demo is based on building an HMM from the Brown Corpus; in this study the tagged training corpus is the Danish PAROLE Corpus[26].

6.3.1 Building a token-based HMM

The common approach of modeling an HMM is to view the word forms of a text as visible observations and the set of possible POS tags as hidden states. In the following experiment, this approach, which is also demonstrated on Sujit Pal's blog, is applied to a setting for Danish.

The first step was to convert PAROLE from its TEI-like XML-structure to the necessary Brown input format, as illustrated here:

```
To/AC kendte/AN russiske/AN historikere/NC Andronik/NP
Mirganjan/NP og/CC Igor/NP Klamkin/NP tror/VA ikke/RG ,/XP
at/CS Rusland/NP kan/VA udvikles/VA uden/SP en/PI
"/XP jernnæve/NC "/XP ./XP

De/PP hævder/VA ,/XP at/CS Ruslands/NP vej/NC til/SP
demokrati/NC går/VA gennem/SP diktatur/NC ./XP

[...][27]
```

[24] http://sujitpal.blogspot.com/2008/11/ir-math-in-java-hmm-based-pos.html

[25] http://www.montefiore.ulg.ac.be/~francois/

[26] http://korpus.dsl.dk/e-resurser/parole-korpus.php?lang=uk

As can be seen, the original PAROLE tags have been cut off after two characters, the first character giving the POS, the second one giving a POS sub-classification. Inflectional information is not present in the tags used here – a simplification that should be avoided in a real setting.[28]

Building an HMM from PAROLE is quite straightforward and it actually does POS tagging afterwards, however some problems need to be addressed:

1. The PAROLE Corpus is quite small in size, approximately 250,000 tokens
2. The tag set has not been adjusted to the actual needs and is probably too large (25 different tags) to allow building an HMM from a corpus of this size
3. Only sentences with known word forms, i.e. such contained in PAROLE, can be tagged on the basis of this HMM

As corpus size cannot be augmented within the scope of the ongoing project, to cope with these restrictions, the tag set should be optimized as should the elements of the observable layer of the HMM. At the moment these are word forms but it might be worth trying to map them to more abstract representations by using a lexicon. The following experiment will address this approach.

6.3.2 Building a lexicon-based HMM

The idea behind this approach is to map word forms in the text to their possible POS tags prior to training and analyzing by applying a lexicon. During the training phase, possible tags for a given token constitute the observable layer and the actual tag the hidden state. Information not relevant to the disambiguation process should not be given in the tags at this state. Similarly, during tagging, the word forms of the text in question are mapped to this simple tag-set. The design of the jaPOS tagger described in Chapter 7 focuses on this approach.

[28] A comprehensive account of the the tag set used in PAROLE can be found in Keson (1998b) and an abridged version in Keson (1998a).

Chapter 7

Design of the ePOS tagger

Making words tell who they are

DK-CLARIN WP 2.1 deliverables concerned

D10 Lemmatizer It is considered indispensable that corpus texts need to indicate the lemma form of each inflected word form in the corpus to let the user of the corpus perform more flexible queries. Therefore, it is necessary to either develop or configure a lemmatizer (that may be based on a full-form lexicon or a morphological analyzer). In the context of WP 2.1, a lemmatizer designed as an integral part of a POS tagger is the preferable solution. **Outcome:** Tool with documentation.

D11 POS tagger In order to tag tokens in corpus texts with part-of-speech information, it is necessary to either develop or configure a POS tagger (either based on a full-form lexicon or a morphological analyzer) and a suitable tag set. **Outcome:** Tool with documentation.

Outline of this chapter

This chapter gives an account of the ePOS tagger that is in part based on Sujit Pal's HMM implementation outlined in Chapter 6.[1] The Danish PAROLE corpus is chosen as source for the language model that the tagger needs in order to work. Even if the quality of the PAROLE corpus is fairly high, it comprises some inconsistencies and mistakes that need to be adjusted before it is viable as a source for a language model. The modifications undertaken in order to enhance the PAROLE corpus are described in Section 7.1. The concepts and functionality of ePOS as well as the tag set and the construction of the language model are the main topics of Section 7.2.

7.1 Modifications of the PAROLE Corpus

The most important considerations on text formats as outlined in Chapter 4 were to keep things as simple as possible. This means that the process of segmenting a text into smaller units should not imply linguistic prior knowledge of any kind. Instead, a mechanical, algorithmic approach is preferred. However, this introduces some intricacies when using PAROLE as a basis for the language model as PAROLE applies a linguistically informed approach to the concepts of sentences and words, and it segments the texts accordingly prior to the POS annotation process. PAROLE thus cannot be used as is as input to a language model that will be applied on material segmented by another, more mechanical approach than PAROLE. Therefore, some modifications of PAROLE were inevitable. These modifications – totaling to nearly 9000 cases – are described in detail in the following sections. The resulting modified version of PAROLE (*PAROLE Version 2*) is freely available upon request.

7.1.1 Sentences

PAROLE is subdivided into sentence-like textual units enclosed by <s> and </s> tags. Feeding the language model builder with this kind of textual chunks means

[1] http://sujitpal.blogspot.com/2008/11/ir-math-in-java-hmm-based-pos.html

that the material to be POS-tagged later on also should resemble that form to a certain extent. This requires a PAROLE type of sentence splitter to be applied prior to (or during) tagging. Punctuation within sentences may to some extent help building a reliable language model during the training phase but must then also be part of the input to be tagged.

The simplest solution would be to work on material without sentence boundary markers but take into account punctuation during training and tagging as this implies a minimum of preprocessing, i.e. just pre-tokenization yielding basic tokens. However, as Sujit Pal's HMM tagger requires the input material during training and tagging to be divided into sentence-like chunks, we end up with a solution where the material is split into individual sentences by some kind of sentence splitting algorithm (baked into the tagger) and where we take into account sentence-internal punctuation.

7.1.2 Tokens and token boundaries

ePOS is entirely based on the concept of *basic tokens* defined in Chapter 4 and has no linguistic concept of what a word is. So, in ePOS, a word is just a string delimited by characters defined as token boundaries. Token boundaries are either space characters or punctuation characters.[2] These basic tokens need to be tagged in some sensible way even in cases where they do not correspond to linguistic concepts of what a word is. The strictly mechanical token concept has certain implications:

Multiword units: As space characters always are treated as token boundaries there is no concept of multiword units. Each token of such a unit is tagged individually.

Punctuation: Abbreviations, numbers, or hyphenated compounds containing punctuation characters like stops, commas, apostrophes, or hyphens are split into basic tokens at the position of the punctuation character. Each of these basic tokens has to be tagged individually.

Tagging parts of what is normally considered words may in some cases seem weird. However, tagging will most likely show a higher degree of consistency as no linguistic knowledge must be provided nor maintained. The tag set applied to handle these special cases is described in Section 7.2.

As the PAROLE corpus applies a more linguistically informed token concept, it allows tokens to contain characters that are considered non-token characters in our context.[3] Therefore, the tokens of PAROLE and their tags are converted into basic tokens (i.e. the type of tokens defined by DK-CLARIN) prior to using this corpus for building a language model. In detail, all tokens containing space, stop, hyphen, slash, backslash, comma, or apostrophe characters must be converted

[2] A list of punctuation characters is found in Table 5.1 in Chapter 5.

[3] Details are listed in the appendix of Keson (1998b).

into their DK-CLARIN equivalents. Table 7.1 shows the number of words in PAROLE that need to be split.

The process of splitting such words, which affects approximately 7600 tokens, was carried out manually and semi-automatically.

7.1.3 Other PAROLE modifications

Apart from re-tagging split words, the following modifications of the PAROLE corpus were carried out:

- Text errors (spelling errors, typos, inflectional errors, etc.) that were tagged XX in PAROLE were manually corrected and re-tagged. All 1053 XX tags in PAROLE 1.x have thus been converted into meaningful tags in PAROLE 2.0 instead.
- During the process of manually modifying XX tags some hundred other tagging errors were identified and corrected.

7.2 The ePOS tag set for Danish

The tag set applied in the ePOS tagger provides POS and inflectional information, i.e., ideally, for each possible inflectional form of a lemma a corresponding, unambiguous tag is assigned. Tags outside this strictly inflectional scope, e.g. on syntax, semantics, or morphological composition of lemmas, are currently not provided.

As the tagger is trained on the Danish PAROLE Corpus, the tag set of the ePOS tagger will be based on that one used by PAROLE (see Keson (1998a) and Keson (1998b)), however with some modifications, some of them as a consequence of the modified token concept, cf. Section 7.1.2, some of them for simplification reasons in order to hopefully achieve a better language model. The ePOS tagger utilizes a full-form lexicon that is compiled from the following three resources:

- An existing full-form lexicon derived from an earlier version of The Danish Dictionary: *FLEXIKON*.
- The freely available *FLEXION* lexicon established by Ole Norling-Christensen and others, called *ONCLEX* in the following to better distinguish it from FLEXIKON. [4]
- A supplementary lexicon derived from the tagged PAROLE material.

The ePOS full-form lexicon is described more in-depth in Chapter 8.

[4] ONCLEX can be downloaded from:
http://korpus.dsl.dk/e-resurser/boejningsformer_download.php?lang=en

7.2.1 Tag structure

The PAROLE tag set is positional which means that within a tag a certain inflectional marker is always found at a fixed position in a sequence of markers making up the tag. For example, in the case of nouns, the gender marker is always found at position 3, number at position 4, case at position 5, and definiteness at position 8. However, the positional system is dependent on the POS in question: In the case of verbs, which also may carry nominal inflectional markers, definiteness is still at position 8, but gender is at 4, number at 6, and case at 11 (see Keson (1998a) and Keson (1998b)). These varying positions would make it very cumbersome later on to perform corpus queries of the type *find all words that are marked for case = "genitive" no matter what their POS is*. The ePOS tag set therefore favors fixed, POS-independent marker positions.

In contrast to ePOS and PAROLE, the tag set applied by FLEXIKON and ONCLEX is a compact tag set that leaves out non-applicable and implicit information. Hence, it is easy to decode for humans, but may be difficult to formulate complex morphological corpus queries on. Therefore, the tag sets of these sources needs to be converted into a positional one with fixed marker positions independent of the POS in question.[5]

The basic structure of an ePOS tag is:

`CLASS:nominal:verbal:additional`

where *CLASS* is a two-character POS classifier comprising a POS indicator (first character) and a sub-classifier. The first colon indicates the boundary between the *CLASS* part and the inflectional part of the tag. Here, *nominal* and *verbal* are strings of markers concerning nominal and verbal morphological information respectively. The *additional* string carries further markers relevant to adjectives, some adverbs, and pronouns. Marker strings have fixed lengths – nominal 4, verbal 2, and additional 4. Each marker in such a string is represented by one *character*. The three groups of morphological markers are separated by colons (`:`). A morphological marker which is irrelevant to a certain paradigm is marked with a dash (**–**) at the respective position(s) of the tag. In cases where inflectional markers are underspecified, this is indicated by a hash sign (**#**) at the respective position(s), meaning *any value of this category*. In certain cases PAROLE applies tags that are underspecified beyond the level of underspecification that the ePOS tag set envisages, that is, they could all be disambiguated by looking at the context in which they occur. Instead of manually disambiguating these tags in the PAROLE corpus prior to using it as a source for the language model of the tagger, they have

[5] The CST tag set used in the CST tagger (see Section 6.2.2), which is also based on PAROLE (and seems in part to silently utilize ONCLEX as well), applies compact tags too, cf. the description of the CST tag set at

▷ http://cst.dk/online/pos_tagger/rapport/bilag/tagset.html.

The same applies to the VISL tag set.

been adopted by ePOS. In these cases, underspecified markers are marked with a section sign (§). In order to further adapt PAROLE to ePOS, the ePOS-tagged version of PAROLE should be examined and §-underspecifications should be manually disambiguated.[6] The applied markers in ePOS reflect the choices made in PAROLE, cf. its documentation (Keson (1998a) or Keson (1998b)).

Nominal markers

The string of *nominal* markers is of length four, it carries the following markers:

1. **Number** (NUM): *singular* (**s**) or *plural* (**p**)
2. **Definiteness** (DEF): *indefinite* (**i**) or *definite* (**d**)
3. **Case** (CAS): *unmarked* (**u**), *genitive* (**g**), or *fossilized* (**f**), and – for personal pronouns only – *nominative* (**n**) (*accusative* is identical with *unmarked* in these cases and tagged with **u**)
4. **Gender** (GEN): *common* (**c**) or *neuter* (**n**)

Like PAROLE, ePOS considers *gender* an inflectional category – not only of adjectives and verbal participles but for nouns as well, whereas ONCLEX leaves out any explicit information on the gender of a noun and considers this phenomenon as inherent to them.

Verbal markers

The string of *verbal* markers comprises two marker positions:

1. **Tense** (TMP): *present* (**s**), *past* (**t**)
2. **Voice** (VOC): *active* (**a**), *passive* (**p**)

Additional markers

Finally, *additional* markers constitute a heterogeneous group of the following four markers:

1. **Degree** (DEG, adjectives and some adverbs): *positive* (**p**), *comparative* (**c**), *superlative* (**s**), *absolute superlative* (**a**)
2. **Person** (PER, personal and possessive pronouns): *first* (**1**), *second* (**2**), *third* (**3**)
3. **Reflexiveness** (RFL, personal and possessive pronouns): *yes* (**y**) or *no* (**n**)
4. **Possessor** (POS, possessive pronouns): *singular* (**s**) or *plural* (**p**)

The following structure shows the positions of the tags used in ePOS:

All three marker strings are always present in a tag even if some of them are unused (–) or underspecified (# or §). This ensures that it is always straightforward to query on certain marker positions regardless of the POS in question.

The *CLASS* part of the tag comprises two characters. The first character indicates part of speech, the second one may indicate a subclass. If there is no subclass, the second character is a *dash* sign (–). Otherwise, for inflecting words, the subclass always is triggered by a variation of the inflectional paradigm of that particular POS, e.g. participle forms of verbs are characterized by the paradigm VP:⋆⋆⋆⋆:⋆–:---- with inflectional markers (from the table and descriptions above) occurring at the positions marked ⋆ whereas finite forms have their inflectional markers according to the paradigm VF:----:⋆⋆:----. Hence verbs come in finite (tagged VF) or in participle (VP) flavor – as well as in a number of other inflectional paradigms, cf. Table 7.3 in the next section.

7.2.2 POS markers and subclassifiers in ePOS

7.2.2.1 Class tags

Table 7.3 shows the *class tags* used in ePOS, i.e. the symbols used at the first position of the ePOS tags indicating the part-of-speech (Column *POS*) as well as the symbols at the second positions of the ePOS tags giving a potential subclass (Column *Sub.*). The *Paradigm* column shows the structure of the full tag where marker positions with an ⋆ carry inflectional information (denoted by one either character from Table 7.2 or # or § as discussed in Section 7.2.1), whereas positions with a dash are unused within the given class. The only exception from this is the VT tag marking the past participle form of verbs that has a constant string of inflectional markers (siu#:t-:----).

As the token concept (cf. Section 7.1.2) underlying ePOS considers hyphens and apostrophes as token delimiters, we have to deal with special cases of tokens that are not words themselves but parts of words. These may be tagged as *lexical elements*, *inflectional morphemes*, or *word formation elements*. Further details about these types of tokens can be found in Chapter 8.

7.2.2.2 Lexical elements and inflectional endings

Lexical elements are parts of words that cannot be tagged as regular parts-of-speech. They are often prefixes attached to a word by a hyphen. As such they always play a role in lexical word formation. Lexical elements do not possess any morphological markers, thus the corresponding part of the tag is always set to ----:--:----. Examples of such elements are

	anti-	*social*	*adfærd*
▷	EW:----:--:----	AC:siuc:--:p---	NC:siuc:--:----

[6] A future project worthwhile to consider.

▷ *øko-* *tapas*
EW:----:--:---- NC:piu#:--:----

Inflectional endings are grammatical morphemes attached to a noun, verb, or adjective by an apostrophe. Some examples of this are

▷ *cv* *'er*
NC:siun:--:---- MN:piu#:--:----

▷ *vinderen* *ta* *'r* *det* *hele*
NC:sduc:--:---- VI:----:-a:---- MV:----:sa:---- PM:s-un:--:---- AC:sdu#:--:p---

▷ *det* *go* *'e* *vejr*
PM:s-un:--:---- AC:siuc:--:p--- MA:§§u§:--:p--- NC:siun:--:----

Inflectional endings are lexicalized in the full-form lexicon as a special case of lemmas. They can easily be identified as they all start with an @ character.

7.2.2.3 Word formation elements

Another effect of the token concept is that virtually any POS marker can also occur with the *word formation* subclassifier W (In Table 7.3 this is only listed for the the lexical element E that occurs with the W subclassifier only). Word formation elements do not possess any morphological markers, thus the corresponding part of the tag is always set to ----:--:----, e.g. a noun token taking part in word formation is tagged with NW:----:--:----. Some examples are listed below.

▷ *sidde-* *eller* *sovepladser*
VW:----:--:---- CC:----:--:---- NC:piu#:--:----

▷ *super-* *formand*
AW:----:--:---- NC:siuc:--:----

▷ *planlægnings-,* *forvaltnings-,* *og* *serviceopgaver*
NW:----:--:---- NW:----:--:---- CC:----:--:---- NC:piu#:--:----

▷ *den* *15-* *årige* *rocktøs*
PM:s-uc:--:---- LW:----:--:---- AC:sdu#:--:p--- NC:siuc:--:----

▷ *fanden-* *i-* *voldsk*
NM:----:--:---- TW:----:--:---- AC:siu§:--:p---

7.2.2.4 PAROLE's *residual* group in ePOS

Tokens that cannot be identified as a regular part-of-speech are assigned to the *residual* group in PAROLE. This group comprises abbreviations (tagged XA), foreign words (XF), formulae (XR), symbols (XS), punctuation (XP), and other (XX). In ePOS, these subclasses have been reduced to XS (symbols) and XF (foreign)

only, i.e. abbreviations, formulae, symbols, and other have been collapsed into XS whereas XF is maintained, and XP is omitted. The XY tag comprises cases where no adequate tag could be assigned, either because the token could not be identified (not in the lexicon) or because the language model could not handle the token in question (even if it is in the lexicon). All tags of the residual group have their morphological part set to ----:--:----.

Character	Count
Period	2488
Space	1556
Hyphen	2831
Comma	150
Apostrophe	412
Slash	141
Colon	8
Brackets	2
Total	7588

Table 7.1: Number of PAROLE words that are split into basic tokens

CLASS	*nominal*				*verbal*		*additional*			
	NUM	**DEF**	**CAS**	**GEN**	**TMP**	**VOC**	**DEG**	**PER**	**RFL**	**POS**
	s	i	u	c	s	a	p	1	y	s
	p	d	g	n	t	p	c	2	n	p
			f				s	3		
			n				a			

Table 7.2: Inflectional markers

POS		Sub.		Paradigm
V	**Verb**	I	infinitive	VI:----:-*:----
		F	finite	VF:----:**:----
		M	imperative	VM:----:--:----
		G	gerund	VG:****:--:---
		P	participle	VP:****:*-:----
		T	past part.	VT:siu#:*-:----
		D	adv. part.	VD:----:*-:----
A	**Adjective**	C	common	AC:****:--:*---
		D	adverbial	AD:----:--:*---
L	**Numeral**	C	cardinal	LC:--*-:--:----
		O	ordinal	LO:--**:--:----
N	**Noun**	C	common	NC:****:--:----
		P	proper	NP:****:--:----
P	**Pronoun**	C	reciprocal	PC:*-*-:--:----
		M	demonstrative	PM:*-**:--:----
		I	indefinite	PI:*-**:--:----
		O	possessive	PO:*--*:--:-***
		P	personal	PP:*-**:--:-**-
		R	relative	PR:*-**:--:----
D	**Adverb**	-		D-:----:--:*---
I	**Interjection**	-		I-:----:--:----
T	**Preposition**	-		T-:----:--:----
C	**Conjunction**	C	coordinating	CC:----:--:----
		S	subordinating	CS:----:--:----
U	**Unique**	I	inf. marker	UI:----:--:----
		S	*som/der*	US:----:--:----
E	**Lexical element**	W	word formation	EW:----:--:----
M	**Inflectional ending**	N	attached to a noun	MN:****:--:----
		V	attached to a verb	MV:----:**:----
		A	attached to an adj.	MA:****:--:*---
X	**Residual**	S	symbol	XS:----:--:----
		F	foreign	XF:----:--:----
		Y	tagging error	XY:----:--:----

Table 7.3: POS markers and subclassifiers

Chapter 8

The full-form lexicon

Same word, different versions

Deliverables concerned

D9 Full-form lexicon Development and/or configuration of a full-form lexicon for POS tagging. **Outcome:** Resource with documentation.

D10 Lemmatizer It is considered indispensable that corpus texts need to indicate the lemma form of each inflected word form in the corpus to let the user of the corpus perform more flexible queries. Therefore, it is necessary to either develop or configure a lemmatizer (that may be based on a full-form lexicon or a morphological analyzer). In the context of WP 2.1, a lemmatizer designed as an integral part of a POS tagger is the preferable solution. **Outcome:** Tool with documentation.

D11 POS tagger In order to tag tokens in corpus texts with part-of-speech information, it is necessary to either develop or configure a POS tagger (either based on a full-form lexicon or a morphological analyzer) and a suitable tag set. **Outcome:** Tool with documentation.

Outline of this chapter

This chapter describes the anatomy of the full-form lexicon that is used for part-of-speech (= POS) tagging. It gives an introduction to material that existed prior to the development of the ePOS tagger (Section 8.1) and provides an account of how this material was enhanced in order to suit the needs of ePOS tagging. Finally, in Section 8.2, the ePOS full-form lexicon is described in detail.

8.1 Enhancing existing material

The input to the full-form lexicon we need for tagging (see Chapter 7) derives from three lexical resources: ONC-Flexion, Flexikon, and – to a certain extent – the PAROLE Corpus itself, cf. Figure 8.1. ONC-Flexion was derived from existing machine-readable dictionaries in the early 1990s and used as a basis for inflectional information in The Danish Dictionary, DDO. Flexikon was derived from an early version of the DDO around 2000 and used for various purposes in conjunction with the Korpus 2000 website. The PAROLE Corpus, on which the initial language model of the ePOS tagger is based, provides additional lexical entries, however, these entries are not verified and are therefore kept separately in an auxiliary lexicon. At a later point in time new corpus material will be used to enhance the ePOS lexicon that is meant to sever as a primary source for inflectional information in the DDO. The following sections give a more detailed account of the lexical sources of ePOS.

8.1.1 ONC-Flexion

8.1.1.1 Description

ONC-Flexion[1] is a full-form list with information on parts of speech and inflection for about 80,000 lemmas. ONC-Flexion was originally developed by Ole Norling Christensen in the early 1990s in order to facilitate the process of writing The Danish Dictionary, DDO. ONC-Flexion has since been enhanced by the Korpus 2000

[1] Free download from:
http://korpus.dsl.dk/e-resurser/boejningsformer_download.php?lang=en

project and a free version of it can be downloaded through the ordnet.dk website. As ONC-Flexion is the most comprehensive and elaborate full-form lexicon of Danish currently freely available, it is used as the major source of the ePOS full-form lexicon.

The lemmas of ONC-Flexion originate from various older sources from the 1980s, and their inflectional forms have been derived from the source information and automatically supplemented in a number of cases. The selection is very wide, and a number of words are hardly relevant (e.g. proper nouns, nonce formations). The majority, however, are words also included in The Danish Dictionary, DDO. In addition, DDO also includes other, particularly newer words that are not in the list.

The structure of the full-form list may be illustrated by the following example:

```
*
certifikat
S
2 certifikat
4 certifikats
8 certifikatet
16 certifikatets
32 certifikater
64 certifikaters
128 certifikaterne
256 certifikaternes
```

A new lemma is always preceded by an asterisk (*) on a separate line. In the following line the lemma appears in its lemma or base form, in line 3 its part of speech is given. The following part of speech markers occur:

```
S: noun
A: adjective
V: verb
D: adverb
F: abbreviation
K: conjunction
L: onomatopoeic word
O: pronoun
P: proper noun
I: prefix
Æ: preposition
T: numeral
U: interjection
X: unidentified
```

Class X comprises words that could not be immediately identified during automatic analysis of the sources. In particular it contains words which usually occur exclusively in fixed expressions with other words, e.g. *badut* (*springe badut*), *bero* (*stille i bero*), *besøgelsestid* (*kende sin besøgelsestid*) or multi word units behaving as a single word, e.g. *au pair*.

In addition to prefixes proper, e.g. *di-*, *eks-*, *fore-*, class I also comprises the first elements of compounds, e.g. *forenings-*, *forhandlings-*, *formue-*.

Class P comprises proper nouns (i.e. in principle, nouns), but as it is almost impossible to apply usable selection criteria discerning important from unimportant within the class, it is characterized by a degree of coincidence.

Class F reflects primarily an orthographic phenomenon. For all abbreviations, a genitive form with apostrophe *-s* has been generated although many of these seem questionable.

The line containing part of speech is followed by the different orthographic forms which the word may take. The forms are always given in small letters even if capital letters are used in the normally correct orthographic representation. Hyphens, full stops (in abbreviations) and spaces (e.g. *a la*), if any, are also omitted. The omitted information can, however always be derived from the base form.

Each orthographic inflectional form in the list is preceded by a number (separated from the word by a tabulator) indicating which inflectional forms of the lemma the form can be assigned to.

The numbers refer to the bits which have been placed in the following bit pattern:

position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768

position 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 value 1 2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 16384 32768 If the bits in position 6, 10 and 11 have been entered, the number becomes 64+1024+2048=3136. Different inflectional forms are attached to each position depending on the part of speech. The following tables show the forms attached to the individual positions in the bit pattern for relevant part of speech.

has been built by algorithmically generating full forms based on lemma forms and inflectional information of The Danish Dictionary (DDO). As the structure of inflectional information in the DDO is suboptimal for NLP exploitation a minor group of the automatically generated forms are erroneous, especially composite nouns.

8.1.1.2 ePOS adaption

ONC-Flexion is based on orthographic forms: each orthographic form is only listed once may have attached several inflectional functions. jaPOS is entirely based on inflectional categories, so the same orthographic forms may be listed under several categories.

ONC-Flexion is based on a slightly different token concept than jaPOS as full stop, hyphen, and apostrophes are considered token characters and not boundaries as is the case in jaPOS. This means that words containing one or more of these characters need special attention. The adaption algorithm identifies these cases in ONC-Flexion and rejects them and puts them on a special list that is checked manually.

8.2 Anatomy of the ePOS lexicon

8.3 Inflectional paradigms

8.3.1 Nouns

8.3.2 Lexical and inflectional elements

More on how these forms are tagged can be found in Chapter 7

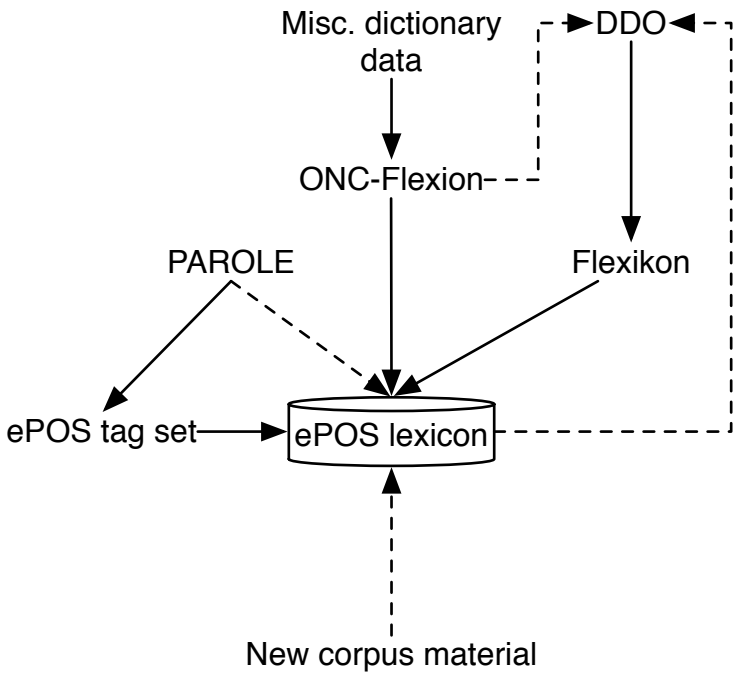

Figure 8.1: Sources of the ePOS lexicon

Part V

Deployment

Chapter 9

Corpus specifications

The ingredients

Deliverables concerned

D18 Final version of corpus Final version of POS-tagged corpus of 45 million words available for the DK-CLARIN repository and accessible through a web-based (or other) concordance tool. **Outcome:** Resource with documentation.

Outline

This chapter describes the composition of the corpus, the text material included, and how the corpus can be accessed.

9.1 Corpus composition

The following table shows from which sources the text material included in the DK-CLARIN corpus were drawn.

Type	Source	Period	Capture	Text Items	Tokens	Remarks
bg	Bentes blog	2008–2011	dsn.dk			
bg	Blogbogstaver	2005–2011	dsn.dk			
bg	Blogsbjerg	2007–2011	dsn.dk			

9.2 Text material

9.2.1 Wikipedia

- Headlines are left out in wikipedia articles which constitutes a severe problem as certain parts of these articles not relevant in a corpus context not can be filtered out, especially the references and links.
- List and tables articles have not been removed. Table articles contain lots of | chars that were erroneously classified as words but have been reclassified into punctuation characters by applying textjuggler.TextCleaner.
- Space characters in <title> elements are erroneously replaced by underscores. Some titles end with a hex number.

9.3 Corpus access

Chapter 10

Corpus access

Some usage scenarios

Deliverables concerned

D14 Prototype of concordance tool A web-based concordance tool needs to be configured/implemented as a prototype for testing. **Outcome:** Report.

D17 Final version of concordance tool Web-based concordancer with public access. **Outcome:** Service with documentation.

Outline of this chapter

This chapter describes the ways of accessing the DK-CLARIN WP 2.1 *Reference corpus of general language* anatomy and gives examples of some basic usage cases.

10.1 Ways of accessing the corpus

10.2 Some usage scenarios

References

Bibliography

Andersen, M. S., Asmussen, H., and Asmussen, J. (2002). The project of Korpus 2000 Going Public. In Braasch, A. and Povlsen, C., editors, *Proceedings of the 10th EURALEX International Congress*, volume 1, pages 291–299, Copenhagen. Euralex.

Asmussen, J. (2005). Automatic detection of new domain-specific words, using document classification and frequency profiling. In *Proceedings of the Corpus Linguistics 2005 conference*, volume 1, Birmingham.

Asmussen, J. (2008). DOT's Sprogteknologiske Drejebog. Udviklingsopgaver i forbindelse med *ordnet*-projektet. Technical report, Det Danske Sprog- og Litteraturselskab, ja-korpus.dsl.lan/doc/drejebogen.pdf.

Burnard, L. (2007). Reference Guide for the British National Corpus (XML Edition). Technical report, Research Technologies Service at Oxford University Computing Services, www.natcorp.ox.ac.uk/XMLedition/URG/index.html.

Evert, S. and Giesbrecht, E. (2009). Part-of-speech tagging – a solved task? An evaluation of POS taggers for the Web as corpus. In Alegria, I., Leturia, I., and Sharoff, S., editors, *Proceedings of the 5th Web as Corpus Workshop (WAC5)*, San Sebastian, Spain.

Karlsson, F. et al. (1995). *Constraint Grammar – A Language-Independent System for Parsing Unrestricted Text*. Mouton de Gruyter.

Keson, B. K. (1998a). Documentation of The Danish Morphosyntactically Tagged PAROLE Corpus. Technical report, DSL, korpus.dsl.dk/e-resurser/paroledoc_en.pdf.

Keson, B. K. (1998b). Vejledning til det danske morfosyntaktisk taggede PAROLE-korpus. Technical report, DSL, korpus.dsl.dk/e-resurser/paroledoc_dk.pdf.

Konchady, M. (2006). *Text Mining Application Programming*. Programming Series. Charles River Media, 1 edition.

Norling-Christensen, O. and Asmussen, J. (1998). The Corpus of The Danish Dictionary. *Lexikos. Afrilex Series*, 8:223–242.