Contents

1 Introduction

FoLiA is a Format for Linguistic Annotation, derived from the DCOI[**?**] format developed at Polderland, and extended for richer annotation at the ILK research group, Tilburg University. The DCOI format is designed for used by the DCOI corpus, as well as by the current incarnation of its successor; the SoNaR corpus.

FoLiA is an XML-based annotation format, suitable for representing language resources such as corpora. Its goal is to unify a variety of linguistic annotations in one single rich format, without commiting to any particular standard annotation set. Instead, it seeks to accomodate any desired system or tagset, and so offering maximum flexibility. This makes FoLiA language independent.

XML is an inherently hierarchic format, FoLiA does justice to this by maximally utilising a hierarchic, inline, setup. We inherit from the DCOI format, which is loosely based on a minimal subset of TEI. However, FoLiA is *not* backwards-compatible with DCOI. FoLiA is a very rich format, and it is always fairly easy to convert back to "less verbose" formats such as the DCOI format, or plain-text.

However, some linguistic annotations such as syntactic parses are implemented as "layers" seperate from the tokenisation, and using a kind of offset notation. This is to provide FoLiA with the necessary flexibility and extensability.

The FoLiA format features the following:

- Open-source
- XML-based, validation against XML schema.
- Full Unicode support; UTF-8 encoded.
- Document structure consists of divisions, paragraphs, sentences and words/tokens.
- Can encode both tokenised as well as untokenised text + partial reconstructability of untokenised form even after tokenisation.
- Support for crude token categories (word, punctuation, number, etc)

- Explicit support for encoding quotations
- Provenance support for all linguistic annotations: annotator, type (automatic or manual), time.
- Support for alternative annotations, optionally with associated confidence values.
- Adaptable to different tag-sets.
- CMDI Metadata or IMDI metadata

It supports the following linguistic annotations:

- Spelling corrections on both a tokenised as well as an untokenised level.
- Semantic sense annotation (to be used in DutchSemCor)
- Part-of-Speech tags (with features)
- Lemmatisation
- Morphological Analysis
- Multi-word units and Named Entities
- Syntactic Parses
- Chunking / Shallow Parsing
- Semantic Role Labelling (?)

FoLiA support will be incorporated directly into the following ILK sofware:

- ucto - A tokeniser which can directly output FoLiA XML
- Frog - A PoS-tagger/lemmatiser/parser suite (the successor of Tadpole), will eventually support reading and writing FoLIA.
- CLAM - Computational Linguistics Application Mediator, will eventually have viewers for the FoLiA format.

- PyNLPl - Python Natural Language Processing Library, will come with libraries for parsing FoLiA
- libfolia - C++ library for parsing FoLiA
- TiCCL

And it may be used in the following corpora:

- SoNaR (yet to be confirmed)
- DutchSemCor (based primarily on SoNaR)

Development Notes
This is all still subject to debate and change and may be a bit pretentious at this stage.

2 Basic Structure

In FoLiA, each document/text is represented by one XML file. The basic structure of such a FoLiA document is as follows and should always be UTF-8 encoded. An eleborate XSLT stylesheet will be provided in order to be able to instantly view FoLiA documents in any modern web browser.

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="http://ilk.uvt.nl/FoLiA/FoLiA.xsl"?>
<FoLiA xmlns="http://ilk.uvt.nl/FoLiA" xmlns:xsi="http://www.w3.org/2001/XMLSche
  <!-- (Here IMDI or CMDI metadata can be inserted) -->
  <annotations>
      ...
  </annotations>
  <text xml:id="example.text">
   <gap></gap>
   <body>
     ...
   </body>
   <gap></gap>
  </text>
</FoLiA>
```

Body contains the to-be-annotated material, more about this later. The gap tags are optional and contain unannotated front matter or back matter.

3 Identifiers

All elements which describe tokens or a span of tokens have an identifier by which it is uniquely identifiable. This makes refering to any part of a FoLiA document easy and follows the lead of the DCOI format. The identifiers

are constructed in the same way as in the DCOI format, thus retaining full compatibility; if a DCOI document is converted to FoLiA, all external references to any entity in these documents will remain intact.

Identifiers in DCOI and FoLiA are cummulative and are indicative of the position of the element within the document; a child element usually takes the ID of its parent, appends its name, a period, and a sequential number. This however only applies to certain primary elements: paragraph elements, sentence elements, and word elements.

The base of all identifiers is that of the document itself, as encoded in `xml:id` attribute of the root `FoLiA` element. This is a unique ID by which the document is identifiable. We choose the identifier *example* in the example above. By convention, the XML file should then ideally be named: `example.xml`.

4 Structure Elements

Within the document body, the structure elements div0, div1, div2, etc.. can be used to create divisions and subdivisions, they stem from DCOI and are unmodified in FoLiA. These divisions are not mandatory, but may be used to distinguish extra structure.

5 Content Elements

Content element occur within the body, the following exist:

- `head` - Header
- `p` - Paragraph
- `s` - Sentence
- `w` - Word (token)
- `quote` - Quote

These are typically nested, the word elements describe the actual tokens, including tokens thare are not technically a word. Let's take a look at an example, we have the following text:

```
This is a paragraph containing only one sentence.

This is the second paragraph. This one has two sentences.
```

In FoLiA XML, this will appear as follows after tokenisation:

```
<p xml:id="TEST.p.1">
   <s xml:id="TEST.p.1.s.1">
       <w xml:id="TEST.p.1.s.1.w.1"><t>This</t></w>
       <w xml:id="TEST.p.1.s.1.w.2"><t>is</t></w>
       ...
       <w xml:id="TEST.p.1.s.1.w.8" space="no"><t>sentence</t></w>
       <w xml:id="TEST.p.1.s.1.w.9"><t>.</t></w>
   </s>
</p>
<p xml:id="TEST.p.2">
   <s xml:id="TEST.p.2.s.1">
       <w xml:id="TEST.p.2.s.1.w.1"><t>This</t></w>
       <w xml:id="TEST.p.2.s.1.w.2"><t>is</t></w>
       ..
       <w xml:id="TEST.p.2.s.1.w.5" space="no"><t>paragraph</t></w>
       <w xml:id="TEST.p.2.s.1.w.6"><t>.</t></w>
   </s>
   <s xml:id="TEST.p.2.s.2">
       <w xml:id="TEST.p.2.s.2.w.1"><t>This</t></w>
       <w xml:id="TEST.p.2.s.2.w.2"><t>one</t></w>
       ..
       <w xml:id="TEST.p.2.s.2.w.5" space="no"><t>sentences</t></w>
       <w xml:id="TEST.p.2.s.2.w.6"><t>.</t></w>
   </s>
</p>
```

The deepest content element should always contain a text element (`t`) which holds the actual textual content. FoLiA may also hold untokenised text, on a paragraph and/or sentence level:

```
<p xml:id="TEST.p.1">
   <s xml:id="TEST.p.1.s.1">
       <t>This is a paragraph containing only one sentence.</t>
   </s>
</p>
<p xml:id="TEST.p.2">
   <s xml:id="TEST.p.2.s.1">
       <t>This is the second paragraph.</t>
   </s>
   <s xml:id="TEST.p.2.s.2">
       <t>This one has two sentences.</t>
   </s>
</p>
```

Higher level elements *may* also contain a text element even when the deeper element do too. But the sentence/paragraph-level text element should always contain the text *prior* to tokenisation! Note also that the word element has an attribute `space`, which defaults to yes, and indicates whether a the word was followed space by a space in the *untokenised* original. This allows for partial reconstructability of sentence in its untokenised form. Reconstructing sentences is generally preferred to grabbing them from the text element at the paragraph or sentence level, as there may be corrections or other changes on the token level.

The following example shows the maximum amount of redundancy.

```
<p xml:id="TEST.p.1">
   <t>This is a paragraph containing only one sentence.</t>
   <s xml:id="TEST.p.1.s.1">
       <t>This is a paragraph containing only one sentence.</t>
       <w xml:id="TEST.p.1.s.1.w.1"><t>This</t></w>
       <w xml:id="TEST.p.1.s.1.w.2"><t>is</t></w>
       ...
       <w xml:id="TEST.p.1.s.1.w.8" space="no"><t>sentence</t></w>
       <w xml:id="TEST.p.1.s.1.w.9"><t>.</t></w>
   </s>
</p>
<p xml:id="TEST.p.2">
   <t>This is the second paragraph. This one has two sentences.</t>
   <s xml:id="TEST.p.2.s.1">
```

```
        <t>This is the second paragraph.</t>
        <w xml:id="TEST.p.2.s.1.w.1"><t>This</t></w>
        <w xml:id="TEST.p.2.s.1.w.2"><t>is</t></w>
        ..
        <w xml:id="TEST.p.2.s.1.w.5" space="no"><t>paragraph</t></w>
        <w xml:id="TEST.p.2.s.1.w.6"><t>.</t></w>
    </s>
    <s xml:id="TEST.p.2.s.2">
        <t>This one has two sentences.</t>
        <w xml:id="TEST.p.2.s.2.w.1"><t>This</t></w>
        <w xml:id="TEST.p.2.s.2.w.2"><t>one</t></w>
        ..
        <w xml:id="TEST.p.2.s.2.w.5" space="no"><t>sentences</t></w>
        <w xml:id="TEST.p.2.s.2.w.6"><t>.</t></w>
    </s>
</p>
```

Also note that the paragraph elements may be omitted if a document is described that does not distinguish paragraphs but only sentences. The IDs change accordingly then. Sentences however should never be omitted, documents can not consist of only tokens!

The content element `head` is reserved for headers and captions, it behaves similarly to the paragraph element and may hold sentences.

6 Paradigm & Terminology

The FoLiA format has a very uniform setup and its XML notation for annotation follows a generalised paradigm.

First of all, we distinguish two different categories of annotation:

- Token annotation - Annotations pertaining to one specific token. These will be elements of the token element. (inline). Linguistic annotations in this category are for example: part-of-speech annotation, lemma annotation, sense annotation, morphological analysis, spelling correction.
- Span annotation - Annotations spanning over multiple tokens. Each type of annotation will be in a seperate annotation layer with offset

notation. These layers are embedded on the sentence level. Examples of this category are: syntax parsers, chunking, semantic role labelling, named entity annotation, temporal analysis.

Most linguistic annotations are associated with what we call a **set**, the set determines the vocabulary (the tags) of the annotation. An element of such a set is referred to as a **class** from the FoLiA perspective. For example, we may have a document with Part-of-Speech annotation according to the CGN set. The CGN set defines classes such as *WW*, *BW*, *ADJ*, *VZ*. FoLiA itself thus never commits to any tagset but leaves you to explicitly define this.

Any annotation element may have a `set` attribute (pointing to a URL of the file that defines the set), and `class` element, that choses a class from the set.

The following example shows a simple Part-of-Speech annotation (without features), but with all annotation attributes according to the FoLiA paradigm:

```
<pos set="http://ilk.uvt.nl/folia/sets/CGN" class="WW"
 annotator="Maarten van Gompel" annotatortype="manual" confidence="0.76" />
```

The example shows that any annotation element can be enriched with an `annotator` attribute and an `annotatortype`. The latter is either "manual", for human annotators, or "auto" for systems. The value for `annotator` is open and should be set to the name or ID of the system or human annotator that made the annotation. Lastly, there is a `confidence` attribute, set to a floating point value between zero and one, that expresses the confidence the annotator places in his annotation. None of these options is mandatory, except for `class`, this may be mandatory for some types of annotation (such as `pos`).

Development Notes
In this stage, the sets are not actually defined yet, i.e. the URLs they point to don't exist yet. But the idea is that a set always points to a URL that defines all its classes. The format for this is still to be specified however. For now, ad-hoc sets that will later be defined will do.

7 Annotation Declarations

Explicitly referring to a set and annotator in each annotation can be cumbersome, especially in a document with a single set and a single annotator for that type of annotation. This problem can be solved by declaring defaults in the annotation declaration.

The annotation declaration is a mandatory part of the metadata that declares all kinds of annotations types that are present in the document, in addition it may define the tagset used, and may declare a default annotator for this type of annotation. These defaults can always be overriden at the annotation level itself, using the XML attributes `set`, `annotator` and `annotatortype`, as discussed in the previous section. None of the attributes are mandatory in the declaration, though the declarations themselves are; they declare what annotations are to be expected in the document. Having a type of annotation that is not declared is invalid. Do note that if you do not specify a set, annotator or annotatortype in either the declaration or in the annotation elements themselves, they will be left undefined. Not declaring sets is generally a very bad idea.

Annotations are declared in the `annotations` block, as shown in the following example:

```
<annotations>
        <token-annotation set="http://ilk.uvt.nl/folia/sets/ucto-tokconfig-nl" a
        <pos-annotation set="http://ilk.uvt.nl/folia/sets/CGN" annotator="Frog"
        <lemma-annotation annotator="Frog" annotatortype="auto" />
        <sense-annotation set="http://ilk.uvt.nl/folia/sets/Cornetto" annotator=
</annotations>
```

8 Token Annotation

Token annotations are annotations that are placed within the word (`w`) element. They all can take any of the attributes described in section 6, this has to be kept in mind when reading this section. Moreover, all token annotations depend on the document being tokenised, i.e. there being a `token-annotation` declaration and `w` elements. The declaration can be as follows:

```
<annotations>
    <token-annotation set="http://ilk.uvt.nl/folia/sets/ucto-tokconfig-nl" annot
</annotations>
```

Being part of a set, this implies that tokens themselves *may* be assigned a class, as is for example done by the tokeniser *ucto*:

```
<s xml:id="example.p.1.s.1">
    <w xml:id="example.p.1.s.1.w.1" class="WORD"><t>I</t></w>
    <w xml:id="example.p.1.s.1.w.2" class="WORD"><t>see</t></w>
    <w xml:id="example.p.1.s.1.w.3" class="NUMBER"><t>2</t></w>
    <w xml:id="example.p.1.s.1.w.4" class="WORD" space="no"><t>children</t></w>
    <w xml:id="example.p.1.s.1.w.5" class="PUNCTUATION"><t>.</t></w>
</s>
```

8.1 Part of Speech Annotation

The following example illustrates a simple Part-of-Speech annotation for the Dutch word “boot”:

```
<w xml:id="example.p.1.s.1.w.2">
    <t>boot</t>
    <pos class="N" />
</w>
```

Part-of-Speech annotations may also include extra features, which are explicitly listed and are defined by the set:

```
<w xml:id="example.p.1.s.1.w.2">
    <t>boot</t>
    <pos class="N">
        <feat class="ntype" value="soort" />
        <feat class="number" value="ev" />
        <feat class="degree" value="basis" />
        <feat class="gender" value="zijd" />
        <feat class="case" value="stan" />
    </pos>
</w>
```

Whenever Part-of-Speech annotations are used, they should be declared in the `annotations` block as follows, the set you use may differ and all attributes are optional. In the declaration example here we do as if the annotations were made by the software *Frog*. Do note the requirement of a `token-annotation` as well.

```
<annotations>
    <token-annotation set="http://ilk.uvt.nl/folia/sets/ucto-tokconfig-nl" annot
    <pos-annotation set="http://ilk.uvt.nl/folia/sets/CGN" annotator="Frog" anno
</annotations>
```

As mentioned earlier, the declaration only sets defaults. They can be overridden in the `pos` element itself (or any other token annotation element for that matter).

8.2 Lemma Annotation

In the FoLiA paradigm, lemmas are perceived as classes within the (possibly open) set of all possible lemmas. Their annotation is thus as follows:

```
<w xml:id="example.p.1.s.1.w.2">
    <t>boot</t>
    <lemma class="boot" />
</w>
```

And the example declaration:

```
<annotations>
    <token-annotation set="http://ilk.uvt.nl/folia/sets/ucto-tokconfig-nl" annot
    <lemma-annotation set="http://ilk.uvt.nl/folia/sets/mblem-nl" annotator="Fro
</annotations>
```

8.3 Semantic Sense Annotation

In semantic sense annotation, the classes in most sets will be a kind of lexical unit ID. In system that make a distinction between lexical units and synsets, the synset attribute is available for description of the latter. In systems with only synsets, the class can be the synset.

The actual value of the *sense* element can be set to a human-readable description, but this is optional.

```
<w xml:id="example.p.1.s.1.w.2">
    <t>boot</t>
    <sense class="r_n-6220" synset="d_n-32683">beeldhouwwerk</sense>
</w>
```

The example declaration is as follows:

```
<annotations>
    <token-annotation set="http://ilk.uvt.nl/folia/sets/ucto-tokconfig-nl" annot
    <sense-annotation set="http://ilk.uvt.nl/folia/sets/cornetto" />
</annotations>
```

8.4 Domain Tags

This is a bit of a peculiar token annotation element, in the sense that it is more than just that. It can also be used directly in any of the content elements, such as sentence (`s`) and paragraph (`p`). It can even be used in the `body` element itself. This annotation defines the domain of the token/content. Example:

```
<w xml:id="example.p.1.s.1.w.2">
```

```
    <t>boot</t>
    <domain class="naut">Nautical</domain>
</w>
```

The value of the element may optionally be set to a human-readable label for the domain.

The declaration:

```
<annotations>
    <token-annotation set="http://ilk.uvt.nl/folia/sets/ucto-tokconfig-nl" annot
    <domain-annotation set="http://ilk.uvt.nl/folia/sets/domains-nl" />
</annotations>
```

8.5 Corrections

8.6 Morphological Analysis

Development Notes
Still to be done..

9 Alternative Token Annotations

The FoLiA format does not just allow for a single "favoured" annotation per token, in addition it allows for the recording of alternative annotations. Alternative token annotations are grouped within one or more `alt` elements. If multiple annotations are grouped together under the same `alt` element, then they are deemed dependent and form an alternative as a group.

Each alternative has an ID. In the following example we see the Dutch word "bank" in the sense of a sofa, alternatively we see two alternative annotations with a different sense and domain.

```
<w xml:id="example.p.1.s.1.w.1">
    <t>bank</t>
    <domain class="furniture" />
    <sense class="r_n-5918" synset="d_n-21410" annotator="John Doe" annotatortyp
    <alt xml:id="example.p.1.s.1.w.1.ALT.1">
        <domain class="finance" />
        <sense class="r_n-5919" synset="d_n-27025" annotator="Jane Doe" annotato
    </alt>
    <alt xml:id="example.p.1.s.1.w.1.ALT.2">
        <domain class="geology" />
        <sense class="r_n-5920" synset="d_n-38257" annotator="Jim Doe" annotator
    </alt>
</w>
```

10 Span Annotation

10.1 Entities

10.2 Syntax

10.3 Chunking

11 Parsing