HAL
open science

PAULA XML Documentation

Amir Zeldes, Florian Zipser, Arne Neumann

To cite this version:

Amir Zeldes, Florian Zipser, Arne Neumann. PAULA XML Documentation. [Research Report] 2013, pp.38. hal-00783716

HAL Id: hal-00783716

https://inria.hal.science/hal-00783716v1

Submitted on 1 Feb 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

PAULA XML Documentation

Format version 1.1

Amir Zeldes, SFB 632 D1 <amir.zeldes@rz.hu-berlin.de>
Florian Zipser, SFB 632 D1 <f.zipser@gmx.de>
Arne Neumann, SFB 632 D1 <arne.neumann@uni-potsdam.de>

PAULA XML Documentation: Format version 1.1

by Amir Zeldes, Florian Zipser, and Arne Neumann

Version: P1.1.2013.1.21a

Publication date 21 Jan 2013

Table of Contents

Preamble vi
1. Datamodel overview 1
2. Corpus structure 3
    Corpus and subcorpus 3
    Documents 3
    AnnoSets 4
3. Required files and DTDs 6
    Minimal document structure 6
    Additional DTDs 6
4. Metadata 7
    Corpus and subcorpus metadata 7
    Document metadata 7
    Using multifeats in metadata 8
    AnnoFeats 8
5. Primary text data 10
6. Spans and markables 11
    Introduction to spans and markables 11
    Tokenizations and token markables 11
    Annotation span markables 12
    Feats 13
    Multifeats 14
7. Hierarchical structures 16
    Structs 16
    Annotating structs and rels 18
8. Pointing relations 20
9. Namespaces 22
10. Special scenarios 23
    Parallel corpora 23
    Dialogue data 24
    Aligned audio/video files 25
11. Naming conventions 27
12. Older versions and deprecated components 29
    Pointing relations in feats 29
    Virtual markables 29
    Synopsis of older PAULA versions and components 30
References 32

List of Figures

1.1. Datamodel for (sub)corpus and document tree 1
1.2. Datamodel for document-internal objects 2
2.1. Directory structure for a PAULA corpus 3
7.1. Syntax tree for "he takes people out to fish" 17
9.1. Directory structure for a PAULA corpus 22
10.1. Directory structure for a document with two parallel texts. 23
10.2. Pointing relations aligning the English text to the German text. 23
10.3. Directory structure for a document with bidirectional alignment. 24
10.4. Dialog data to be modelled in PAULA. 24
10.5. *A* `mark` *file defining the span of tokens aligned with a multimedia file.* 25
10.6. A `feat` file giving the name of the multimedia file. 26
12.1. *A* `mark` *file containing a pseudo-hierarchical markable of the deprecated "virtual" type.* 30

List of Examples

2.1. An `annoSet` file for doc1 in mycorpus 4
2.2. An `annoSet` file for the corpus `mycorpus` with three documents 5
4.1. Metadata for the corpus `mycorpus` 7
4.2. Metadata for the document `mycorpus/doc1` 7
4.3. Multiple metadata annotations in one file using `multiFeat` elements. 8
4.4. An `annoFeat` file for doc1 in mycorpus 8
5.1. A primary text data file 10
6.1. Tokenization of the `primary text data` "This is an example." 11
6.2. Tokenization of the primary data "he takes people out to fish" 11
6.3. Markables of the type "chunk" above a set of six tokens "I" "'ve "picked" "the" "kids" "up" 12
6.4. Annotating tokens with `feat` annotations for part of speech 13
6.5. Annotating spans from a markable file with `feat` annotations for chunk type 14
6.6. Annotating multiple annotations using `multiFeat` elements. 14
7.1. Constructing a hierarchical syntax tree with `struct` elements type 16
7.2. Annotating nodes from a `struct` file with `feat` annotations for phrase category: "cat" 18
7.3. Annotating edges from a `struct` file with `feat` annotations for phrase function: "func" 18
8.1. Pointing relations between token nodes to annotate dependencies of type "dep" 20
8.2. Annotating the grammatical function "func" for dependency pointing relations 20
10.1. A primary text data file 24
10.2. Tokenization for Speaker1 25
12.1. A deprecated pointing relation `Feat` file. 29

Preamble

PAULA XML or *PAULA* for short (*Potsdamer AUstauschformat Linguistischer Annotationen*, 'Potsdam Exchange Format for Linguistic Annotations') is a standoff XML format designed to represent a wide range of linguistically annotated textual and multi-modal corpora. The format was created at Potsdam University and developed within SFB 632, the collaborative research centre "Information Structure", subproject D1, "Linguistic Database" at Potsdam University and Humboldt-Universität zu Berlin (see [Dipper2005], [DipperGoetze2005], [ChiarcosEtAl2008]). The description below represents the normative documentation for PAULA version 1.1, with some notes on previous versions of PAULA. For the latest documentation always check the PAULA Website which also contains an online HTML version of this documentation.

The standoff nature of PAULA refers to the fact that each layer of linguistic annotation, such as part-of-speech annotations, lemmatizations, syntax trees, coreference annotation etc. are stored in separate XML files which refer to the same raw data. In this manner annotations can easily be added, deleted and updated without disturbing independent annotation layers, and discontinuous or hierarchically conflicting structures can be represented. Additionally the format ensures the retainment of unaltered raw data, including white space and other elements often lost due to restrictions of the encoding format. As a generalized XML format, PAULA is indifferent to particular names or semantics of annotation structures. It concentrates instead on the representation of corpus data as a set of arbitrarily labeled directed acyclic graphs (so called multi-DAGs, wherein annotation projects may contain cycles as long as these are on different annotation levels).

This documentation is structured as follows: the next chapter gives an overview of the overall data model of the current PAULA format, followed by a chapter on corpus structure for XML files and folders. Further chapters review different file types: the minimal necessary files for PAULA documents, metadata, primary text data, tokenizations and span annotations, hierarchical graphs and pointing relations. The final chapters give additional information on the optional use of namespaces, some special scenarios such as building parallel corpora, dialogue corpora and multimodal corpora, recommendations for file naming conventions and information on older/deprecated elements of the PAULA XML standard focusing on differences to the current version.

Chapter 1. Datamodel overview

PAULA projects are graphs dominated by a top level node refered to as a `corpus`. Corpus objects comprise graphs of one or more annotated `document` objects, optionally organized within a tree of `subcorpus` objects. The tree of corpus, subcorpora and documents corresponds to a file system folder tree. Corpora, subcorpora and documents can all receive `metadata` annotations.

All documents must contain at least one source of `primary text data`, possibly more in cases of parallel corpora or dialogue data, and at least one `tokenization` of this data. Tokenized data may be annotated directly using features called `feat`, such as parts-of-speech, lemmatization, etc. Further hierarchical structures can be built on top of tokens using flat span objects called `mark` (i.e. *markables*) or hierarchically nestable objects called `struct` (i.e. *structures*), which may also be annotated with `feat` objects. The type of node or annotation (part-of-speech, phrase-category etc.) is given by the type attribute of each set of nodes or annotations.

Beyond the edges resulting from the construction of hierarchies through structs, further non-hierarchical edges may be defined between any two nodes in a document using pointing relations. Both edges connecting structs to tokens or other structs and pointing relations may be annotated using feats and given a type. All objects and annotations below the document level may carry a PAULA `namespace` bundling relevant annotation layers which belong together under a common identifier (note that these are not identical with XML namespaces). The following two figures give an overview of this general data model for the corpus/document structure and the structure of objects within them. For details and examples of the individual model elements and their specific XML serialization see the next chapters.

Figure 1.1. Datamodel for (sub)corpus and document tree

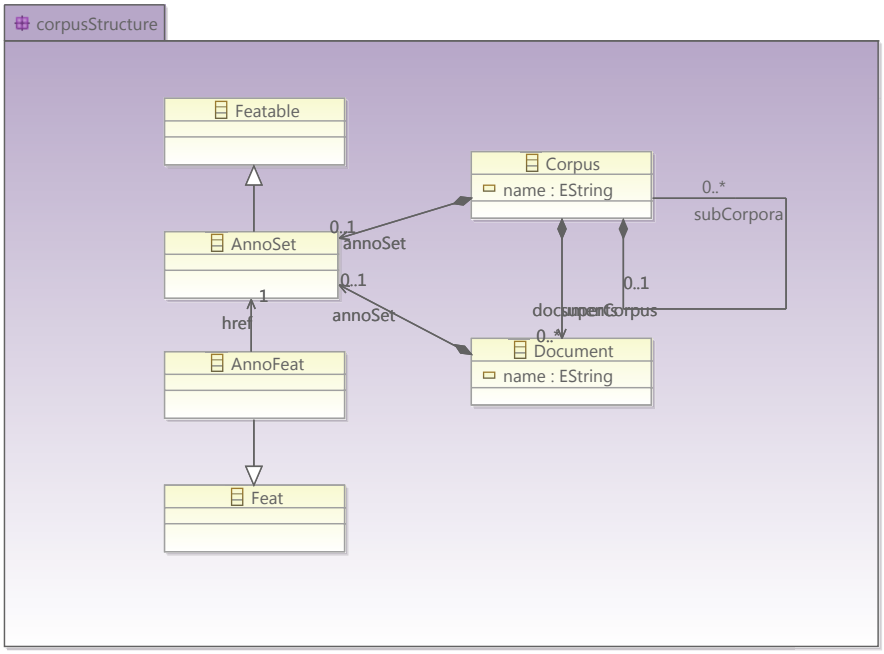
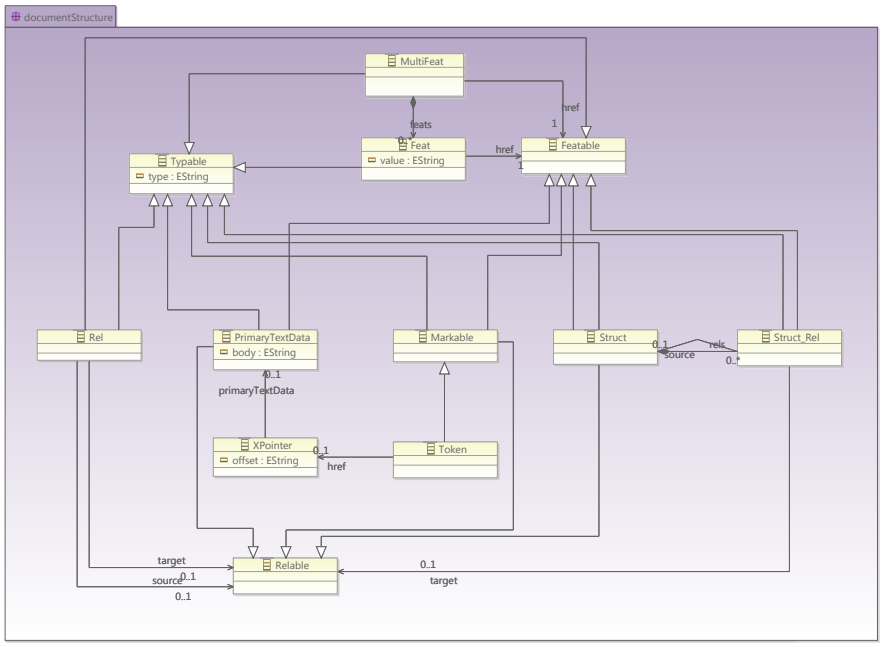


Figure 1.2. Datamodel for document-internal objects



Chapter 2. Corpus structure

Corpus and subcorpus

In PAULA a corpus structure is defined by means of a file system folder structure. The name of the corpus is determined by the name of the top level directory of the folder structure. The top level directory may contain further directories. If these directories contain subdirectories themselves, then they are considered to be subcorpora. Subcorpora are generally used to provide meaningful subdivisions of a corpus, e.g. based on genre, period, language etc. These may be accompanied by appropriate metadata.

Each subcorpus carries the name of its directory. It is possible, but not recommended, to repeat subcorpus names at different levels of nesting. A directory cannot contain two identically named subdirectories, and therefore it is impossible for two sibling subcorpora to have the same name. Under *NIX systems it is possible to have directories with identical names except for capitalization. This is not recommended for compatibility with other operating systems. In addition to directories, a top level corpus or a subcorpus may contain an `annoSet` file, which lists the set of subfolders in the same directory (see annoSets). This is not required unless the corpus or subcorpus should receive metadata annotations (see metadata).

Figure 2.1. Directory structure for a PAULA corpus

```
+-- mycorpus/
¦   +-- subcorpus1/
¦   ¦   +-- doc1/
¦   ¦   +-- doc2/
¦   ¦   +-- doc3/
¦   +-- subcorpus2/
¦   ¦   +-- doc4/
¦   ¦   +-- doc5/
¦   ¦   +-- ...
¦   +-- subcorpus3/
... ...
```

A subdirectory which contains no further directories is a document. Every corpus and subcorpus must contain at least one document (possibly nested within a lower level folder), empty corpora or subcorpora are not allowed. The minimal structure for a PAULA corpus is therefore a corpus folder containing a document folder, which must contain the minimal document structure described under documents.

Documents

A PAULA `document` is a terminal directory within the directoy structure of the PAULA `corpus`, i.e. it is a folder that contains no subfolders. Usually documents corresponds to coherent texts (e.g. an article), but in some contexts other divisions may be sensible (e.g. chapters of a book as individual documents). The primary consideration is whether or not annotations need to cross boundaries between segments of the annotated texts, since annotation nodes and edges can only exist within a document. It is not possible for an element in one document to refer to or include an element from another document.

The name of the document is determined by the name of the folder representing it. A document must contain at least a `primary text data` file, a `tokenization`, an `annoSet` file and the relevant DTDs used in the document, unless these are stored in a separate folder and refered to with appropriate relative paths. If the document contains no tokenization or other annotations, then

these will be paula_text.dtd, paula_struct.dtd and paula_header.dtd. Typically, however, a document almost always contains a tokenization of the primary text data and some annotations, meaning at least paula_mark.dtd and paula_feat.dtd (see DTDs for more information). It is generally advisable to contain all DTDs used in a corpus in every document, as redundant DTDs do not disrupt processing or validation.

By convention, all XML files within a document (i.e. all files except DTDs) share the document name as part of the file name, which appears first except for possible namespaces, and is followed by annotation layer-specific elements. For more information about recommended naming practices see naming conventions.

AnnoSets

Each PAULA document must contain an annoSet file which describes the set of annotations contained in the document. The annoSet conforms with the DTD paula_struct.dtd and contains a structList element which contains one or more struct elements, each of which contains one or more rel elements (these are the same elements used for the description of hierarchical annotations as well). Every XML file within the document directory (but not DTDs and not the annoSet file itself) must be the @xlink:href attribute of some rel in the annoSet, including the special annoFeat file if it has been included (see Annofeats). There are therefore as many rel elements in the annoSet as there are XML files in the directory, minus one (since the annoSet itself is not referenced). Different structs can be used to group together files belonging to one logical annotation layer, such as the primary text data and its tokenization, or related annotations such as part of speech and lemma. The following example shows some typical groupings following the PAULA naming conventions.

Example 2.1. An annoSet file for doc1 in mycorpus

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE paula SYSTEM "paula_struct.dtd">

<paula version="1.1">
<header paula_id="mycorpus.doc1.anno" />

<structList xmlns:xlink="http://www.w3.org/1999/xlink"
type="annoSet">
  <struct id="anno_1">
   <rel id="rel_1" xlink:href="mycorpus.doc1.anno_feat.xml" />
  </struct>
  <struct id="anno_2">
   <rel id="rel_2" xlink:href="mycorpus.doc1.text.xml" />
   <rel id="rel_3" xlink:href="mycorpus.doc1.tok.xml" />
  </struct>
  <struct id="anno_3">
   <rel id="rel_4" xlink:href="mycorpus.doc1.tok_pos.xml" />
   <rel id="rel_5" xlink:href="mycorpus.doc1.tok_lemma.xml" />
  </struct>
  <struct id="anno_4">
   <rel id="rel_6" xlink:href="mycorpus.doc1.phrase.xml" />
   <rel id="rel_7" xlink:href="mycorpus.doc1.phrase_cat.xml" />
   <rel id="rel_8" xlink:href="mycorpus.doc1.phrase_func.xml" />
  </struct>
 </structList>

</paula>
```

Annotation layers within the same struct are often interdependent, such that removing one of the files from the document may disrupt the annotation graph shared with the others. Also note that since

namespaces are also used to group related annotation layers together, often (but not necessarily always) layers with the same namespace will also be in the same `struct` in the `annoSet`.

A second function of annoSets is to list the contents of corpora or subcorpora. AnnoSets within subcorpus or corpus folders are optional, though if they are missing, the contents of the folder cannot be validated against a list. AnnoSets in corpora or subcorpora are only required if the corpus or subcorpus should receive metadata annotations, in which case an `annoSet` to which the metadata features must point is required (see metadata for more information). An `annoSet` for a subcorpus or corpus can look like the following example.

Example 2.2. An `annoSet` file for the corpus `mycorpus` with three documents

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE paula SYSTEM "paula_struct.dtd">

<paula version="1.1">
<header paula_id="mycorpus.anno" />

<structList xmlns:xlink="http://www.w3.org/1999/xlink"
type="annoSet">
  <struct id="anno_1">
   <rel id="rel_1" xlink:href="doc1/" />
   <rel id="rel_2" xlink:href="doc2/" />
   <rel id="rel_3" xlink:href="doc3/" />
  </struct>
 </structList>

</paula>
```

Corpus or subcorpus annoSets generally place all child subcorpora or documents within one `struct` element as in the example above, though it is not prohibited to group some items into different `struct` elements. It is also possible to mix subcorpora and documents within the same corpus or subcorpus level folder. There is no difference in notation and all immediate subfolders in the file system are simply listed: `subcorpus1/`, `doc1/` etc.

Chapter 3. Required files and DTDs

Minimal document structure

Every document within a PAULA corpus requires at least one instance of each of the following three XML file types: a `primary text data` file, a `tokenization`, and an `annoSet` file. These accordingly define the raw data, a basic segmentation of the data into minimal units and a list of the files in the directory (see documentation of the individual file types for details).

Additionally, the relevant DTDs must be added which define these file types. At a minimum, the DTDs necessary for the required files above are:

- `paula_header.dtd`
- `paula_struct.dtd`
- `paula_mark.dtd`
- `paula_text.dtd`

The DTDs may be repeated in each document to simplify moving and adding documents at any point in the corpus strucutre (as in the examples in this documentation), or else DTDs can be saved in one folder (e.g. the corpus root) and refered to from each document using a relative path.

Additional DTDs

Beyond the DTDs in the previous section, if the document contains any `feat` annotations or an `annoFeat` file, it will require the DTD `paula_feat.dtd`, and if it contains pointing relations using the `rel` element, the file `paula_rel.dtd` will also be necessary. A further DTD, `paula_multiFeat.dtd`, is needed if multiple feat annotations should be defined in one XML file, see multifeats.

Usually the necessary DTDs are repeatedly included in every document folder for validation purposes, though it is possible to include them in only one folder and refer to them from each document using a relative path (cf. the previous section). It is not necessary to include `paula_rel.dtd` or `paula_feat.dtd` for corpora or documents that do not contain pointing relations, even if some other documents in the corpus do, though it may be recommended to have the same DTDs or DTD references in all folders in case pointing relations or feature annotations are added to further corpus documents later on. The following full list of DTDs may therefore be included in every document:

- `paula_header.dtd`
- `paula_struct.dtd`
- `paula_mark.dtd`
- `paula_text.dtd`
- `paula_feat.dtd`
- `paula_rel.dtd`
- `paula_multiFeat.dtd`

Chapter 4. Metadata

Metadata encompasses annotations that apply to an entire object in the corpus structure, i.e. to a corpus, subcorpus or document. The metadata does not annotate specific elements within a text, but rather characterizes the entire container object. In PAULA XML metadata is realized in lists of `feat` elements (features), which refer to the `annoSet` of the relevant object (see annoSets). It is also possible for metadata annotations to carry a namespace, just like any other form of annotation.

Corpus and subcorpus metadata

Corpus and subcorpus level metadata can optionally be added to any corpus or subfolder containing an `annoSet`. It is not possible to add metadata to a folder not containing an `annoSet`. The following example illustrates a metadata annotation for the corpus `mycorpus`.

Example 4.1. Metadata for the corpus `mycorpus`

```
<?xml version="1.0" standalone="no"?>

<!DOCTYPE paula SYSTEM "paula_feat.dtd">
<paula version="1.1">

<header paula_id="mycorpus.meta_lang"/>

<featList xmlns:xlink="http://www.w3.org/1999/xlink"
type="lang" xml:base="mycorpus.anno.xml">
    <feat xlink:href="#anno_1" value="eng"/><!-- English -->
</featList>

</paula>
```

Since the name of the metadata attribute is determined in the the `@type` attribute of the `featList` element, it is necessary to define a separate `feat` file for each metadata annotation, unless multiFeat metadata files are used. Note also that in this example the feat is only pointing at the `struct` element "anno_1" from the `annoSet` file `mycorpus.anno.xml`. It is also possible to have multiple `feat` elements, pointing to each one of the `struct` elements in the `annoSet`. In the current version of PAULA this makes no difference: once a metadata annotation has been applied to any `struct` element in the `annoSet`, it applies to the entire object described by the `annoSet`.

Document metadata

Document metadata works exactly like corpus metadata: it is defined within a `feat` file which has the annotation name in the `featList` `@type` attribute and the value in the `feat` `@value` attribute. The `feat` element should point at a `struct` element from the document's `annoSet`. It is possible but not necessary to annotate all `struct` elements in the `annoSet`. The following example demonstrates this.

Example 4.2. Metadata for the document `mycorpus/doc1`

```
<?xml version="1.0" standalone="no"?>

<!DOCTYPE paula SYSTEM "paula_feat.dtd">
<paula version="1.1">

<header paula_id="mycorpus.doc1.meta_year"/>
```

```
<featList xmlns:xlink="http://www.w3.org/1999/xlink" type="year"
xml:base="mycorpus.doc1.anno.xml">
    <feat xlink:href="#anno_1" value="1999"/><!-- year 1999 -->
</featList>

</paula>
```

If the `annoSet` of doc1 contains several structs names "anno_1", "anno_2" etc., it is possible to annotate them all using multiple `feat` elements. This is identical to annotating just one of the elements, as in the example above: the metadata annotation "year" has been applied to the document and given the value "1999".

Using multifeats in metadata

When using a large number of metadata annotations, it is sometimes more convenient to use just one XML document to define all meta annotations. This is made possible by using `multiFeat` files. The following example illustrates the use of `multiFeat` annotations to define metadata. For more detailed information on `multiFeat` annotations see also multiFeat annotations.

Example 4.3. Multiple metadata annotations in one file using `multiFeat` elements.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE paula SYSTEM "paula_multiFeat.dtd">

<paula version="1.1">
<header paula_id="mycorpus.doc1.meta_multiFeat"/>

<multiFeatList xmlns:xlink="http://www.w3.org/1999/xlink"
type="multiFeat" xml:base="mycorpus.doc1.anno.xml">

    <multiFeat xlink:href="#anno_1">
        <feat name="year" value="2012"/>
        <feat name="language" value="English"/>
        <feat name="source_format" value="PAULA XML"/>
        <!-- ... -->
    </multiFeat>

</multiFeatList>

</paula>
```

AnnoFeats

Each PAULA document may optionally contain an `annoFeat` file listing the types of all annotation files including `mark`, `feat`, `struct` and `rel` files, for validation purposes. Not including an `annofeat` file means that the annotation layers available within the files specified in the `annoSet` cannot be validated, though it may make it easier to update annotation layers dynamically. The following example illustrates the use of the `annoFeat` file in reference to Example 2.1, “An `annoSet` file for doc1 in mycorpus” in the previous section.

Example 4.4. An `annoFeat` file for doc1 in mycorpus

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE paula SYSTEM "paula_feat.dtd">

<paula version="1.1">
```

```
 <header paula_id="mycorpus.doc1.annoFeat" />
 <featList type="annoFeat" xml:base="mycorpus.doc1.anno.xml"
xmlns:xlink="http://www.w3.org/1999/xlink">
  <feat xlink:href="#rel_1" value="annoFeat" />
  <feat xlink:href="#rel_2" value="text" />
  <feat xlink:href="#rel_3" value="tok" />
  <feat xlink:href="#rel_4" value="pos" />
  <feat xlink:href="#rel_5" value="lemma" />
  <feat xlink:href="#rel_6" value="phrase" />
  <feat xlink:href="#rel_7" value="cat" />
  <feat xlink:href="#rel_8" value="func" />
 </featList>

</paula>
```

Note that since the value of the `feat` is a string and not an ID, it is possible for multiple rels to refer to the same annotation type name. In order to disambiguate in such cases, it is possible to use namespaces, provided that these have been used in the corresponding annotation files. The value then takes the form "namespace:anno_name", e.g. "stts:pos".

The `annoFeat` file cannot be used in corpus and subcorpus directories.

Chapter 5. Primary text data

The `primary text data` forms the lowest level of resource representation, corresponding to the minimally analyzed linguistic data: a strech of untokenized plain text. The presence of at least one such file is obligatory in every PAULA `document`. Even if the resource to be annotated originates in spoken data for which a primary recording exists, its textual transcription forms the primary data. A segment of a recording is therefore seen to 'take place' in correspondences with a certain stretch of text (see Aligned audio/video files for details). The primary data follows the schema definition in `paula_text.dtd`, which must be present. The type of the file is "text", and by convention the file name ends with the extension `*.text.xml` and its paula_id is the same as the file name prefix, ending in `_text` instead of the file extension `*.text.xml`. Example 5.1, “A primary text data file” illustrates a `primary text data` file called `mycorpus.doc1.text.xml`.

Example 5.1. A primary text data file

```
<?xml version="1.0" standalone="no"?>

<!DOCTYPE paula SYSTEM "paula_text.dtd">

<paula version="1.1">
<header paula_id="mycorpus.doc1_text" type="text"/>

<body>This is an example.</body>

</paula>
```

A PAULA document can also contain more than one `primary text data` file. There are at least two scenarios where this is recommended, for which the respective sections should be consulted: parallel corpora with aligned texts in multiple languages and dialogue data with multiple simultaneous speakers.

As with other PAULA XML files, the first segment of text before a period within the filename of the `primary text data` file can be interpreted as a PAULA `namespace`. In documents with only one such file, this is usually not important, but it is possible to use namespaces to group together text from different languages or speakers in parallel corpora or dialogue data respectively.

Chapter 6. Spans and markables

Introduction to spans and markables

In PAULA it is possible to define spans of data for further annotation. Spans are defined using the `mark` element, which stands for *markable* and has two primary functions: defining a tokenization for a primary text data and defining a non-terminal annotation span node above the token level.

Tokenizations and token markables

A `tokenization` forms a minimal level of analysis that segments a `primary text data` file into units that can be annotated further. It is not possible to directly annotate text that is not tokenized, and every PAULA document must contain at least one `tokenization`. It is possible to include whitespace characters within the primary data and then ignore these characters while tokenizing, so that adjacent tokens are not interrupted by any characters on the tokenized level. Example 6.1, “Tokenization of the `primary text data` "This is an example."” illustrates this principle.

Example 6.1. Tokenization of the `primary text data` "This is an example."

```
<?xml version="1.0" standalone="no"?>

<!DOCTYPE paula SYSTEM "paula_mark.dtd">
<paula version="1.1">

<header paula_id="mycorpus.doc1_tok"/>

<markList xmlns:xlink="http://www.w3.org/1999/xlink" type="tok"
xml:base="mycorpus.doc1.text.xml">
 <mark id="tok_1"
  xlink:href="#xpointer(string-range(//body,'',1,4))"/><!-- This -->
 <mark id="tok_2"
  xlink:href="#xpointer(string-range(//body,'',6,2))"/><!-- is -->
 <mark id="tok_3"
  xlink:href="#xpointer(string-range(//body,'',9,2))"/><!-- an -->
 <mark id="tok_4"
  xlink:href="#xpointer(string-range(//body,'',12,7))"/><!--example-->
 <mark id="tok_5"
  xlink:href="#xpointer(string-range(//body,'',19,1))"/><!-- . -->
</markList>
</paula>
```

The first token element with the id "tok_1" begins at the first character of the text (the letter "T") and goes covering a total of 4 character: "This". Character 5 is a space, which has not been tokenized. The next token, "tok_2", begins at character 6, covering 2 characters: "is". It is also possible to define tokens with no textual extension, i.e. empty tokens. Such tokens have a string range spanning zero characters. However, they must still have an anchor position within the text. The following example illustrates an empty token in the sentence "he takes people out to fish", where the unrealized subject of "to fish" is tokenized between "out" and "to" with a character span of zero characters.

Example 6.2. Tokenization of the primary data "he takes people out to fish"

```
<?xml version="1.0" standalone="no"?>

<!DOCTYPE paula SYSTEM "paula_mark.dtd">
<paula version="1.1">
```

```
<header paula_id="mycorpus.doc2_tok"/>

<markList xmlns:xlink="http://www.w3.org/1999/xlink" type="tok"
xml:base="mycorpus.doc2.text.xml">
<mark id="tok_1"
 xlink:href="#xpointer(string-range(//body,'',1,2))"/><!-- he -->
<mark id="tok_2"
 xlink:href="#xpointer(string-range(//body,'',4,5))"/><!-- takes -->
<mark id="tok_3"
 xlink:href="#xpointer(string-range(//body,'',10,6))"/><!--people-->
<mark id="tok_4"
 xlink:href="#xpointer(string-range(//body,'',17,3))"/><!-- out -->
<mark id="tok_5"
 xlink:href="#xpointer(string-range(//body,'',21,0))"/><!--   -->
<mark id="tok_6"
 xlink:href="#xpointer(string-range(//body,'',22,2))"/><!-- to -->
<mark id="tok_7"
 xlink:href="#xpointer(string-range(//body,'',25,4))"/><!--fish-->
</markList>
</paula>
```

Although a PAULA tokenization file is defined with reference to the general markable DTD `paula_mark.dtd`, it is distinguished from other types of markables, specifically annotation markables, in two ways. Firstly, the `@type` attribute of the element `markList`, which must be set to the value `tok`. Secondly, tokenization can only refer to a `primary text data` file. It is not possible to define a token pointing to a more complex structure (e.g. another markable or token).

As of PAULA version 1.1 it is possible to have multiple `primary text data` files, each of which must then be tokenized. Multiple tokenizations of the same `primary text data` are not possible in PAULA 1.1, but are planned as part of a future version of PAULA XML.

Annotation span markables

The element `mark` may be used to group together a set of tokens for further annotation. This is usually done in order to annotate a certain feature-value pair which applies to these tokens. Span annotations therefore have the semantics of *attribution* within the graph structure, i.e. stating that an area of the data has a certain property or attribute. These attributes are realized in PAULA using `feat` annotation files, one or more of which can apply to any span defined by a markable. Span markables are defined with reference to the DTD `paula_mark.dtd`. The type of markable being annotated (e.g. a referent or referring expression in a discourse, a chunk for chunking annotation, etc.) is given by the `@type` attribute of the `markList` element, and may be any string value other than "tok" which is reserved for tokenizations. Other values are not ruled out by the format, but it is recommended to use types that follow XML element naming conventions, i.e. strings that contain only alphanumeric ascii characters with no spaces and beginning with an alphabetic character.

Markables may be continuous or discontinuous, i.e. they may apply to a set of consecutive tokens or to non-consecutive tokens. The following example illustrates both types of markables in a single file with the type "chunk".

Example 6.3. Markables of the type "chunk" above a set of six tokens "I" "'ve "picked" "the" "kids" "up"

```
<?xml version="1.0" standalone="no"?>

<!DOCTYPE paula SYSTEM "paula_mark.dtd">
<paula version="1.1">
```

```
<header paula_id="mycorpus.doc1_chunk_seg"/>

<markList xmlns:xlink="http://www.w3.org/1999/xlink" type="chunk"
xml:base="mycorpus.doc1.tok.xml">
 <!-- I -->
 <mark id="chunk_1" xlink:href="#tok_1"/>
 <!-- 've picked...up -->
 <mark id="chunk_2"
  xlink:href="(#xpointer(id('tok_2')/range-to(id('tok_3'))),#tok_6)"/>
 <!-- the kids -->
 <mark id="chunk_3"
  xlink:href="#xpointer(id('tok_3')/range-to(id('tok_4')))"/>
</markList>

</paula>
```

In the example, three markables have been defined which refer to six tokens in the token file `mycorpus.doc1.tok.xml`, as entered in the `markList` element's `@xml:base` attribute. The first markable, "chunk_1" points to "#tok_1" in the token file which covers the string "I". The third markable, "chunk_3", points to a range of consecutive tokens, from "tok_3" to "tok_4", which covers the words "the kids". The chunk in the middle, "chunk_2", points to a discontinuous set of tokens, namely a range "tok_2" to "tok_3" and a further individual token "tok_6", corresponding to the tokens "'ve picked" and a later token "up". These markables cannot be annotated further within this file (e.g. with the type of chunk as nominal, verbal, etc.). Further annotation of the markables beyond the markable list `@type` must be added in separate files as `feat` annotations.

Note that the markable type is set once in the `markList` element for all markables in the file. To define markables of a different type, a separate markable file must be generated. Separate files are not required to have the same segmentations and constitute independent layers of annotation.

Feats

The element `feat` and corresponding feat files represent arbitrary key-value feature annotations which may be applied to a variety of elements, such as parts of speech or syntactic categories, but also metadata. They can be applied to mark elements to annotate spans of tokens or even tokens directly, but also to `struct` elements as part of non-hierarchical annotations or metadata annotation of `annoSet` elements. The following two examples illustrate feature annotation of spans and tokens. For other uses see metadata and annotating structs. In Example 6.4, “Annotating tokens with `feat` annotations for part of speech” a `featList` with the `@type` "pos" contains six `feat` elements, each annotating a single token with its part of speech in the `@value` attribute.

Example 6.4. Annotating tokens with `feat` annotations for part of speech

```
<?xml version="1.0" standalone="no"?>

<!DOCTYPE paula SYSTEM "paula_feat.dtd">
<paula version="1.1">

<header paula_id="mycorpus.doc1_pos"/>

<featList xmlns:xlink="http://www.w3.org/1999/xlink" type="pos"
xml:base="mycorpus.doc1.tok.xml">
    <feat xlink:href="#tok_1" value="PP"/><!-- I -->
    <feat xlink:href="#tok_2" value="VBP"/><!-- 've -->
    <feat xlink:href="#tok_3" value="VBN"/><!-- picked -->
    <feat xlink:href="#tok_4" value="DT"/><!-- the -->
    <feat xlink:href="#tok_5" value="NNS"/><!-- kids -->
    <feat xlink:href="#tok_6" value="RP"/><!-- up -->
```

```
</featList>

</paula>
```

It is also possible to annotate more than one token at a time by using annotation span markables, which cover one or more tokens each. In this case the features do not refer to a token file, but to a markable file which refers to some tokens in itself. The following example illustrates the annotation of such spans, which works in much the same way as the annotation of tokens.

Example 6.5. Annotating spans from a markable file with `feat` annotations for chunk type

```
<?xml version="1.0" standalone="no"?>

<!DOCTYPE paula SYSTEM "paula_feat.dtd">
<paula version="1.1">

<header paula_id="mycorpus.doc1_chunk_seg_chunk_type"/>

<featList xmlns:xlink="http://www.w3.org/1999/xlink"
type="chunk_type" xml:base="mycorpus.doc1.chunk_seg.xml">
    <feat xlink:href="#chunk_1" value="N"/><!-- I -->
    <feat xlink:href="#chunk_2" value="V"/><!-- 've picked _ up -->
    <feat xlink:href="#chunk_3" value="N"/><!-- the kids -->
</featList>

</paula>
```

In this case, three features of the type "chunk_type" have been assigned to three markables in the file `mycorpus.doc1.chunk_seg.xml`. The "chunk_type" of the first markable is given the value "N". The second markable receives the "chunk_type" "V" and the third is "N" again. Note that the tokens covered by the respective markables are not defined here, though comments to the right of each element can help keep track of the text covered by each annotation. The actual tokens covered by each markable are defined in the separate file `mycorpus.doc1.chunk_seg.xml`. There is also no necessary connection between the type of feature and the type of markable, though in many cases it makes sense to give them similar names, e.g. markables called "chunk" and an annotation "chunk_type" (see also naming conventions).

Multifeats

In cases where multiple annotations always apply to the same nodes, it may be more economic to specify multiple, usually related annotations in the same file. This is made possible by the use of `multiFeat` files, together with the associated `paula_multiFeat.dtd`. Each multiFeat contains multiple feat annotations applying to the element specified in the `@xlink:href` attribute of the `multiFeat` element. Since the `multiFeat` itself is not an actual annotation, but a container for other annotations, the `multiFeatList` element is conventionally given the type "multiFeat". The example below illustrates the use of multiFeat annotations.

Example 6.6. Annotating multiple annotations using `multiFeat` elements.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE paula SYSTEM "paula_multiFeat.dtd">

<paula version="1.1">
<header paula_id="mycorpus.doc1.tok_multiFeat"/>

<multiFeatList xmlns:xlink="http://www.w3.org/1999/xlink"
```

```
type="multiFeat" xml:base="mycorpus.doc1.tok.xml">

    <multiFeat xlink:href="#tok_1"> <!-- I -->
        <feat name="pos" value="PPER"/>
        <feat name="lemma" value="I"/>
    </multiFeat>
    <multiFeat xlink:href="#tok_2"> <!-- 've -->
        <feat name="pos" value="VBP"/>
        <feat name="lemma" value="have"/>
    </multiFeat>
    <!-- ... -->

</multiFeatList>

</paula>
```

Note that there is no difference from the data model point of view between the use of multiple `feat` files or one `multiFeat` file specifying the same annotation types. Note also that when using namespaces, all annotations in a `multiFeat` have the same namespace, determined by the `multiFeat` file name. While it is possible to have different annotation in different `multiFeat` elements in the same file, it is recommended to avoid this, as it can quickly become confusing. The use of `multiFeat` annotations can also make it potentially difficult to add, remove and edit annotations after the fact, since separate annotation layers are mixed in one XML file.

Chapter 7. Hierarchical structures

Hierarchical structures are used in PAULA for two different purposes: for the creation of hierarchically nested annotation graphs (e.g. syntax trees, rhetorical structure annotation, hierarchical topological fields) and for the definition of structured `annoSet` objects (see annoSets). Hierarchical structures express the graph semantic property that a parent node *consists of* its children, or in reverse, that children nodes *constitute* their parent nodes. The semantics of hierarchical edges is also called *dominance* (a parent node *dominates* a child node), and they are consequently known as *dominance edges* as well. This chapter describes hierarchical annotation graphs. For non-hierarchical annotations see also spans and markables.

Structs

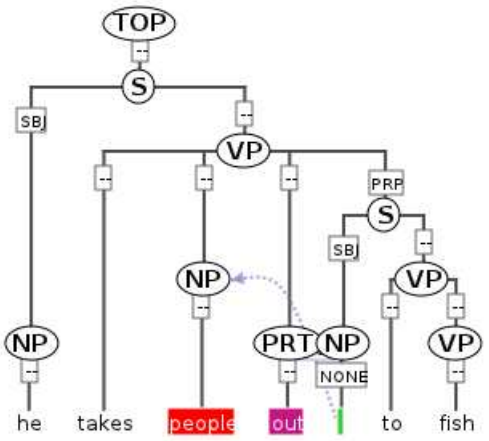
To form hierarchically nested (i.e. recursive) non-terminal nodes above the token level, the `struct` element should be used. *Directed acyclic graphs* (DAGs) of struct elements may be defined in struct files according to `paula_struct.dtd`. The `struct` element is embedded within a `structList` which determines the `@type` for all structs in the file. It has only one attribute, an `@id` which allows it to become the target of incoming edges. Outgoing edges are annotated using the child element `rel`, which has its own `@type` (the type of edge) and an attribute `@xlink:href` determining the target's id, as well as its own `@id` attribute for further annotation (see annotating structs and rels). The following example illustrates a simple syntax tree for the sentence "he ". The correpsonding syntax tree is also visualized in Figure 7.1, “Syntax tree for "he takes people out to fish"”.

Example 7.1. Constructing a hierarchical syntax tree with `struct` elements type

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE paula SYSTEM "paula_struct.dtd">

<paula version="1.1">
<header paula_id="mycorpus.doc2_phrase"/>

<structList xmlns:xlink="http://www.w3.org/1999/xlink"
type="phrase">
<struct id="phrase_1"> <!-- NP -->
 <!-- he -->
 <rel id="rel_1" type="edge" xlink:href="mycorpus.doc2.tok.xml#tok_1"/>
</struct>
<struct id="phrase_2"> <!-- VP -->
 <!-- takes -->
 <rel id="rel_2" type="edge" xlink:href="mycorpus.doc2.tok.xml#tok_2"/>
 <rel id="rel_3" type="edge" xlink:href="#phrase_3"/>
 <rel id="rel_4" type="edge" xlink:href="#phrase_4"/>
 <rel id="rel_5" type="edge" xlink:href="#phrase_5"/>
</struct>
<struct id="phrase_3"> <!-- NP -->
 <!-- people -->
 <rel id="rel_6" type="edge" xlink:href="mycorpus.doc2.tok.xml#tok_3"/>
 <!-- _ -->
 <rel id="rel_7" type="secedge" xlink:href="mycorpus.doc2.tok.xml#tok_5"/>
</struct>
<struct id="phrase_4"> <!-- PRT -->
 <!-- out -->
 <rel id="rel_8" type="edge" xlink:href="mycorpus.doc2.tok.xml#tok_4"/>
</struct>
<struct id="phrase_5"> <!-- S -->
```

```
 <rel id="rel_9" type="edge" xlink:href="#phrase_6"/>
 <rel id="rel_10" type="edge" xlink:href="#phrase_7"/>
</struct>
<struct id="phrase_6"> <!-- NP -->
 <!-- _ -->
 <rel id="rel_11" type="edge" xlink:href="mycorpus.doc2.tok.xml#tok_5"/>
</struct>
<struct id="phrase_7"> <!-- VP -->
 <!-- to -->
 <rel id="rel_12" type="edge" xlink:href="mycorpus.doc2.tok.xml#tok_6"/>
 <rel id="rel_13" type="edge" xlink:href="#phrase_8"/>
</struct>
<struct id="phrase_8"> <!-- VP -->
 <!-- fish -->
 <rel id="rel_14" type="edge" xlink:href="mycorpus.doc2.tok.xml#tok_7"/>
</struct>
<struct id="phrase_9"> <!-- S -->
 <rel id="rel_15" type="edge" xlink:href="#phrase_1"/>
 <rel id="rel_16" type="edge" xlink:href="#phrase_2"/>
</struct>
<struct id="phrase_10"> <!-- TOP -->
 <rel id="rel_17" type="edge" xlink:href="#phrase_9"/>
</struct>
</structList>

</paula>
```

Figure 7.1. Syntax tree for "he takes people out to fish"



In this example, the individual nodes in the tree from the figure above are represented by `struct` elements. Each `struct` element contains `rel` elements which define edge leading to its children. Thus "phrase_1" directly dominates a token "tok_1", corresponding to the word "he". Note that, since the tokens are in a separate file, references to the tokens give a full href attribute with the token file name: mycorpus.doc2.tok.xml#tok_1. Phrase nodes dominating other phrase nodes within the same file do not require any prefix: "phrase_9" dominates "#phrase_5" directly. Most edges in the tree have been given the edge `@type` "edge", but one edge, by which the NP above "people" (marked in red in the figure above) indirectly dominates an empty token between "out" and "to" (marked in green) with a different `@type`: "secedge" (a 'secondary' edge). There is no limit to the amount of edge types used in a document, but XML naming conventions should be followed in giving type names that are ascii

alphanumeric, without spaces and beginning with an alphabetic character (see naming conventions). The node labels ("NP", "VP") and the edge labels ("SBJ", "PRP") are not defined within the `struct` file, but are given as separate annotation files: see annotating structs and rels.

Annotating structs and rels

Hierarchical graphs made of `struct` and `rel` elements may be further annotated using `feat` elements, much like annotation spans. To annotate `struct` nodes, use a `feat` file pointing to the nodes and give the annotation name in the `@type` attribute. The following example illustrates the phrase annotations for the tree in Example 7.1, “Constructing a hierarchical syntax tree with `struct` elements type” in the previous section.

Example 7.2. Annotating nodes from a `struct` file with `feat` annotations for phrase category: "cat"

```
<?xml version="1.0" standalone="no"?>

<!DOCTYPE paula SYSTEM "paula_feat.dtd">
<paula version="1.1">

<header paula_id="mycorpus.doc2_phrase_cat"/>

<featList xmlns:xlink="http://www.w3.org/1999/xlink" type="cat"
xml:base="mycorpus.doc2.phrase.xml">
    <feat xlink:href="#phrase_1" value="NP"/><!-- he -->
    <feat xlink:href="#phrase_2" value="VP"/><!-- takes -->
    <feat xlink:href="#phrase_3" value="NP"/><!-- people _ -->
    <feat xlink:href="#phrase_4" value="PRT"/><!-- out -->
    <feat xlink:href="#phrase_5" value="S"/><!-- _ to fish -->
    <feat xlink:href="#phrase_6" value="NP"/><!-- _ -->
    <feat xlink:href="#phrase_7" value="VP"/><!-- to fish -->
    <feat xlink:href="#phrase_8" value="VP"/><!-- fish -->
    <!-- he takes people out _ to fish -->
    <feat xlink:href="#phrase_9" value="S"/>
    <!-- he takes people out _ to fish -->
    <feat xlink:href="#phrase_10" value="TOP"/>
</featList>

</paula>
```

The annotation name is set as "cat" and it applies to the elements "phrase_1" to "phrase_10" in the xml:base file, which contains the phrase nodes. For conventions how to name the `@paula_id` and XML files, see naming conventions.

Annotating edges works in a similar way, except that `rel` elements are references instead of `struct` elements. It is possible to annotate edges of multiple types in the same XML file, as long as the name of the annotation being applied to them is identical. The following example illustrates this using the edges from Example 7.1, “Constructing a hierarchical syntax tree with `struct` elements type” in the preivous section (note that "rel_7" had the type "secedge" while the others had "edge", and also that not all edges have been annotated, which is fine).

Example 7.3. Annotating edges from a `struct` file with `feat` annotations for phrase function: "func"

```
<?xml version="1.0" standalone="no"?>

<!DOCTYPE paula SYSTEM "paula_feat.dtd">
```

```
<paula version="1.1">

<header paula_id="mycorpus.doc2_phrase_func"/>

<featList xmlns:xlink="http://www.w3.org/1999/xlink" type="func"
xml:base="mycorpus.doc2.phrase.xml">
    <feat xlink:href="#rel_5" value="PRP"/><!-- _ to fish -->
    <feat xlink:href="#rel_9" value="SBJ"/><!-- _ -->
    <feat xlink:href="#rel_11" value="NONE"/><!-- _ -->
    <feat xlink:href="#rel_15" value="SBJ"/><!-- he -->
</featList>

</paula>
```

Just as with markables, it is also possible to specify multiple annotations for the same nodes in one XML document using multiFeat files (see multiFeats for details).

Chapter 8. Pointing relations

Pointing relations are ahierarchical edges between any two annotation node elements, that is between any combination of `tok`, `mark` or `struct`. Unlike hierarchical edges, pointing relations do not express 'dominance' semantics, meaning that the source of the edge is not understood to 'consist of' the target of the edge. The edge merely marks a relationship between two nodes. For this reason, pointing relations are useful in expressing such links as coreference (e.g. a link between anaphor and antecedent) and syntactic dependencies. Pointing relations are represented using `rel` elements in rel files, and obey the definition in `paula_rel.dtd` (see DTDs). The following example illustrates rel edges between tokens defined in the file `mycorpus.doc1.tok.xml`, but the sources and targets of the edges can also be any `struct` or `mark` within a document.

Example 8.1. Pointing relations between token nodes to annotate dependencies of type "dep"

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE paula SYSTEM "paula_rel.dtd">

<paula version="1.1">

<header paula_id="mycorpus.doc1_dep"/>

<relList xmlns:xlink="http://www.w3.org/1999/xlink" type="dep"
xml:base="mycorpus.doc1.tok.xml">
    <!-- I - 've -->
    <rel id="rel_1" xlink:href="#tok_1" target="#tok_2"/>
    <!-- 've - picked -->
    <rel id="rel_2" xlink:href="#tok_3" target="#tok_2"/>
    <!-- the - kids -->
    <rel id="rel_3" xlink:href="#tok_4" target="#tok_5"/>
    <!-- picked - kids -->
    <rel id="rel_4" xlink:href="#tok_5" target="#tok_3"/>
    <!-- picked - up -->
    <rel id="rel_5" xlink:href="#tok_6" target="#tok_3"/>
</relList>

</paula>
```

The `rel` file only defines the edges and the `@type` of the `relList`, in this case "dep". To add an annotation to these edges, for example grammatical functions, a `feat` file is used, as in the following example:

Example 8.2. Annotating the grammatical function "func" for dependency pointing relations

```
<?xml version="1.0" standalone="no"?>

<!DOCTYPE paula SYSTEM "paula_feat.dtd">
<paula version="1.1">

<header paula_id="mycorpus.doc1_dep_func"/>

<featList xmlns:xlink="http://www.w3.org/1999/xlink" type="func"
xml:base="mycorpus.doc1.dep.xml">
    <feat xlink:href="#rel_1" value="SBJ"/><!-- I - 've -->
    <feat xlink:href="#rel_2" value="VC"/><!-- 've picked -->
    <feat xlink:href="#rel_3" value="NMOD"/><!-- the - kids -->
```

```
    <feat xlink:href="#rel_4" value="OBJ"/><!-- picked - kids -->
    <feat xlink:href="#rel_5" value="PRT"/><!-- picked - up -->
</featList>

</paula>
```

Each `feat` element points to a `rel` element in the pointing relation file and gives the annotation value in its `@value` attribute. The name of the annotation, "func", is determined in the @type attribute of the `featList`.

Just as with markables, it is also possible to specify multiple annotations for the same pointing relations in one XML document using multiFeat files (see multiFeats for details).

Chapter 9. Namespaces

Namespaces in PAULA are user-defined strings that may be used to group together XML files belonging to semantically related annotation layers. PAULA namespaces are not XML namespaces, but are signaled through a prefix to the file name which by convention should contain only alphanumeric ASCII characters and should not begin with a number. The end of the prefix is marked by a period.

As an example, consider the following document's directory structure:

Figure 9.1. Directory structure for a PAULA corpus

```
+-- mycorpus/
¦   +-- doc1/
¦   ¦   ¦-- coref.doc1.discourse.xml
¦   ¦   ¦-- coref.doc1.discourse_anaphoric.xml
¦   ¦   ¦-- mycorpus.doc1.anno.xml
¦   ¦   ¦-- mycorpus.doc1.annoFeat.xml
¦   ¦   ¦-- mycorpus.doc1.text.xml
¦   ¦   ¦-- mycorpus.doc1.tok.xml
¦   ¦   ¦-- syntax.mycorpus.doc1.const.xml
¦   ¦   ¦-- syntax.mycorpus.doc1.const_cat.xml
¦   ¦   ¦-- syntax.mycorpus.doc1.const_func.xml
... ... ...
```

The first two file names being with the prefix "coref". This prefix groups them together into one namespace, which contains semantically related annotations, such as some non-terminal "discourse" nodes, and some annotations or edges defined above these nodes, in this case of the type "anaphoric" (for conventional relations between node and annotation file names, see naming conventions). The last three files begin with "syntax" and belong to the corresponding "syntax" namespace. In this case they represent annotations such as those seen in the examples in Chapter 7: nodes of the type "const", an annotation document of the type "cat" and annother annotation called "func", which represents annotated edges between the nodes. Finally, the files in the middle begin with the corpus name "mycorpus", which is therefore also their namespace. They could also be given a separate namespace (e.g. "general.mycorpus...."), but there is no rule prohibiting use of the corpus name as a namespace: this will usually be the case when following the naming conventions if namespaces are not intentionally used (then all annotations have the same namespace: the corpus name).

There is no necessary graph-topological connection between annotation layers in the same namespace. Often, nodes and their annotations are grouped together using a namespace in order to signal their interdependence. However it is entirely possible to group any combination of files under one namespace. At present there is no way of assigning multiple namespaces to a single file: only the string before the first period in a file name is evaluated as its namespace. It is recommended to repeat the namespace in the `@paula_id` attribute of each XML file for consistency, but the filename itself is the deciding factor in determining the namespace.

Chapter 10. Special scenarios

Parallel corpora

Parallel corpora can be modelled in PAULA XML in a variety of ways that are more or less appropriate. For instance, an implicit parallel alignment can be achieved by treating an aligned text as an annotation of a source text (each word or group of words is annotated with parallel words). However, the explicit and recommended representation of parallel corpora in PAULA is modelled by defining multiple `primary text data` files within a `document` directory, each with at least one `tokenization`. In this way, each text is explicitly made independent from the others and text level alignment is represented by the shared document folder. It is recommended to give each text and tokenization a separate, meaningful namespace, such as the name of the language if dealing with a multilingual parallel corpus. Alignment between elements within parallel texts, including aligned tokens, markable spans (e.g. sentences or chunks) or hierarchical structures, is achieved using pointing relations. The following example illustrates the document structure and an alignment for some tokens.

Figure 10.1. Directory structure for a document with two parallel texts.

```
+-- mycorpus/
¦   +-- doc1/
¦   ¦   ¦-- english.doc1.text.xml
¦   ¦   ¦-- english.doc1.tok.xml
¦   ¦   ¦-- german.doc1.text.xml
¦   ¦   ¦-- german.doc1.tok.xml
¦   ¦   ¦-- mycorpus.doc1.align.xml
¦   ¦   ¦-- mycorpus.doc1.anno.xml
... ... ...
```

Figure 10.2. Pointing relations aligning the English text to the German text.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE paula SYSTEM "paula_rel.dtd">


<paula version="1.1">

<header paula_id="mycorpus.doc1_align"/>

<relList xmlns:xlink="http://www.w3.org/1999/xlink" type="align">
    <rel id="rel_1" xlink:href="english.doc1.tok.xml#tok_1"
    target="german.doc.tok.xml#tok_1"/>
    <rel id="rel_1" xlink:href="english.doc1.tok.xml#tok_2"
    target="german.doc.tok.xml#tok_3"/>
    <rel id="rel_1" xlink:href="english.doc1.tok.xml#tok_3"
    target="german.doc.tok.xml#tok_2"/>
</paula>
```

Note that since pointing relations of the same type may not create a cycle, bidirectional alignment is only possible if the pointing relation files are given different types, as in the following example. The two alignment files use the types "align_g-e" and "align_e-g" for each alignment direction.

Figure 10.3. Directory structure for a document with bidirectional alignment.

```
+-- mycorpus/
¦   +-- doc1/
¦   ¦   ¦-- english.doc1.align_e-g.xml
¦   ¦   ¦-- english.doc1.text.xml
¦   ¦   ¦-- english.doc1.tok.xml
¦   ¦   ¦-- german.doc1.align_g-e.xml
¦   ¦   ¦-- german.doc1.text.xml
¦   ¦   ¦-- german.doc1.tok.xml
¦   ¦   ¦-- mycorpus.doc1.anno.xml
... ... ...
```

Dialogue data

There are two main ways of representing dialog data in PAULA XML: either each speaker's text and annotations are modeled as a text in a parallel corpus (see parallel corpora) or else a `primary textual data` file is created with as many blank characters as necessary for the representation of all speakers, and this is then used as a common timeline for the tokens of each speaker. The latter solution is implemented as follows. Supposing two speakers utter the following two semi overlapping sentence:

Figure 10.4. Dialog data to be modelled in PAULA.

```
Speaker1:   he thinks so
Speaker2:              I think so too
```

Speaker2 utters the word "I" at the same time as the "o" is uttered in "so" by Speaker1. In order to model this overlap using only one "text", the `primary textual data` must contain a sufficient amount of characters. The text for Speaker1 is 12 characters long, including spaces, and the text for Speaker2 begins at character 12 of Speaker1 and extends for a further 14 characters. This means we require 25 characters in total (not 26, since there is an overlap of one character). The raw text file can therefore look like this:

Example 10.1. A primary text data file

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE paula SYSTEM "paula_text.dtd">

<paula version="1.1">
<header paula_id="mycorpus.doc4_text" type="text"/>

<body>1234567890123456789012345</body>

</paula>
```

The body of the text contains repeating numbers: 1234567890... to make it easier to count the characters. However it is equally possible to use 25 spaces: the contents of this dummy text file are not important. In a second step, two tokenizations of the data are carried out: one for each speaker. The tokenization for Speaker1 is given in the following example. It is recommended to give each speaker a separate namespace for easier identifiability.

Example 10.2. Tokenization for Speaker1

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE paula SYSTEM "paula_mark.dtd">
<paula version="1.1">

<header paula_id="mycorpus.doc4_tok"/>

<markList xmlns:xlink="http://www.w3.org/1999/xlink" type="tok"
xml:base="mycorpus.doc4.text.xml">
 <!-- he -->
 <mark id="tok_1" xlink:href="#xpointer(string-range(//body,'',1,2))"/>
 <!-- thinks -->
 <mark id="tok_2" xlink:href="#xpointer(string-range(//body,'',4,6))"/>
 <!-- so -->
 <mark id="tok_3" xlink:href="#xpointer(string-range(//body,'',11,2))"/>
</markList>

</paula>
```

Annotations for each speaker can then be added by refering to the relevant token file and building hierarchical structures above the tokens.

Aligned audio/video files

Aligned multimedia files, such as audio or video files, can be added to a PAULA document by placing them in the relevant document directory. In order to specify which part of a text is represented in the aligned file or files, a `mark` element covering the appropriate span of tokens should be defined and annotated using a `feat` which contains the file name as in the example below. It is possible to annotate the same `mark` element with multiple multimedia files.

Figure 10.5. *A `mark` file defining the span of tokens aligned with a multimedia file.*

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE paula SYSTEM "paula_mark.dtd">

<paula version="1.1">
<header paula_id="mycorpus.doc1_audioFileSeg"/>

<markList xmlns:xlink="http://www.w3.org/1999/xlink"
type="audioFileSeg" xml:base="mycorpus.doc1.tok.xml">
 <!-- audio file span for the first 50 tokens -->
 <mark id="audioFileSeg_1"
  xlink:href="#xpointer(id('tok_1')/range-to(id('tok_50')))"/>
</markList>

</paula>
```

Figure 10.6. A `feat` file giving the name of the multimedia file.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE paula SYSTEM "paula_feat.dtd">

<paula version="1.1">
<header paula_id=""mycorpus.doc1_audioFileSeg_audioFile"/>

<featList xmlns:xlink="http://www.w3.org/1999/xlink"
type="audioFile" xml:base="mycorpus.doc1.audioFileSeg.xml">
  <!-- wav file -->
  <feat xlink:href="#audioFileSeg_1" value="file:/./mycorpus.doc1.wav"/>
</featList>

</paula>
```

Chapter 11. Naming conventions

General conventions

- File names in a directory other than the DTDs should ideally contain their corpus path, or at least the document name, i.e. the name of the folder they are in. This ensures that files carry unique names that make them easier to identify. For example, the tokenization file of the document `doc01/` in the corpus `mycorpus` might be called `mycorpus.doc01.tok.xml` or `doc01.tok.xml`.
- Do not use file or folder names with spaces or non-ascii characters.
- Do not use file or folder names that begin with a number or underscore.
- When using namespaces, remember that the string before the first period in the file name is construed as the namespace. If you do not wish to use namespaces and follow the file naming conventions given here, the namespace for all of your files will be the corpus name, since files will always be named: `mycorpus.*`.

annoSet, annoFeat, primary text data and tokenization

- The `annoSet` and, if used, `annoFeat` files in a document are conventionally named using the document path convention above, with the suffixes anno.xml and anno_feat.xml respectively. For example they can be called: `mycorpus.doc01.anno.xml` and `mycorpus.doc01.anno_feat.xml`.
- If there is only one `primary text data` file and one `tokenization`, they are usually named similarly, but with the suffixes text and tok: `mycorpus.doc01.text.xml` and `mycorpus.doc01.tok.xml`.
- If there are multiple primary text data files or tokenization, their distinguishing features may be used as namespaces, e.g. the name of the language in a parallel corpus documents: `english.mycorpus.doc01.text.xml` and `english.mycorpus.doc01.tok.xml`. If the namespaces are already being used for some other purpose (e.g. names of speakers when using a parallel corpus architecture for dialogue data), suffixes distinguishing text and token files may be used before "text" and "tok", as in: `speaker1.mycorpus.doc01.english.text.xml` and `speaker1.mycorpus.doc01.german.text.xml`, and the same for `*.tok.xml` files.

Anntotation span markables and feature annotations

- By convention, annotation span markable files are named using the current document name as a prefix, followed by an underscore and the markList's type, followed by "_seg.xml". For example, a markable file that marks text segments corresponding to discourse referents for further annotation may be named `mycorpus.doc01.referent_seg.xml`. This tells us just by looking at the file name that the markable `@type` attribute in the `markList` element is "referent".
- The above file may may also be put in a namespace with some other files relevant to discourse annotation, in which case the files receive a common prefix, e.g. the file could be named: `discourse.mycorpus.doc01.referent_seg.xml`.
- A feature annotation of the above file giving the referent segment e.g. an annotation called "type" (marking the referent, say, as a person or geopolitical entity), will be given a file name identical to that of the `_seg` file, but with the annotation name appended after a further underscore: `discourse.mycorpus.doc01.referent_seg_type.xml`.

Hierarchical struct nodes and feature annotations

- Hierarchical `struct` nodes are placed in files using the same general conventions with regard to namespaces and corpus/document path above, and carry a suffix corresponding to the `@type` attribute in the `structList` element after an underscore, as follows. For nodes annotating

syntactic constituents of the type "const" within the namespace "syntax" we may get a file called: `syntax.mycorpus.doc01.const.xml`.

- Annotations of struct nodes are given the same name as the corresponding node file, with a suffix consisting of an underscore and the annotation's name from the `@type` attribute of the `featList` element. For example, an annotation of the above constituent nodes giving the syntactic category called "cat" should be named: `syntax.mycorpus.doc01.const_cat.xml`.

- Feature annotations of edges in the same `struct` file should be named using the same convention, e.g. a syntactic function annotation of the type "func" may be called: `syntax.mycorpus.doc01.const_func.xml`.

Pointing relations and rel annotations

- Pointing relation files are named using the same conventions as above, with the edge type used as a suffix after the document name, e.g. a coreference edge file of the type "coref" in the discourse namespace should be named: `discourse.mycorpus.doc01.coref.xml`.

- Feature annotations of pointing relation edges are given the file name of the pointing relation file with an underscore and the annotation type as a suffix. For example, annotating the "coref" edge above with the annotation "type" (e.g. anaphoric or appositional) results in the file name: `discourse.mycorpus.doc01.coref_type.xml`.

multiFeat annotations

- A `mutliFeat` file has no single annotation type. It is therefore usually named using the name of the file to which it adds annotations, with the suffix "_multiFeat". Therefore the name of a `multiFeat` file annotation a token file is e.g. `mycorpus.doc1.tok_multiFeat.xml`, a `multiFeat` file annotating syntactic constituents called "const" might be called `mycorpus.doc1.const_multiFeat.xml`, etc.

- For metadata multiFeat annotations, usually the document path and the suffix "meta_multiFeat" are used, e.g. `mycorpus.doc1.meta_multiFeat.xml`.

The paula_id attribute

- The `@paula_id` attribute of the `header` element in each filed should be named like the file name itself without the .xml extension, e.g. the paula_id of `mycorpus.doc01.tok.xml` might be `mycorpus.doc01.tok`.

- If the resulting name has no suffix containing an underscore, it is possible to replace the final period in the file name with an underscore, e.g. `mycorpus.doc01_tok`.

Chapter 12. Older versions and deprecated components

Pointing relations in feats

Up to PAULA XML version 1.0 it was possible to create pointing relations by assigning a feature annotation to a source node with the target node's URI as a feature value (in PAULA 0.9 only) or using the now deprecated `@target` attribute of the feat element (from PAULA 1.0). The use of `@value` for this purpose is illustrated in the example below.

Example 12.1. A deprecated pointing relation `Feat` file.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE paula SYSTEM "paula_feat.dtd">

<paula version="1.1">
 <header paula_id="mycorpus.doc1_coref" />
 <featList type="coref" xml:base="mycorpus.doc1.referent_Seg.xml"
 xmlns:xlink="http://www.w3.org/1999/xlink">
  <feat xlink:href="#referent_10" value="#referent_8" />
 </featList>

</paula>
```

The problem with this structure is that it is not unambiguously clear that the annotation signifies a pointing relation, rather than a label annotation that happens to resemble a URI (an annotation with the string value "#referent_8" in the example above). As of PAULA 1.1, `rel` files with their corresponding DTD should be used to define pointing relations, making the identification of source and target nodes unambiguous. It is still possible (though deprecated) to use a `feat` file for this purpose, as long as the pointing relation's target is marked using `@target` instead of `@value`. However, this will not be supported in future versions of the PAULA standard.

Virtual markables

In PAULA XML version 0.9 it was possible to define "virtual markables" which could span several markables, either in the same markable file or in any number of different markable files applying to the same tokenization. The following example illustrates such a file, where the virtual markable, designated by the `@type` "virtual", refers to two markables within the same file (the path `mycorpus.doc5.referentSeg.xml` must be specified since `@xml:base` is set to a separate tokenization file).

Figure 12.1. ***A `mark` file containing a pseudo-hierarchical markable of the deprecated "virtual" type.***

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE paula SYSTEM "paula_mark.dtd">

<paula version="1.1">
<header paula_id="mycorpus.doc5_referentSeg"/>

<markList xmlns:xlink="http://www.w3.org/1999/xlink"
type="referent" xml:base="mycorpus.doc5.tok.xml">
 <!-- audio file span for the first 50 tokens -->
 <mark id="referentSeg_1"
  xlink:href="#xpointer(id('tok_1')/range-to(id('tok_2')))"/>
 <mark id="referentSeg_2"
  xlink:href="#xpointer(id('tok_5')/range-to(id('tok_8')))"/>
 <mark id="referentVirt_1"
  xlink:href="(mycorpus.doc5.referentSeg.xml#referentSeg_1,
  mycorpus.doc5.referentSeg.xml#referentSeg_2)" type="virtual"/>
</markList>

</paula>
```

Though virtual markables technically appear to be hierarchical structures by pointing at constituent markables, they are interpreted as flat spans which apply to exactly the same tokens as those covered by the constituent markables. Therefore the virtual markable in the example above is the same as a markable applying to tokens 1-2 and 5-8. The use of virtual markables has been deprecated and is no longer part of the current PAULA XML standard. Note that it is possible to create discontinuous spans using normal markables, by specifying discontinous ranges of tokens in the `@xlink:href` attribute.

Synopsis of older PAULA versions and components

This section lists distinctive characteristics of the different PAULA XML standard versions to date.

Version 0.9

- Use of virtual markables is possible.
- Use of `feat` attribute `@value` to specify pointing relation target nodes is possible/
- No support for metadata.
- Use of `annoFeat` is mandatory.

Version 1.0

- Use of virtual markables is no longer possible.
- Use of `feat` attribute `@value` or `@target` to specify pointing relation target nodes is possible.
- No support for metadata.
- Use of `annoFeat` is mandatory.

Version 1.1

- Use of virtual markables is not possible.
- Use of `feat` attribute `@value` to specify pointing relation target nodes is not possible.
- Use of `feat` attribute `@target` to specify pointing relation target nodes is possible but deprecated.
- New file type and element `rel` is recommended for the specification of pointing relations.
- Support for metadata on the corpus, subcorpus and document levels.
- Use of `annoFeat` is optional and deprecated.
- Support for parallel corpora via pointing relations.
- Support for aligned multimedia files.

References

Chiarcos, C., Dipper, S., Götze, M., Leser, U., Lüdeling, A., Ritz, J. & Stede, M. (2008), A Flexible Framework for Integrating Annotations from Different Tools and Tag Sets. *Traitment automatique des langues* 49, 271-293.

Dipper, S. (2005), XML-based Stand-off Representation and Exploitation of Multi-Level Linguistic Annotation. In: *Proceedings of Berliner XML Tage 2005 (BXML 2005)*. Berlin, Germany, 39-50.

Dipper, S. & Götze, M. (2005), Accessing Heterogeneous Linguistic Data - Generic XML-based Representation and Flexible Visualization . In: *Proceedings of the 2nd Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics*. Poznan, Poland, 206-210.