FoLiA: Format for Linguistic Annotation - Documentation

Release v2.0 (rev 9.0)

Maarten van Gompel

Aug 19, 2021

Contents

version: 2.5.0

Abstract

FoLiA, an acronym for **Format for Linguistic Annotation**, is a data model and file format to represent digitised language resources enriched with linguistic annotation, e.g. linguistically enriched textual documents or transcriptions of speech. The format is intended to provide a standard for the storage and exchange of such language resources, including corpora and to promote interoperability amongst Natural Language Processing tools that use the format.

CHAPTER 1

Introduction

version: 2.5.0

FoLiA, an acronym for **Format for Linguistic Annotation**, is a data model and file format to represent digitised language resources enriched with linguistic annotation, e.g. linguistically enriched textual documents or transcriptions of speech. The format is intended to provide a standard for the storage and exchange of such language resources, including corpora and promote interoperability amongst Natural Language Processing tools that use the format.

Our aim is to introduce a single rich format that can accommodate a wide variety of linguistic annotation types through a single generalised paradigm. We do not commit to any label/vocabulary set, language or linguistic theory. This is always left to the developer of the language resources, and provides maximum flexibility. We merely specify the broad category of annotation types and provide other mechanisms that allow resource constructors to formalize vocabulary sets.

FoLiA has the following characteristics:

- **Expressive**: The format is highly expressive, annotations can be expressed in great detail and with flexibility to the user's needs, without forcing unwanted details. Moreover, FoLiA has generalised support for representing annotation alternatives, and elaborate provenance data, i.e. data on who or what performed certain annotations and through which NLP pipelines the document has gone.
- **Generic** - We apply the same paradigm to a wide variety of annotation types, assuring a uniform and consistent way of representing different annotation.
- **Specific** - FoLiA very explicitly defines various annotation types. This means it choses *not* to subscribe to a looser paradigm where users themselves can invent their annotation types, but keeps this centralised to promote compliance to a more rigid structure. This ensures that any FoLiA-capable tools know what to expect.
- **Formalised** - The format is formalised, and can be validated on both a shallow and a deep level (the latter including tagset validation), and easily machine parsable, for which tools are provided.
- **Practical** - FoLiA has been developed in a bottom-up fashion right alongside applications, programming libraries, and other toolkits and converters. Whilst the format is rich, we try to maintain it as simple and straightforward as possible, minimising the learning curve and making it easy to adopt FoLiA in practical applications.

The FoLiA format is XML-based and makes mixed use of inline and stand-off annotation. XML is an inherently hierarchic format and FoLiA does justice to this by utilising a hierarchic, inline, setup where possible. Inline annotation is used for annotations pertaining to singular structural elements such as words/tokens, whilst stand-off annotation in separate annotation layers is adopted for annotation types that span over multiple structural elements.

FoLiA favours a **single-document** approach, meaning that a text and all linguistic annotations on that text are stored in a single XML file. This facilitates keeping all annotation layers in sync with eachother and prevents incomplete or loss of information. The single-document approach is not just limited to the annotation and text, but also encompasses the document structure and document mark-up (e.g. basic text styling). Nevertheless, there exists a FoLiA mechanism that does allow you to take a more stand-off approach and store annotations in separate external FoLiA documents if absolutely needed.

This documentation is limited to describing, in great detail, how FoLiA works, for more about the motivation behind the construction of FoLiA and how it compares to somewhat similar or comparable initiatives such as TEI [Burnard2007] , LAF [Ide2004] , TCF [Heid2010], NAF [Fokkens2014], Paula XML, Tiger XML, and others, we refer you to our research paper providing a descriptive and comparative study: [vanGompel2014] .

1.1 Annotation Types

FoLiA defines various XML elements to represent document structure and various annotations, we can divide these XML elements into several generic annotation groups. In each of these categories, FoLiA defines specific elements for specific annotation types. This is a deliberate limit on the extensibility of FoLiA in favour of specificity; i.e. you can't just add your own annotation type. If a particular annotation type is not properly accommodated yet, contact the FoLiA developers and we will see how we can extend FoLiA.

For good measure, we again emphasise that this is a limitation on annotation types only, not on the vocabulary the annotation types make use of, which is deliberately separated from the FoLiA standard itself. The next section will elaborate on this.

Below are the categories and underlying annotation types, you can click each for exhaustive information (but please finish this introductory chapter first):

- *Structure Annotation* – This category encompasses annotation types that define the structure of a document, e.g. paragraphs, sentences, words, sections like chapters, lists, tables, etc... These types are not strictly considered linguistic annotation and equivalents are also commonly found in other document formats such as HTML, TEI, MarkDown, LaTeX, and others. For FoLiA it provides the necessary structural basis that linguistic annotation can build on.
  - *Token Annotation* – `<w>` – This annotation type introduces a tokenisation layer for the document. The terms **token** and **word** are used interchangeably in FoLiA as FoLiA itself does not commit to a specific tokenisation paradigm. Tokenisation is a prerequisite for the majority of linguistic annotation types offered by FoLiA and it is one of the most fundamental types of Structure Annotation. The words/tokens are typically embedded in other types of structure elements, such as sentences or paragraphs.
  - *Division Annotation* – `<div>` – Structure annotation representing some kind of division, typically used for chapters, sections, subsections (up to the set definition). Divisions may be nested at will, and may include almost all kinds of other structure elements.
  - *Paragraph Annotation* – `<p>` – Represents a paragraph and holds further structure annotation such as sentences.
  - *Head Annotation* – `<head>` – The `head` element is used to provide a header or title for the structure element in which it is embedded, usually a division (`<div>`)
  - *List Annotation* – `<list>` – Structure annotation for enumeration/itemisation, e.g. bulleted lists.
  - *Figure Annotation* – `<figure>` – Structure annotation for including pictures, optionally captioned, in documents.
  - *Vertical Whitespace* – `<whitespace>` – Structure annotation introducing vertical whitespace
  - *Linebreak* – `<br>` – Structure annotation representing a single linebreak and with special facilities to denote pagebreaks.

- *Sentence Annotation* – `<s>` – Structure annotation representing a sentence. Sentence detection is a common stage in NLP alongside tokenisation.
- *Event Annotation* – `<event>` – Structural annotation type representing events, often used in new media contexts for things such as tweets, chat messages and forum posts (as defined by a user-defined set definition). Note that a more linguistic kind of event annotation can be accomplished with *Entity Annotation* or even *Time Segmentation* rather than this one.
- *Quote Annotation* – `<quote>` – Structural annotation used to explicitly mark quoted speech, i.e. that what is reported to be said and appears in the text in some form of quotation marks.
- *Note Annotation* – `<note>` – Structural annotation used for notes, such as footnotes or warnings or notice blocks.
- *Reference Annotation* – `<ref>` – Structural annotation for referring to other annotation types. Used e.g. for referring to bibliography entries (citations) and footnotes.
- *Table Annotation* – `<table>` – Structural annotation type for creating a simple tabular environment, i.e. a table with rows, columns and cells and an optional header.
- *Part Annotation* – `<part>` – The structure element `part` is a fairly abstract structure element that should only be used when a more specific structure element is not available. Most notably, the part element should never be used for representation of morphemes or phonemes! Part can be used to divide a larger structure element, such as a division, or a paragraph into arbitrary subparts.
- *Utterance Annotation* – `<utt>` – An utterance is a structure element that may consist of words or sentences, which in turn may contain words. The opposite is also true, a sentence may consist of multiple utterances. Utterances are often used in the absence of sentences in a speech context, where neat grammatical sentences can not always be distinguished.
- *Entry Annotation* – `<entry>` – FoLiA has a set of structure elements that can be used to represent collections such as glossaries, dictionaries, thesauri, and wordnets. *Entry annotation* defines the entries in such collections, *Term annotation* defines the terms, and *Definition Annotation* provides the definitions.
- *Term Annotation* – `<term>` – FoLiA has a set of structure elements that can be used to represent collections such as glossaries, dictionaries, thesauri, and wordnets. *Entry annotation* defines the entries in such collections, *Term annotation* defines the terms, and *Definition Annotation* provides the definitions.
- *Definition Annotation* – `<def>` – FoLiA has a set of structure elements that can be used to represent collections such as glossaries, dictionaries, thesauri, and wordnets. *Entry annotation* defines the entries in such collections, *Term annotation* defines the terms, and *Definition Annotation* provides the definitions.
- *Example Annotation* – `<ex>` – FoLiA has a set of structure elements that can be used to represent collections such as glossaries, dictionaries, thesauri, and wordnets. *Examples annotation* defines examples in such collections.
- *Hidden Token Annotation* – `<hiddenw>` – This annotation type allows for a hidden token layer in the document. Hidden tokens are ignored for most intents and purposes but may serve a purpose when annotations on implicit tokens is required, for example as targets for syntactic movement annotation.

▪ *Content Annotation* – This category groups text content and phonetic content, the former being one of the most frequent elements in FoLiA and used to associate text (or a phonetic transcription) with a structural element.

- *Text Annotation* – `<t>` – Text annotation associates actual textual content with structural elements, without it a document would be textless. FoLiA treats it as an annotation like any other.
- *Phonetic Annotation/Content* – `<ph>` – This is the phonetic analogy to text content (`<t>`) and allows associating a phonetic transcription with any structural element, it is often used in a speech context. Note that for actual segmentation into phonemes, FoLiA has another related type: `Phonological Annotation`

  - *Raw Content* – `<content>` – This associates raw text content which can not carry any further annotation. It is used in the context of *Gap Annotation*

- *Inline Annotation* – This category encompasses (linguistic) annotation types describing a single structural element. Examples are Part-of-Speech Annotation or Lemmatisation, which often describe a single token.
  - *Part-of-Speech Annotation* – `<pos>` – Part-of-Speech Annotation, one of the most common types of linguistic annotation. Assigns a lexical class to words.
  - *Lemmatisation* – `<lemma>` – Lemma Annotation, one of the most common types of linguistic annotation. Represents the canonical form of a word.
  - *Domain/topic Annotation* – `<domain>` – Domain/topic Annotation. A form of inline annotation used to assign a certain domain or topic to a structure element.
  - *Sense Annotation* – `<sense>` – Sense Annotation allows to assign a lexical semantic sense to a word.
  - *Error Detection Annotation (DEPRECATED)* – `<errordetection>` – This annotation type is deprecated in favour of *Observation Annotation* and only exists for backward compatibility.
  - *Subjectivity Annotation (DEPRECATED)* – `<subjectivity>` – This annotation type is deprecated in favour of *Sentiment Annotation* and only exists for backward compatibility.
  - *Language Annotation* – `<lang>` – Language Annotation simply identifies the language a part of the text is in. Though this information is often part of the metadata, this form is considered an actual annotation.
- *Span Annotation* – This category encompasses (linguistic) annotation types that span one or more structural elements. Examples are (Named) Entities or Multi-word Expressions, Dependency Relations, and many others. FoLiA implements these as a stand-off layer that refers back to the structural elements (often words/tokens). The layer itself is embedded in a structural level of a wider scope (such as a sentence).
  - *Syntactic Annotation* – `<su>` – Assign grammatical categories to spans of words. Syntactic units are nestable and allow representation of complete syntax trees that are usually the result of consistuency parsing.
  - *Chunking* – `<chunk>` – Assigns shallow grammatical categories to spans of words. Unlike syntax annotation, chunks are not nestable. They are often produced by a process called Shallow Parsing, or alternatively, chunking.
  - *Entity Annotation* – `<entity>` – Entity annotation is a broad and common category in FoLiA. It is used for specifying all kinds of multi-word expressions, including but not limited to named entities. The set definition used determines the vocabulary and therefore the precise nature of the entity annotation.
  - *Dependency Annotation* – `<dependency>` – Dependency relations are syntactic relations between spans of tokens. A dependency relation takes a particular class and consists of a single head component and a single dependent component.
  - *Time Segmentation* – `<timesegment>` – FoLiA supports time segmentation to allow for more fine-grained control of timing information by associating spans of words/tokens with exact timestamps. It can provide a more linguistic alternative to *Event Annotation*.
  - *Coreference Annotation* – `<coreferencechain>` – Relations between words that refer to the same referent (anaphora) are expressed in FoLiA using Coreference Annotation. The co-reference relations are expressed by specifying the entire chain in which all links are coreferent.
  - *Semantic Role Annotation* – `<semrole>` – This span annotation type allows for the expression of semantic roles, or thematic roles. It is often used together with *Predicate Annotation*
  - *Predicate Annotation* – `<predicate>` – Allows annotation of predicates, this annotation type is usually used together with Semantic Role Annotation. The types of predicates are defined by a user-defined set definition.

  - *Observation Annotation* – `<observation>` – Observation annotation is used to make an observation pertaining to one or more word tokens. Observations offer a an external qualification on part of a text. The qualification is expressed by the class, in turn defined by a set. The precise semantics of the observation depends on the user-defined set.
  - *Sentiment Annotation* – `<sentiment>` – Sentiment analysis marks subjective information such as sentiments or attitudes expressed in text. The sentiments/attitudes are defined by a user-defined set definition.
  - *Statement Annotation* – `<statement>` – Statement annotation, sometimes also refered to as attribution, allows to decompose statements into the source of the statement, the content of the statement, and the way these relate, provided these are made explicit in the text.
  - *Modality Annotation* – `<modality>` – Modality annotation is used to describe the relationship between cue word(s) and the scope it covers. It is primarily used for the annotation of negation, but also for the annotation of factuality, certainty and truthfulness:.

- *Subtoken Annotation* – This category contains morphological annotation and phonological annotation, i.e. the segmentation of a word into morphemes and phonemes, and recursively so if desired. It is a special category that mixes characteristics from structure annotation (the `morpheme` and `phoneme` elements are very structure-like) and also from span annotation, as morphemes and phonemes are embedded in an annotation layer and refer back to the text/phonetic content they apply to. Like words/tokens, these elements may also be referenced from `wref` elements.
  - *Morphological Annotation* – `<morpheme>` – Morphological Annotation allows splitting a word/token into morphemes, morphemes itself may be nested. It is embedded within a layer `morphology` which can be embedded within word/tokens.
  - *Phonological Annotation* – `<phoneme>` – The smallest unit of annotatable speech in FoLiA is the phoneme level. The phoneme element is a form of structure annotation used for phonemes. Alike to morphology, it is embedded within a layer `phonology` which can be embedded within word/tokens.

- *Text Markup Annotation* – The text content element (`<t>`) allows within its scope elements of a this category; these are **Text Markup** elements, they always contain textual content and apply a certain markup to certain spans of the text. One of it's common uses is for styling (emphasis, underlines, etc.). Text markup elements may be nested.
  - *Style Annotation* – `<t-style>` – This is a text markup annotation type for applying styling to text. The actual styling is defined by the user-defined set definition and can for example included classes such as italics, bold, underline
  - *Hyphenation* – `<t-hbr>` – This is a text-markup annotation form that indicates where in the original text a linebreak was inserted and a word was hyphenised.
  - *Horizontal Whitespace* – `<t-hspace>` – Markup annotation introducing horizontal whitespace

- *Higher-order Annotation* – Higher-order Annotation groups a very diverse set of annotation types that are considered *annotations on annotations*
  - *Correction Annotation* – `<correction>` – Corrections are one of the most complex annotation types in FoLiA. Corrections can be applied not just over text, but over any type of structure annotation, inline annotation or span annotation. Corrections explicitly preserve the original, and recursively so if corrections are done over other corrections.
  - *Gap Annotation* – `<gap>` – Sometimes there are parts of a document you want to skip and not annotate at all, but include as is. This is where gap annotation comes in, the user-defined set may indicate the kind of gap. Common omissions in books are for example front-matter and back-matter, i.e. the cover.
  - *Relation Annotation* – `<relation>` – FoLiA provides a facility to relate arbitrary parts of your document with other parts of your document, or even with parts of other FoLiA documents or external resources, even in other formats. It thus allows linking resources together. Within this context, the `xref` element is used to refer to the linked FoLiA elements.

- *Span Relation Annotation* – `<spanrelation>` – Span relations are a stand-off extension of relation annotation that allows for more complex relations, such as word alignments that include many-to-one, one-to-many or many-to-many alignments. One of its uses is in the alignment of multiple translations of (parts) of a text.
- *Metric Annotation* – `<metric>` – Metric Annotation is a form of higher-order annotation that allows annotation of some kind of measurement. The type of measurement is defined by the class, which in turn is defined by the set as always. The metric element has a `value` attribute that stores the actual measurement, the value is often numeric but this needs not be the case.
- *String Annotation* – `<str>` – This is a form of higher-order annotation for selecting an arbitrary substring of a text, even untokenised, and allows further forms of higher-order annotation on the substring. It is also tied to a form of text markup annotation.
- *Alternative Annotation* – `<alt>` – This form of higher-order annotation encapsulates alternative annotations, i.e. annotations that are posed as an alternative option rather than the authoratitive chosen annotation
- *Comment Annotation* – `<comment>` – This is a form of higher-order annotation that allows you to associate comments with almost all other annotation elements
- *Description Annotation* – `<desc>` – This is a form of higher-order annotation that allows you to associate descriptions with almost all other annotation elements
- *External Annotation* – `<external>` – External annotation makes a reference to an external FoLiA document whose structure is inserted at the exact place the external element occurs.

1.2 Vocabulary sets

FoLiA specifically defines various types of annotation, but it never defines the vocabulary (aka label/tag sets) you can use for those annotations. The vocabulary for, for instance, Part-of-Speech annotation can be defined by anyone in a separate publicly available file known as a **Set Definition**. Anybody is free to create and host their own set definitions on the internet. These set definitions are typically formulated according to a linked open data model (SKOS) and as-such provide a semantic foundation. Each FoLiA document *declares* in its metadata section, what set definitions to use (described by a URL pointing to a set definition file) for what annotation types. The individual labels inside a set are called **classes** in the FoLiA paradigm. Classes in a Part-of-Speech tagset, for instance, could be `Noun`, `Verb` or `Adjective`, or a more symbolic version thereof (human readable labelling is exlusively done inside the set definition, classes typically refer to more symbollic names, such as `N`, `V` or `ADJ` in this case).

This vocabulary paradigm of independently defined sets and classes is a fundamental part of FoLiA and stretches accross all annotation types.

See also:

Read the full specification in the following section: *Set Definitions (Vocabulary)*

1.3 Validation

If you create FoLiA documents in any shape or form, it is of great importance that you validate whether they indeed conform to the FoLiA specification; otherwise they can not be processed correctly by any FoLiA-aware software. FoLiA is a strict format by design, we prefer to be explicit and do away with any ambiguity or any ad-hoc constructions, this ensures that parsing FoLiA is clear for both humans and machines. Specific validator software is provided to this end.

- A first level of validation is performed by comparing your document against the FoLiA schema (in RelaxNG), this gives you a good indication whether the document is formed corrected; but is **not sufficient** for full validation!

- For full validation, process the document using one of the provided validation tools. These tools make a distinction between **shallow validation** and **deep validation**, the distinction being that only in the latter case the validity of all used classes will be put to the test using the set definitions. Shallow validations allows users to still use FoLiA without formally defining their annotation vocabularies.

Validators are provided by the FoLiA tools (Python) or by the FoliAutils (C++), a command-line example of installation and invocation of the former:

```
$ pip install folia-tools
$ foliavalidator myfoliadocument.folia.xml
```

1.4 Metadata

Every FoLiA document starts with a metadata block, this contains at least a set of **declarations of used annotation types**, which is always mandatory. Optionally it then contains a **provenance** section and after that there is space for custom metadata, either document-wide metadata or submetadata applying to particular parts of the document.

1.4.1 Annotation Declarations

All annotation types that are used in a FoLiA document have to be *declared*. In the metadata block you will find the `<annotations>` block in which each annotation type that occurs in the document is mentioned, i.e. declared. So does your document include Part of Speech tagging? Then there will be an entry declaring it does so, and linking to the set definition used.

This allows software to identify exactly what a FoLiA document consists of without needing to go through the entire document, on the basis of this software can determine whether it can handle the document in the first place. You can for instance imagine an NLP tool that does Named Entity Recognition but requires Part-of-Speech tags and Lemmas to do so, feeding it a FoLiA document without such annotation layers would then be pointless and easy to detect.

See also:

Read the full specification in the following section: *Annotation Declarations*

1.4.2 Provenance Data

Throughout its lifecycle, a FoLiA document may be enriched by multiple FoLiA-aware NLP tools. The provenance block in the metadata header of the document allows us to register precisely what tools were invoked, and optionally when they were invoked and by whom. It is tied to the *Declarations* section.

See also:

Read the full specification in the following section: provenance_data

1.4.3 Document Metadata

FoLiA has support for metadata. Here we define metadata as distinct from (linguistic) annotation in the sense that it is information that describes either the document as a whole or a significant sub-part thereof, as opposed to a particular annotation on the text/speech, which is already covered by FoLiA's main paradigm. Metadata contains information such as authorship of the document, affiliations, sources, licenses, publication date, or whatever else you can think of. Note that it's up to the resource creator, FoLiA does not define any metadata vocabulary!

FoLiA offers a simple native metadata system, which is essentially just a simple key-value store. Alternatively, you can embed foreign metadata schemes such as Dublin Core, CMDI, or whatever you please. You can also refer to metadata in external files, keeping it all separate from the FoLiA document.

In addition to document-wide metadata, i.e. metadata that is applicable to the document as a whole, we already mentioned that FoLiA supports metadata on arbitrary parts of the document. This is referred to as submetadata.

See also:

Read the full specification in the following section: *Metadata*

1.5 Document structure

FoLiA is a document-based format, representing each document and all relevant annotations in a single XML file. [#fex]

We have not included any XML examples in this introduction thus-far, but from now on we will make heavy use of it. From this point forward, we therefore assume the reader has at least a basic familiarity with XML, its use of elements, attributes, comments and a simple understanding of the notion of an XML namespace and an XML schema. If not, we recommend the following XML Tutorial.

In our first XML snippet, we show the basic structure of such a FoLiA document is as follows and should always be UTF-8 encoded.

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/FoLiA"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="2.0"
  xml:id="example">
  <metadata>
      <annotations>
          ...
      </annotations>
      <provenance>
          ..
      </provenance>
      ...
  </metadata>
  <text xml:id="example.text">
     ...
  </text>
</FoLiA>
```

The root element of a FoLiA document is always the `FoLiA` element. This, and *all* other FoLiA elements should always be in the FoLiA XML Namespace, `http://ilk.uvt.nl/FoLiA`[2] .

The mandatory `version` attribute describes the FoLiA version that the document complies to (this is **not** the version of the document! There is room in the provenance_data for that). The document as a whole always carries an ID (`xml:id`), like all identifiers in FoLiA, this has to be a unique string. More about identifiers can be read in the next section.

The structure of a FoLiA document can roughly be divided into two parts, the `metadata` section and a body, the body is formed by either the `<text>` element or the `<speech>` element (see Speech for more information about using FoLiA for speech). The body elements (`<text>`/`<speech>`) are structural elements but take no sets, classes, nor a declaration.

The `metadata` section features a mandatory `annotations` section containing the *Annotation Declarations*, next is the optional but recommended `provenance` block that contains the provenance_data. After this there is space for other *Metadata*.

[2] For historical reasons, the XML namespace URI refers to a research group at the University of Tilburg where FoLiA was first founded, but which no longer exists.

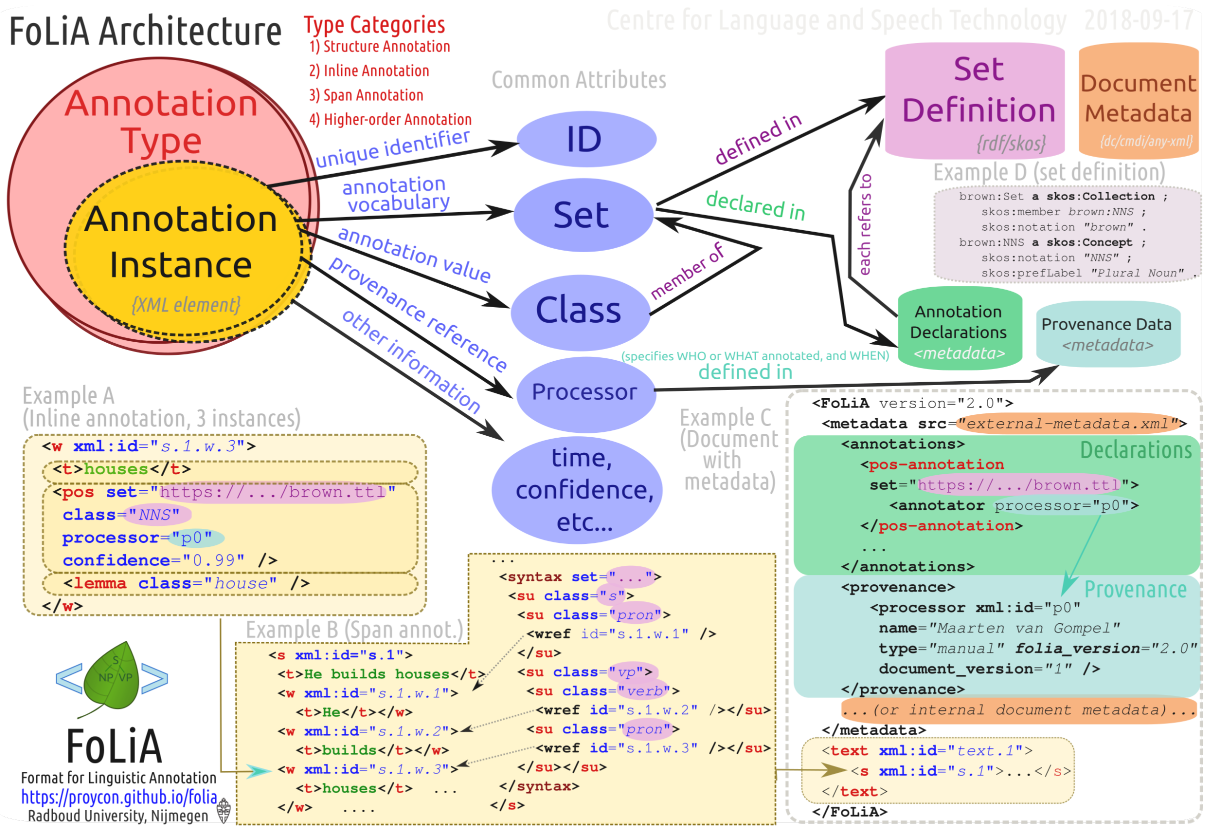
Note: Do not confuse the `<text>` body element with the `<t>` element and `<text-annotation>` declaration, which are both for *Text Annotation*.

1.6 Annotation Instances

All forms of annotation in FoLiA are encoded using an distinct XML element. The first few layers of nested XML elements are usually structural elements (see *Structure Annotation*) such as divisions, paragraphs and sentences. Then the deepest structure layer is usually tokenisation (`<w>`, *Token Annotation*). Within these structures, you find inline annotation elements (see *Inline Annotation*) encoding linguistic information, you also find *layers* with span annotation (see *Span Annotation*), which refer back to the tokens/words in a stand-off fashion.

Whatever the annotation type, all annotation elements for it are bound by the same paradigm, making FoLiA predictable and consistent to a large degree. Central to this paradigm are the notion of sets, declarations, set definitions and classes, as introduced in earlier sections, and the notion of *common attributes*, as explained in the next section.

The FoLiA paradigm can be schematically visualised as follows, don't worry if not all the details are immediately clear. This documentation will provide examples for all annotation types to guide you along.



```
brown:Set a skos:Collection ;
    skos:member brown:NNS ;
    skos:notation "brown" .
brown:NNS a skos:Concept ;
    skos:notation "NNS" ;
    skos:prefLabel "Plural Noun" .
```

```
<w xml:id="s.1.w.3">
 <t>houses</t>
 <pos set="https://.../brown.ttl"
  class="NNS"
  processor="p0"
  confidence="0.99" />
  <lemma class="house" />
</w>
```

```
<s xml:id="s.1">
 <t>He builds houses</t>
 <w xml:id="s.1.w.1">
   <t>He</t></w>
 <w xml:id="s.1.w.2">
   <t>builds</t></w>
 <w xml:id="s.1.w.3">
   <t>houses</t>  ...
 </w>   ....
```

```
...
 <syntax set="...">
  <su class="s">
   <su class="pron">
    <wref id="s.1.w.1" />
   </su>
   <su class="vp">
    <su class="verb">
     <wref id="s.1.w.2" /></su>
    <su class="pron">
     <wref id="s.1.w.3" /></su>
   </su></su>
 </syntax>
</s>
```

```
<FoLiA version="2.0">
 <metadata src="external-metadata.xml">
   <annotations>
     <pos-annotation
      set="https://.../brown.ttl">
       <annotator processor="p0">
     </pos-annotation>
     ...
   </annotations>
   <provenance>
     <processor xml:id="p0"
      name="Maarten van Gompel"
      type="manual" folia_version="2.0"
      document_version="1" />
   </provenance>
   ...(or internal document metadata)...
 </metadata>
  <text xml:id="text.1">
    <s xml:id="s.1">...</s>
  </text>
</FoLiA>
```

1.6.1 Common attributes

Annotation elements in FoLiA carry so-called *common attributes*, these are common properties, represented as XML attributes, that can be set on different annotations. The exact subset of mandatory or optional common attributes differs slightly per annotation type. In this documentation we will explicitly list the required and optional common attributes per annotation type. Altogether, we distinguish the following:

Core Attributes:

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `id` – A reference to the ID of another element. This is a reference and not an assignment, unlike xml:id, so do not confuse the two! It is only supported on certain elements that are referential in nature.

Provenance attributes:

- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.

Authorship attributes, these provides a simpler mechanism stemming from earlier versions of FoLiA and can be used without full provenance (instead of `processor`):

- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.

Annotation attributes:

- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `textclass` – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to `current`. See *Text class attribute (advanced)*.
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always `yes`, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to `no`. This allows for reconstruction of the detokenised original text.

Speech attributes, the following attributes apply mostly in a speech context (please read speech for more):

- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.

XLink attributes, the following apply mainly on text and text markup elements and allow creating hyperlinks. See the section *Hyperlinks* for details.

- `xlink:href` – Creates a hyperlink on a text to the specified URL

- `xlink:type` – Specifies the type of the hyperlink. (should be set to `simple` in almost all cases)

Processing tags; These tags can be used by FoLiA tools to help their processing. We're essentially encoding some extra cue in the FoLiA to help another tool do its job, and such a cue may be needed because the information is not present in the FoLiA yet, or is too complexly encoded for the other tool to unravel.

- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

1.6.2 Identifiers

Many elements in FoLiA take an identifier by which the element is uniquely identifiable. This makes referring to any part of a FoLiA document easy. Identifiers should be unique in the entire document, and ideally within the entire corpus collection if you have multiple documents, though that is a recommendation and not enforced. The ID can be anything that qualifies as a valid ID according to the XML standard, that is, it is a non-colonized name (NCName) that starts with either a letter or an underscore and contains none other than letters, digits, underscores, hyphens and periods. A well proven *convention* for IDs is of a cumulative nature, in which you append the element name, a period, and a sequence number, to the identifier of a parent element higher in the hierarchy. Identifiers are always encoded in the `xml:id` attribute.

The FoLiA document as a whole also carries an identifier.

Identifiers are very important and used throughout the FoLiA format, and mandatory for almost all structural elements. They enable external resources and databases to easily point to a specific part of the document or an annotation therein. FoLiA has been set up in such a way that *identifiers should never change*. Once an identifier is assigned, it should never change, re-numbering is strictly prohibited unless you intentionally want to create a new resource and break compatibility with the old one.

Certain FoLiA elements take an `id` attribute in the FoLiA XML namespace instead of the `XML` namespace, these are always *references* to the ID of another element. It's important not to confuse the two.

1.7 Speech

FoLiA is also suited for annotation of speech data. The following additional generic FoLiA attributes are available for *all* structure annotation elements in a speech context:

- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.

Speech generally asks for a different document structure than text documents. The top-level element for speech-centred resources is `speech`, rather than `text`. Most elements described in the section on text structure may be used under `speech` as well; such as *Division Annotation*, *Sentence Annotation*, *Token Annotation*. Notions such as paragraphs, tables and figures make less sense in a speech context.

In a speech context, you can use *Utterance Annotation* as an alternative or complement to *Sentence Annotation*, as it is often more logical to segment speech into utterances than grammatically sound sentences.

For non-speech events, you can use *Event Annotation*. Consider the following small example, with speech-context attributes associated:

```
<event class="cough" src="soundclip.mp3" begintime="..." endtime="..." />
```

If you want to associate timing information and the use of `begintime` and `endtime` on structural elements is insufficient for your needs, then look into *Time Segmentation*.

Speech has its counterpart to text, in the form of a phonetic or phonological transcription, i.e. a representation of the speech as it was pronounced/recorded. FoLiA has a separate content element for this, see *Phonetic Annotation/Content*. You should still use the normal *Text Annotation* for a normal textual transcription of the speech.

For further segmentation of speech into phonemes, you can use *Phonological Annotation*.

1.7.1 Example

An example of a simple speech document:

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <phon-annotation>
                         <annotator processor="p1" />
          </phon-annotation>
          <utterance-annotation>
                         <annotator processor="p1" />
          </utterance-annotation>
          <token-annotation>
                         <annotator processor="p1" />
          </token-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <speech xml:id="example.speech">
    <utt xml:id="example.utt.1" src="helloworld.mp3"  begintime="00:00:01.000" endtime="00:00:02.000">
        <ph>hel o  w ld</ph>
        <w xml:id="example.utt.1.w.1" begintime="00:00:00.000" endtime="00:00:01.000">
            <ph>hel o </ph>
        </w>
        <w xml:id="example.utt.1.w.2" begintime="00:00:01.000" endtime="00:00:02.000">
            <ph>w ld</ph>
        </w>
    </utt>
  </speech>
</FoLiA>
```

1.8 Hyperlinks

Hyperlinks are ubiquitous in documents from the web and are therefore supported in FoLiA as well. A hyperlink can be defined as a pointer from a span of text to an external resource. In FoLiA, this method is therefore implemented as a simple property on *Text Annotation* itself. Text content elements (`<t>`) as well as any

Text Markup elements that may be contained therein (`<t-*>`), may act as a hyperlink. The link itself is implemented through XLink semantics:

```
<s>
 <w><t>The</t></w>
 <w><t>FoLiA</t></w>
 <w><t>website</t></w>
 <w><t>is</t></w>
 <w><t xlink:type="simple" xlink:href="https://proycon.github.io/folia">here</t></w>
 <w><t>.</t></w>
</s>
```

Or on a substring (see *String Annotation*) in untokenised text:

```
<s>
 <t>The FoLiA website is <t-str xlink:type="simple"
 xlink:href="https://proycon.github.io/folia">here</t-str>.</t>
</s>
```

Before using this, make sure to investigate *Reference Annotation* as well. There we describe a more semantic way of hyperlinking, which uses references (`<ref>` elements) that are actual structure elements. The hyperlinking method described in this section is of a more text-mark-up or stylistic nature. Actual references are usually preferred when applicable.

Another notion of linking is implemented using FoLiA's *relations* (*Relation Annotation*). Relations are actual higher-order annotations that can link anything but it needs not be reflected in the actual text, whereas the hyperlinks described here and the references of *Reference Annotation* do always show in the text.

CHAPTER 2

Metadata

FoLiA supports associating metadata with your document, you will find this in the <metadata> document at the very beginning of the document. An extensive and mandatory part of this metadata is the *Annotation Declarations* block (<annotations>), and second (optionally) the block for provenance_data (<provenance>). The remainder of the <metadata> block may be filled with Document Metadata as described in *Document Metadata* later. on.

2.1 Annotation Declarations

All annotation types that are used in a FoLiA document have to be *declared*. In the metadata block you will find the <annotations> block in which each annotation type that occurs in the document is mentioned, i.e. declared. So does your document include Part of Speech tagging? Then there will be an entry declaring it does so, and linking to the set definition used.

These declarations allow software to identify exactly what a FoLiA document consists of without needing to go through the entire document, on the basis of this software can determine whether it can handle the document in the first place. You can for instance imagine an NLP tool that does Named Entity Recognition but requires Part-of-Speech tags and Lemmas to do so, feeding it a FoLiA document without such annotation layers would then be pointless and easy to detect.

Each annotation type has a specific XML element to use as declaration in the <annotations> block, these all end in with the suffix `-annotations` and take the following attributes:

- `set` - The set should be a URL to a publicly hosted set definition that defines the vocabulary used (see *Set Definitions (Vocabulary)*) with this particular annotation type. Sets are intentionally kept separate from FoLiA itself and can be created by anyone. FoLiA also allows for ad-hoc sets, these are sets that are not actually defined and they are therefore an arbitrary string rather than a URL. They allow for a more flexible use of FoLiA without full formal closure, but limit it to only shallow validation. Some annotation types also work without an associated vocabulary, and for some they are optional, on such declarations the `set` attribute is not used or optional.
- `format` - Set definitions can be stored in several formats, the format may be indicated (not mandatory) by the `format` attribute on each declaration, its value should be a MIME type.
- `alias` - This is an optional attribute that specifies an alias for the set, this is useful in case an annotation type occurs multiple times with distinct sets, in which case individual annotation needs to explicitly mention the set but referring to sets by long URLs gets cumbersome. In such cases annotations can use the alias instead of the full set URL. An alias has to be unique for the annotation type.

Within the scope of each annotation's declaration, you can declare one or more *annotators*, each annotator refers (by ID) to what we call a `processor` in the *provenance data*. These processors represent software tools or human annotators and carry various attributes, e.g. the name of the annotator/tool. So this part of declaration identifies *who* or *what* performed the annotation. Consider the following example:

```
<annotations>
    <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
        <annotator processor="p1.ucto"/>
    </token>
    <pos-annotation set="https://github.com/proycon/folia/blob/master/setdefinitions/
↪cgn.foliaset.ttl">
        <annotator processor="p2.frog"/>
        <annotator processor="p3.proycon"/>
    </pos>
</annotations>
```

The section provenance_data will explain in depth how the processors that the `annotator` elements refer to are defined. If there is only one annotator declared, then this is the default for all annotations of this type and set, in which case individual annotation instances need not refer to any processor. If there are multiple annotators, the individual annotation instances should refer to a processor to disambiguate.

Provenance data is *recommended* but not required in FoLiA. A simpler mechanism from prior to FoLiA v2.0 is also still available: If you do not refer to processors for a certain annotation type and set (i.e. no `<annotator>` elements), then you can specify the following *optional* attributes on your declaration to set a default annotator. They act as a default value that can be overriden on individual annotations:

- `annotator` - The name of the default annotator (either human or software)
- `annotatortype` - Set to `auto` if the default annotator is automatic annotation by software or `manual` for human annotators
- `datetime` – The date and time when all annotations of this type were recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.

2.2 Set definitions

A *set definition* (see *Set Definitions (Vocabulary)*) specifies exactly what classes are allowed in a particular vocabulary. It for example specifies exactly what part-of-speech tags exist. This information is necessary to validate the document completely at its deepest level. If the sets point to URLs that do not exist or are not URLs at all, warnings will be issued. Validation can still proceed but with the notable exception that there is no deep validation of these sets, so no full formal closure.

Though we recommend using and creating actual sets. FoLiA itself is rather agnostic about their existence for most purposes. For deep validation, proper formalisation, and for certain applications they may be required; but as long as they serve as proper *unique identifiers* you can get get away with non-existing sets. In this case, simply do not use a URL but another arbitrary identification string.

If multiple sets are used for the same annotation type, which is perfectly valid, they each need a separate declaration.

See also:

- *Set Definitions (Vocabulary)*
- provenance_data

2.3 Document Metadata

To associate other arbitrary metadata with a FoLiA document, there is FoLiA's native metadata system, in which simple metadata fields can be defined and used at will through the `<meta>` element. The following example shows document-wide metadata:

```
<metadata type="native">
    <annotations>
    ..
    </annotations>
    <meta id="title">Title of my document</meta>
    <meta id="language">eng</meta>
</metadata>
```

The native metadata format just offers a simple key-value store. You can define fields with custom IDs. FoLiA itself does not predefine any, strictly speaking, although certain fields like *language*, *title* and *author* are conventional and can be interpreted by some FoLiA-capable tools and libraries.

The native metadata format is deliberately limited, as various other formats already tackle the metadata issue. FoLiA is able to operate with any other metadata format, such as for example Dublin Core or for example CMDI. The `type` attribute specifies what metadata format is used. We see it was set to `native` for FoLiA's native metadata format, for foreign formats it can be set to any other string.

Foreign metadata can be stored in two ways:

- Externally in a different file
- Internally in the metadata block of the FoLiA document itself

When the metadata is stored externally in a different file, a reference is made from the `src` attribute. As shown in the following example:

```
<metadata type="cmdi" src="/path/or/url/to/metadata.cmdi">
  <annotations>
  ..
  </annotations>
</metadata>
```

If you want to store the metadata in the FoLiA document itself, then the metadata must be places inside a `<foreign-data>` element. All elements under foreign-data *must* be in another XML namespace, that is, not the default FoLiA namespace. Consider the following example for Dublin Core:

```
<metadata type="dc">
  <annotations>
  ..
  </annotations>
  <foreign-data xmlns:dc="http://purl.org/dc/elements/1.1/">
    <dc:identifier>mydoc</dc:identifier>
    <dc:format>text/xml</dc:format>
    <dc:type>Example</dc:type>
    <dc:contributor>proycon</dc:contributor>
    <dc:creator>proycon</dc:creator>
    <dc:language>en</dc:language>
    <dc:publisher>Radboud University</dc:publisher>
    <dc:rights>public Domain</dc:rights>
  </foreign-data>
</metadata>
```

The namespace prefix and the type specified in the `<metadata>` element should match.

2.4 Submetadata

Whereas the metadata discussed in the previous section concerns document-wide metadata, i.e. metadata that is applicable to the document as a whole, FoLiA also supports metadata on arbitrary parts of the document. This we call *submetadata*. Within the `<metadata>` block, one can include one or more `<submetadata>` blocks. Like `<metadata>`, a `<submetadata>` block carries a `type` attribute, a `src` attribute in case the metadata is an external reference, and it may hold `<meta>` elements or `<foreign-data>` elements. It differs from `<metadata>` in that it carries a mandatory `xml:id` attribute and *never* has an `<annotations>` or `<provenance>` block. The ID is in turn used to back to the metadata from particular elements in the text. Such a reference is made using the `metadata` attribute, which is a common FoLiA attribute allowed on many elements. Consider the following example (certain details are omitted for brevity):

```
<FoLiA>
<metadata>
 <annotations>...</annotations>
 <submetadata xml:id="metadata.1" type="native">
   <meta id="author">proycon</meta>
   <meta id="language">nld</meta>
 </submetadata>
 <submetadata xml:id="metadata.2" type="native">
   <meta id="author">Shakespeare</meta>
   <meta id="language">eng</meta>
 </submetadata>
</metadata>
<text>
 <p metadata="metadata.1">
   <t>Het volgende vers komt uit Hamlet:</t>
 </p>
 <p metadata="metadata.2">
  <s><t>To be, or not to be, that is the question:</t></s>
  <s><t>Whether 'tis nobler in the mind to suffer<br/>The slings and arrows of outrageous fortune,<br/>Or to take Arms against a Sea of troubles,<br/> And by opposing end them:</s></t>
 </p>
</text>
</FoLiA>
```

Since metadata can be associated with anything, any arbitrary sub-part of untokenised text can even be selected and associated with the existing facilities `<str>` or `t-str` (see *String Annotation*). Some redundancy occurs only at places where structural boundaries are crossed (the metadata attribute might have to be repeated on multiple structural elements if there is no catch-all structure).

Submetadata is inherited (recursively), i.e. it is not necessary to explicitly assign the `metadata` attribute to the children of an element that already has such an assignment.

2.5 Provenance Data

It is often desireable to know exactly what tools (and what versions thereof and even with what parameters) were invoked in which order to produce a FoLiA document, this is called **provenance data**. In the metadata section, right after the *Annotation Declarations* FoLiA allows for a `<provenance>` block containing this information. It is not mandatory but it is strongly recommended.

The `<provenance>` block defines one or more **processors**, processors are processes or entities that have processed and often performend some kind of manipulation of the document, such as adding annotations. The processors are listed in the order they were invoked. The *Annotation Declarations* in turn link to these processors to tie a particular annotation type and set to one or more processors.

A `<processor>` carries the following attributes:

- `xml:id` (mandatory) – The ID of the processor, this is how it is referred to from the `<annotator processor=".." />` element in the *Annotation Declarations* and from the `processor` attribute (part of the common FoLiA attributes) on individual annotations.
- `name` (mandatory) – The name identifies actual tool or human annotator
- `type` – **Each processor contains a type:**
  - `auto` - (default) - The processor is an automated tool that provided annotations
  - `manual` - The processor refers a manual annotator
  - `generator` - The processor indicates the FoLiA library used by the parent and sibling processors (unless sibling processes specify another generator in their scope)
  - `datasource` - The processor is a reference to a particular data source that was used by the parent processor. If there is no parent processor but it is instead directly part of the provenance chain, often as the very first element, then you can interpret this to be the original data source from which the document sprung.
- `version` – (optional but strongly recommended) is the version of the processor aka tool
- `document_version` (optional) – The version of the document, refers to any label the user desires to indicate a version of the document, so the format is not predetermined and needs not be numeric.
- `command` (optional) – The exact command that was run
- `host` (optional) – The host on which the processor ran, this identifies individual systems on a network/cluster.
- `user` (optional) – The user/executor which ran the processor, this identifies who ran an automated process rather than who the annotator was!
- `src` (optional) – The source of the processor, a URL to the tool itself in case the software is an online tool, or to its website or source code repository if not. If the processor is of the `datasource` type, then this attribute should point to that data set or a website describing it. The `format` attribute can be used to further specify the type of source.
- `format` (optional) – MIME type describing the kind of resource pointed to by `src`. Use `text/html` for websites. Especially useful for processors of type `datasource`.
- `folia_version` (optional) - The folia version that was written
- `begindatetime` (optional) – Specifies when the process started, format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `enddatetime` (optional) – Specifies when the process finished, format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `resourcelink` (optional) - The URI of any RDF resource describing this processor. This allows linking to the external world of linked open data from the provenance chain in FoLiA.
- Additional custom metadata is allowed in the form of `<meta>` elements (just like with folia native metadata) inside the scope of a processor, FoLiA does not define the semantics of any such metadata, i.e. they are tool/application-specific and could for instance be used to specify tool parameters used.

First consider a fairly minimalistic example, note that we include the *Annotation Declarations* as well with a link to the processor:

```
<annotations>
  <token-annotation set="tokconfig-nl">
      <annotator processor="p0" />
  </token-annotation>
```

(continues on next page)

(continued from previous page)

```
</annotations>
<provenance>
    <processor xml:id="p0" name="ucto" version="0.15" folia_version="2.0" command=
↪"ucto -Lnld" host="mhysa" user="proycon" begindatetime="2018-09-12T00:00:00"␣
↪enddatetime="2018-09-12T00:00:10" document_version="1" />
</provenance>
```

Individual annotations in the document can refer to this processor using the `processor` attribute:

```
<w class="PUNCTUATION" processor="p0">
 <t>.</t>
</w>
```

If there is only one `<annotator>` defined for a certain annotation type and set in the *Annotation Declarations*, then it is the default and no `processor` attribute is necessary.

One of the powerful features of processors is that they can be nested, this creates *subprocessors* and captures situations where one processor invokes others as part of its operation. Subprocessors can also provide some extra information on their parent processor, as they can for example state what FoLiA library was used (`type="generator"`) or what data sources were used by the processor (`type="datasource"`). Moreover, arbitrary metadata can be added to any processor in the form of `<meta>` elements (just like with FoLiA's native *Metadata*), FoLiA does not define the semantics of any such metadata, i.e. they are tool/application-specific and could for instance be used to specify tool parameters used. Note that whereas the order of the processors in the *<provenance>* block is strictly significant, the order of subprocessors is not.

With all this in mind, we can expand our previous example:

```
<provenance>
    <processor xml:id="p0" name="ucto" version="0.15" folia_version="2.0" command=
↪"ucto -Lnld" host="mhysa" user="proycon" begindatetime="2018-09-12T00:00:00"␣
↪enddatetime="2018-09-12T00:00:10" document_version="1" />
        <meta id="config">tokconfig-nld</meta>
        <meta id="language">nld</meta>
        <processor xml:id="p0.1" name="libfolia" version="2.0" folia_version="2.0"␣
↪type="generator" />
        <processor xml:id="p0.1" name="tokconfig-nld" version="2.0" folia_version="2.0
↪" type="datasource" />
    </processor>
</provenance>
```

Or consider the following example in which we have a tool that is an annotation environment in which human annotators edit a FoLiA document and add/edit annotations:

```
<provenance>
    <processor xml:id="p2" name="flat" version="0.8" folia_version="2.0" host="flat.
↪science.ru.nl" begindatetime="2018-09-12T00:10:00" enddatetime="2018-09-12T00:20:00
↪" document_version="3">
        <processor xml:id="p2.0" name="foliapy" version="2.0" folia_version="2.0"␣
↪type="generator" />
        <processor xml:id="p2.1" name="proycon" type="manual" />
        <processor xml:id="p2.2" name="ko" type="manual" />
    </processor>
</provenance>
```

From the *Annotation Declarations*, we can then also refer directly to subprocessors. Moreover, a processor can be referred to from multiple annotation types/sets:

```
<annotations>
  ...
  <pos-annotation set="...">
      <annotator processor="p2.1" />
      <annotator processor="p2.2" />
  </pos-annotation>
  <lemma-annotation set="...">
      <annotator processor="p2.1" />
  </lemma-annotation>
  ...
</annotations>
```

Of course, providing all this is not mandatory and requires the specific tool to actually supply this provenance data. It is still possible to have FoLiA documents without provenance data at all.

The following example provides a small but complete FoLiA document with provenance data:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="folia.xsl"?>
<FoLiA xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="http://ilk.uvt.nl/folia"␣
↪xml:id="untitled" generator="manual" version="2.0.0">
  <metadata type="native">
    <annotations>
      <text-annotation set="https://raw.githubusercontent.com/proycon/folia/master/
↪setdefinitions/text.foliaset.ttl"/>
      <paragraph-annotation />
      <sentence-annotation />
      <token-annotation set="tokconfig-nl">
          <annotator processor="p0" />
      </token-annotation>
      <pos-annotation set="http://ilk.uvt.nl/folia/sets/frog-mbpos-cgn">
          <!-- There are multiple annotators, this means that each pos annotation␣
↪should explicitly refer to one of them using the @processor attribute -->
          <annotator processor="p1.1" />
          <annotator processor="p2.1" />
          <annotator processor="p2.2" />
      </pos-annotation>
      <lemma-annotation set="http://ilk.uvt.nl/folia/sets/frog-mblem-nl">
          <!-- There is only one annotator so this will be the default, no need to␣
↪explicitly refer to it from lemma annotations using the @processor attribute -->
          <annotator processor="p1.2" />
      </lemma-annotation>
    </annotations>
    <provenance>
        <processor xml:id="p0" name="ucto" version="0.15" folia_version="2.0" command=
↪"ucto -Lnld" host="mhysa" user="proycon" src="https://github.com/LanguageMachines/
↪ucto" begindatetime="2018-09-12T00:00:00" enddatetime="2018-09-12T00:00:00"␣
↪document_version="1">
            <!-- We can add arbitrary meta fields to any processor, they are not␣
↪defined by FoLiA but application-specific  -->
            <meta id="config">tokconfig-nld</meta>
            <meta id="language">nld</meta>
            <processor xml:id="p0.1" name="libfolia" version="2.0" folia_version="2.0
↪" type="generator" />
        </processor>
        <processor xml:id="p1" name="frog" version="0.16" folia_version="2.0" command=
↪"frog --skip=pn" host="mhysa" user="proycon" src="https://github.com/
↪LanguageMachines/frog" begindatetime="2018-09-12T00:01:00" enddatetime="2018-09-
↪12T00:02:00" document_version="2">
```

(continues on next page)

(continued from previous page)

```
            <processor xml:id="p1.0" name="libfolia" version="2.0" folia_version="2.0" type="generator" />
            <processor xml:id="p1.1" name="mbpos" version="0.16">
                  <processor xml:id="p1.1.1" type="datasource" name="CGN Corpus" version="unknown" />
                  <processor xml:id="p1.1.2" type="datasource" name="WOTAN Corpus" version="unknown" />
                  <processor xml:id="p1.1.3" type="datasource" name="DCOI Corpus" version="unknown" />
                  <processor xml:id="p1.1.4" type="datasource" name="Lassy Klein Corpus" version="unknown" />
            </processor>
            <processor xml:id="p1.2" name="mblem" />
        </processor>
        <processor xml:id="p2" name="flat" version="0.8" folia_version="2.0" host="flat.science.ru.nl" src="https://flat.science.ru.nl" begindatetime="2018-09-12T00:10:00" enddatetime="2018-09-12T00:20:00" document_version="3">
            <processor xml:id="p2.0" name="foliapy" version="2.0" folia_version="2.0" type="generator" src="https://github.com/proycon/foliapy" />
            <processor xml:id="p2.1" name="proycon" type="manual" />
            <processor xml:id="p2.2" name="ko" type="manual" />
        </processor>
    </provenance>
  </metadata>
  <text xml:id="untitled.text">
    <p xml:id="untitled.p.1">
      <s xml:id="untitled.p.1.s.1">
        <t>De belastingdienst doet aangifte tegen frauderende mensen.</t>
        <w xml:id="untitled.p.1.s.1.w.1" class="WORD">
          <t>De</t>
          <pos class="LID(bep,stan,rest)" confidence="0.999701" head="LID" set="http://ilk.uvt.nl/folia/sets/frog-mbpos-cgn" processor="p1.1">
            <feat class="bep" subset="lwtype"/>
            <feat class="stan" subset="naamval"/>
            <feat class="rest" subset="npagr"/>
          </pos>
          <lemma class="de"/>
        </w>
        <w xml:id="untitled.p.1.s.1.w.2" class="WORD">
          <t>belastingdienst</t>
          <pos class="N(soort,ev,basis,zijd,stan)" confidence="0.998836" head="N" set="http://ilk.uvt.nl/folia/sets/frog-mbpos-cgn" processor="p2.1">
            <feat class="soort" subset="ntype"/>
            <feat class="ev" subset="getal"/>
            <feat class="basis" subset="graad"/>
            <feat class="zijd" subset="genus"/>
            <feat class="stan" subset="naamval"/>
          </pos>
          <lemma class="belastingdienst"/>
        </w>
        <w xml:id="untitled.p.1.s.1.w.3" class="WORD">
          <t>doet</t>
          <pos class="WW(pv,tgw,met-t)" confidence="0.999262" head="WW" set="http://ilk.uvt.nl/folia/sets/frog-mbpos-cgn" processor="p1.1">
            <feat class="pv" subset="wvorm"/>
            <feat class="tgw" subset="pvtijd"/>
```

(continues on next page)

(continued from previous page)

```
            <feat class="met-t" subset="pvagr"/>
          </pos>
          <lemma class="doen"/>
        </w>
        <w xml:id="untitled.p.1.s.1.w.4" class="WORD">
          <t>aangifte</t>
          <pos class="N(soort,ev,basis,zijd,stan)" confidence="0.998701" head="N" set=
↪"http://ilk.uvt.nl/folia/sets/frog-mbpos-cgn" processor="p2.2">
            <feat class="soort" subset="ntype"/>
            <feat class="ev" subset="getal"/>
            <feat class="basis" subset="graad"/>
            <feat class="zijd" subset="genus"/>
            <feat class="stan" subset="naamval"/>
          </pos>
          <lemma class="aangifte"/>
        </w>
        <w xml:id="untitled.p.1.s.1.w.5" class="WORD">
          <t>tegen</t>
          <pos class="VZ(init)" confidence="0.854093" head="VZ" set="http://ilk.uvt.
↪nl/folia/sets/frog-mbpos-cgn" processor="p1.1">
            <feat class="init" subset="vztype"/>
          </pos>
          <lemma class="tegen"/>
        </w>
        <w xml:id="untitled.p.1.s.1.w.6" class="WORD">
          <t>frauderende</t>
          <pos class="WW(od,prenom,met-e)" confidence="0.96" head="WW" set="http://
↪ilk.uvt.nl/folia/sets/frog-mbpos-cgn" processor="p1.1">
            <feat class="od" subset="wvorm"/>
            <feat class="prenom" subset="positie"/>
            <feat class="met-e" subset="buiging"/>
          </pos>
          <lemma class="frauderen"/>
        </w>
        <w xml:id="untitled.p.1.s.1.w.7" class="WORD" space="no">
          <t>mensen</t>
          <pos class="N(soort,mv,basis)" confidence="0.999865" head="N" set="http://
↪ilk.uvt.nl/folia/sets/frog-mbpos-cgn" processor="p1.1">
            <feat class="soort" subset="ntype"/>
            <feat class="mv" subset="getal"/>
            <feat class="basis" subset="graad"/>
          </pos>
          <lemma class="mens"/>
        </w>
        <w xml:id="untitled.p.1.s.1.w.8" class="PUNCTUATION">
          <t>.</t>
          <pos class="LET()" confidence="1" head="LET" set="http://ilk.uvt.nl/folia/
↪sets/frog-mbpos-cgn" processor="p1.1"/>
          <lemma class="."/>
        </w>
      </s>
    </p>
  </text>
</FoLiA>
```

And another more real-life example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="folia.xsl"?>
<FoLiA xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="http://ilk.uvt.nl/folia" xml:id="example.deep" generator="libfolia-v1.5" version="2.0.0">
  <metadata type="native">
    <annotations>
      <text-annotation>
                         <annotator processor="p1" />
      </text-annotation>
      <sentence-annotation>
                         <annotator processor="p1" />
      </sentence-annotation>
      <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/uctodata/folia1.4/setdefinitions/tokconfig-nld.foliaset.ttl">
                         <annotator processor="p2" />
      </token-annotation>
      <pos-annotation set="https://raw.githubusercontent.com/proycon/folia/master/setdefinitions/frog-mbpos-cgn">
                         <annotator processor="p3.1" />
      </pos-annotation>
      <lemma-annotation set="https://raw.githubusercontent.com/proycon/folia/master/setdefinitions/frog-mblem-nl">
                         <annotator processor="p3.2" />
      </lemma-annotation>
    </annotations>
    <provenance>
       <processor xml:id="p1" name="proycon" type="manual" />
       <processor xml:id="p2" name="ucto" version="0.14" />
       <processor xml:id="p3" name="frog" version="0.16" begindatetime="2016-11-15T15:12:00">
           <processor xml:id="p3.0" name="libfolia" version="1.14" type="generator" />
           <processor xml:id="p3.1" name="mbpos" version="1.0" />
           <processor xml:id="p3.2" name="mblem" version="1.1" />
       </processor>
    </provenance>
    <meta id="language">nld</meta>
  </metadata>
  <text xml:id="example.deep.text">
      <s xml:id="example.deep.p.1.s.1">
        <t>De Russen kennen Nova Zembla sinds de 11e of 12e eeuw, toen handelaars van Novgorod het eiland al aandeden.</t>
        <w xml:id="example.deep.p.1.s.1.w.1" class="WORD">
          <t>De</t>
          <pos class="LID(bep,stan,rest)" confidence="0.779762" head="LID">
            <feat class="bep" subset="lwtype"/>
            <feat class="stan" subset="naamval"/>
            <feat class="rest" subset="npagr"/>
          </pos>
          <lemma class="de"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.2" class="WORD">
          <t>Russen</t>
          <pos class="SPEC(deeleigen)" confidence="1" head="SPEC">
            <feat class="deeleigen" subset="spectype"/>
          </pos>
          <lemma class="Russen"/>
        </w>
```

(continues on next page)

(continued from previous page)

```
        <w xml:id="example.deep.p.1.s.1.w.3" class="WORD">
          <t>kennen</t>
          <pos class="WW(pv,tgw,mv)" confidence="0.833333" head="WW">
            <feat class="pv" subset="wvorm"/>
            <feat class="tgw" subset="pvtijd"/>
            <feat class="mv" subset="pvagr"/>
          </pos>
          <lemma class="kennen"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.4" class="WORD">
          <t>Nova</t>
          <pos class="SPEC(deeleigen)" confidence="1" head="SPEC">
            <feat class="deeleigen" subset="spectype"/>
          </pos>
          <lemma class="Nova"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.5" class="WORD">
          <t>Zembla</t>
          <pos class="SPEC(deeleigen)" confidence="1" head="SPEC">
            <feat class="deeleigen" subset="spectype"/>
          </pos>
          <lemma class="Zembla"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.6" class="WORD">
          <t>sinds</t>
          <pos class="VZ(init)" confidence="0.999078" head="VZ">
            <feat class="init" subset="vztype"/>
          </pos>
          <lemma class="sinds"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.7" class="WORD">
          <t>de</t>
          <pos class="LID(bep,stan,rest)" confidence="0.981886" head="LID">
            <feat class="bep" subset="lwtype"/>
            <feat class="stan" subset="naamval"/>
            <feat class="rest" subset="npagr"/>
          </pos>
          <lemma class="de"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.8" class="NUMBER-ORDINAL">
          <t>11e</t>
          <pos class="TW(rang,prenom,stan)" confidence="0.990632" head="TW">
            <feat class="rang" subset="numtype"/>
            <feat class="prenom" subset="positie"/>
            <feat class="stan" subset="naamval"/>
          </pos>
          <lemma class="11"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.9" class="WORD">
          <t>of</t>
          <pos class="VG(neven)" confidence="0.855677" head="VG">
            <feat class="neven" subset="conjtype"/>
          </pos>
          <lemma class="of"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.10" class="NUMBER-ORDINAL">
```

(continues on next page)

(continued from previous page)

```
          <t>12e</t>
          <pos class="TW(rang,prenom,stan)" confidence="0.990632" head="TW">
            <feat class="rang" subset="numtype"/>
            <feat class="prenom" subset="positie"/>
            <feat class="stan" subset="naamval"/>
          </pos>
          <lemma class="12"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.11" class="WORD" space="no">
          <t>eeuw</t>
          <pos class="N(soort,ev,basis,zijd,stan)" confidence="0.999633" head="N">
            <feat class="soort" subset="ntype"/>
            <feat class="ev" subset="getal"/>
            <feat class="basis" subset="graad"/>
            <feat class="zijd" subset="genus"/>
            <feat class="stan" subset="naamval"/>
          </pos>
          <lemma class="eeuw"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.12" class="PUNCTUATION">
          <t>,</t>
          <pos class="LET()" confidence="1" head="LET"/>
          <lemma class=","/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.13" class="WORD">
          <t>toen</t>
          <pos class="VG(onder)" confidence="0.571429" head="VG">
            <feat class="onder" subset="conjtype"/>
          </pos>
          <lemma class="toen"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.14" class="WORD">
          <t>handelaars</t>
          <pos class="N(soort,mv,basis)" confidence="0.99944" head="N">
            <feat class="soort" subset="ntype"/>
            <feat class="mv" subset="getal"/>
            <feat class="basis" subset="graad"/>
          </pos>
          <lemma class="handelaar"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.15" class="WORD">
          <t>van</t>
          <pos class="VZ(init)" confidence="0.999469" head="VZ">
            <feat class="init" subset="vztype"/>
          </pos>
          <lemma class="van"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.16" class="WORD">
          <t>Novgorod</t>
          <pos class="SPEC(deeleigen)" confidence="1" head="SPEC">
            <feat class="deeleigen" subset="spectype"/>
          </pos>
          <lemma class="Novgorod"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.17" class="WORD">
          <t>het</t>
```

(continues on next page)

(continued from previous page)

```
          <pos class="LID(bep,stan,evon)" confidence="0.996855" head="LID">
            <feat class="bep" subset="lwtype"/>
            <feat class="stan" subset="naamval"/>
            <feat class="evon" subset="npagr"/>
          </pos>
          <lemma class="het"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.18" class="WORD">
          <t>eiland</t>
          <pos class="N(soort,ev,basis,onz,stan)" confidence="0.996804" head="N">
            <feat class="soort" subset="ntype"/>
            <feat class="ev" subset="getal"/>
            <feat class="basis" subset="graad"/>
            <feat class="onz" subset="genus"/>
            <feat class="stan" subset="naamval"/>
          </pos>
          <lemma class="eiland"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.19" class="WORD">
          <t>al</t>
          <pos class="BW()" confidence="0.90383" head="BW"/>
          <lemma class="al"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.20" class="WORD" space="no">
          <t>aandeden</t>
          <pos class="WW(pv,verl,mv)" confidence="0.999559" head="WW">
            <feat class="pv" subset="wvorm"/>
            <feat class="verl" subset="pvtijd"/>
            <feat class="mv" subset="pvagr"/>
          </pos>
          <lemma class="aandoen"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.21" class="PUNCTUATION">
          <t>.</t>
          <pos class="LET()" confidence="1" head="LET"/>
          <lemma class="."/>
        </w>
      </s>
  </text>
</FoLiA>
```

Another example with many annotation types and extensive provenance data:

See also:

Annotation Declarations Set Definitions (Vocabulary)

CHAPTER 3

Set Definitions (Vocabulary)

3.1 Introduction

The sets and classes used by the various linguistic annotation types are never defined in the FoLiA documents themselves, but externally in **set definitions**.

By using set definitions, a FoLiA document can be validated on a deep level, i.e. the validity of the used classes can be tested. Set definitions provide semantics to the FoLiA documents that use them and are an integral part of FoLiA. When set definitions are absent, validation can only be conducted on a shallow level that is agnostic about all sets and the classes therein.

Recall that all sets that are used need to be declared in the *Annotation Declarations* section in the document header and that they point to URLs holding a FoLiA set definitions. If no set definition files are associated, then a full in-depth validation cannot take place.

The role of FoLiA Set Definitions is:

- to define which classes are valid in a set
- to define which subsets and classes are valid in *Features* in a set
- to constrain which subsets and classes may co-occur in an annotation of the set
- to allow enumeration over classes and subsets
- to assign human-readable labels to symbolic classes
- to relate classes to external resources defining them (data category registries, linked data)
- to define a hierarchy/taxonomy of classes

Prior to FoLiA v1.4, set definitions were stored in a simple custom XML format, distinct from FoLiA itself, which we call the legacy format and which is still supported for backward compatibility. Since FoLiA v1.4 however, we strongly prefer and recommend to store the set definitions as RDF [*RDF*], i.e. the technology that powers the semantic web. In this way, set definitions provide a formal semantic layer for FoLiA.

Set definitions may be stored in various common RDF serialisation formats. The format can be indicated on the declarations in the document metadata using the `format` attribute, recognised values are:

- `application/rdf+xml` – XML for RDF (assumed for `rdf.xml` or `rdf` extensions
- `text/turtle` – Turtle (for RDF) (assumed for `ttl` extensions)
- `text/n3` – Notation 3 (for RDF) (assumed for `n3` extensions)

- `application/foliaset+xml` - Legacy FoLiA Set Definition format (XML) (assumed for `xml` extensions and in most other cases)

FoLiA applications should attempt to autodetect the format based on the extension. Not all applications may be able to deal with all formats/serialisations, however.

In this documentation, we will use the Turtle format for RDF, alongside our older legacy format. In all cases, FoLiA requires that only one set is defined per file, any other defined sets must be subsets of the primary set. In our legacy XML format, an otherwise empty set definition would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<set
 xmlns="http://ilk.uvt.nl/folia"
 xml:id="your-set-id" type="closed" label="Human readable label for your set">
</set>
```

Note that the legacy XML format takes an XML namespace that is always the same (the FoLiA namespace).

In RDF, FoLiA Set Definitions follow a particular model. The model we use is a small superset of the SKOS model. SKOS is a W3C standard for the representation of Simple Knowledge Organization Systems [SKOS]. Not everything can be expressed in the SKOS model, so we have some extensions to it which are formally defined in our set definition schema at https://raw.githubusercontent.com/proycon/folia/master/schemas/foliasetdefinition.ttl. The RDF namespace for our extension is `http://folia.science.ru.nl/setdefinition#`, for which we use the prefix `fsd:` generally, though this is mere convention.

Some familiarity with RDF and Turtle is recommended for this chapter, but it is also still possible to work with the XML legacy format, which is a bit more concise and simple, and automatically convert it to Turtle format using our superset of the SKOS model.

Your own set definitions typically has its own RDF namespace, which in Turtle syntax is defined by the `@base` directive at the top of your set definition.

Warning: Never reuse the SKOS or FoLiA Set Definition namespaces!

```
@base <http://your/namespace/> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix fsd: <http://folia.science.ru.nl/setdefinition#> .
```

SKOS uses a different terminology than we do, which may be the source of some confusion. We attempt to map the terms in the following table:

Our term	SKOS	SKOS class
Set/Subset ID	Collection Notation	`skos:Collection` `skos:notation`

After this preamble, we can define a set as follows:

```
<#your-set-id>
    a skos:Collection ;
    skos:notation   "your-set-id" ;
    skos:prefLabel  "Human readable label for your set" ;
    fsd:open        false .
```

The first two lines state that `http://your/namespace/#your-set-i` is *a*[1] SKOS Collection, which is what we use for FoLiA Sets. The `skos:notation` property corresponds to the ID of the Set, only one is allowed[2] .

[1] the *a* in Turtle syntax is shorthand for `rdf:type`
[2] Technically, SKOS allows multiple, but we restrict it for Set Definitions.

A set can be either open or closed (default), an open set allows any classes, even if they are not defined. This can be used for open vocabularies. The `fsd:open` property is used to indicate this, it is not part of SKOS but an extension of ours, hence the different namespace prefix.

References

3.2 Classes

A set (collection in SKOS terms) consists of classes (concepts in SKOS terms). Consider a simple part-of-speech set with three classes. First we define the set and refer to all the classes it contains:

```
<#simplepos>
    a skos:Collection ;
    skos:notation   "simplepos" ;
    skos:prefLabel "A simple part of speech set" ;
    skos:member <#N> , <#V> , <#A> .
```

Then we define the classes:

```
<#N>
    a skos:Concept ;
    skos:notation   "N" ;
    skos:prefLabel "Noun" .

<#V>
    a skos:Concept ;
    skos:notation   "V" ;
    skos:prefLabel "Verb" .

<#A>
    a skos:Concept ;
    skos:notation   "A" ;
    skos:prefLabel "Adjective" .
```

The **ID** (`skos:notation`) of the class is mandatory for FoLiA Set Definitions and determines a value the `class` attribute may take in the FoLiA document, for elements of this set. The `skos:prefLabel` property, both on the set itself as well as the classes, carries a human readable description for presentational purposes, this is optional but highly recommended.

In our legacy set definition format this is fairly straightforward and more concise:

```
<set
  xmlns="http://ilk.uvt.nl/folia"
  xml:id="simplepos" type="closed"
  label="Simple Part-of-Speech">
  <class xml:id="N" label="Noun" />
  <class xml:id="V" label="Verb" />
  <class xml:id="A" label="Adjective" />
</set>
```

3.3 Class Hierarchy

In FoLiA Set Definitions, classes can be nested to create more complex hierarchies or taxonomy trees, in which both nodes and leaves act as valid classes. This is best illustrated in our legacy XML format first. Consider the following set definition for named entities, in which the *location* class has been extended into more fine-grained subclasses.

```
<set xml:id="namedentities" type="closed" xmlns="http://ilk.uvt.nl/folia">
  <class xml:id="per" label="Person" />
  <class xml:id="org" label="Organisation" />
  <class xml:id="loc" label="Location">
    <class xml:id="loc.country" label="Country" />
    <class xml:id="loc.street" label="Street" />
    <class xml:id="loc.building" label="Building">
      <class xml:id="loc.building.hospital" label="Hospital" />
      <class xml:id="loc.building.church" label="Church" />
      <class xml:id="loc.building.station" label="Station" />
    </class>
  </class>
</set>
```

In the SKOS model, this is more verbose as the hierarchy has to be modelled explicitly using the `skos:broader` property, as shown in the following excerpt:

```
<#namedentities>
    a skos:Collection ;
    skos:member <#loc> , <#loc.country> .

<#loc>
    a skos:Concept ;
    skos:notation   "loc" ;
    skos:prefLabel "Location" .

<#loc.country>
    a skos:Concept ;
    skos:notation   "loc.country" ;
    skos:prefLabel "Country" ;
    skos:broader <#loc> .
```

It is recommended, but not mandatory, to set the class ID (`skos:notation`) of any nested classes to represent a full path, as a full path makes substring queries possible. FoLiA, however, does not dictate this and neither does it prescribe a delimiter for such paths, so the period in the above example (`loc.country`) is merely a convention. Each ID, however, does have to be unique in the entire set.

3.4 Subsets

3.4.1 Features

In addition to a main class, an arbitrary number of *features* can be added to *any* annotation element that takes a set. Each feature pertains to a specific *subset* in that set and assigns a *class* in the subset. The subsets and classes therein are defined in the set definition (See *Set Definitions (Vocabulary)*), so may be entirely user-defined.

The element `<feat>` is used to add features to any kind of annotation. In the following example we make use of a subset we invented which ties a lemma to its plural form. This is just an example, you can think of any subset you like and associate all kinds of information with it.

```
<lemma class="house">
  <feat subset="plural" class="houses" />
</lemma>
```

Note: Do make sure not to use features and create subsets if there is already a more appropriate FoLiA annotation available. For example; don't use a part-of-speech subset in a lemma set, because there is already *Part-of-Speech Annotation* for that.

A more thorough example for part-of-speech tags with features will be explained in the section on *Part-of-Speech Annotation*.

Some annotation types take *predefined subsets* because some features are very commonly used. These subsets have clearly defined semantics. However, it still depends on the set on whether these can be used, and which classes these take. Whenever subsets are predefined by FoLiA, they can be assigned directly using *XML attributes*. Consider the following example of lexical semantic sense annotation, in which subset `synset` is a predefined subset.

```
<sense class="X" synset="Y" />
```

This is semantically equivalent to:

```
<sense class="X">
    <feat subset="synset" class="Y" />
</sense>
```

The following example of event annotation with the feature with predefined subset `actor` is similar:

```
<event class="tweet" actor="John Doe">
 ...
</event>
```

```
<event class="tweet">
 <feat subset="actor" class="John Doe" />
 ...
</event>
```

Features can also be used to assign multiple classes within the same subset, which is impossible with main classes. In the following example the event is associated with a list of two actors. In this case the XML attribute shortcut no longer suffices, and the `<feat>` element must be used explicitly.

```
<event class="conversation">
 <feat subset="actor" class="John Doe" />
 <feat subset="actor" class="Jane Doe" />
 <p>...</p>
</event>
```

To recap: the `<feat>` element can always be used freely to associate *any* additional classes of *any* designed subset with *any* annotation element. For certain elements, there are predefined subsets, in which case you can assign them using the XML attribute shortcut. This, however, only applies to the predefined subsets.

Another elaborate example of features can be found in the section on *Part-of-Speech Annotation*.

The section on *Features* introduced subsets. Please ensure you are familiar with this notion before continuing with the current section.

Subset can be defined in a similar fashion to sets. Consider the legacy XML format first:

```
<set xml:id="simplepos" type="closed" xmlns="http://ilk.uvt.nl/folia">
  <class xml:id="N" label="Noun" />
  <class xml:id="V" label="Verb" />
  <class xml:id="A" label="Adjective" />
  <subset xml:id="gender" class="closed">
      <class xml:id="m" label="Masculine" />
      <class xml:id="f" label="Feminine" />
```

(continues on next page)

(continued from previous page)

```
      <class xml:id="n" label="Neuter" />
  </subset>
</set>
```

In RDF, subsets are defined as SKOS Collections, just like the primary set. The primary set refers to the subsets using the same `skos:member` relation as is used for classes/concepts.

```
<#simplepos>
    a skos:Collection ;
    skos:member <#N> , <#V> , <#A> , <#gender> .

<#gender>
    a skos:Collection ;
    skos:notation   "gender" ;
    skos:member <#gender.m> .

<#gender.m>
    a skos:Concept ;
    skos:notation   "m" ;
    skos:prefLabel "Location" .
```

Note that in this example, we prefixed the resource name for the class (`#gender.m` instead of `#m`). This is just a recommended convention as URIs have to be unique and we may want to re-use the `m` ID in other subsets as well. The ID in the `skos:notation` property does not need to carry this prefix, as it needs only be unique within the subset. This property always determines how it is referenced from the FoLiA document, so we would still get `<feat subset="gender" class="m" />`

3.5 Constraints

It is possible to define constriants on which subsets can be used with which classes and which classes within subsets can be combined, though SKOS has no mechanism to express such constraints. We introduce our own resources and properties to define to define constraints, in the namespace of our extension (`http://folia.science.ru.nl/setdefinition#`, with prefix `fsd:` in this documentation).

The core of the constraints is the `fsd:constrain` relation which can be made between any subset (`skos:Collection`) and class (`skos:Concept`). Consider the following Part-of-Speech tag example in which we constrain the subset *gender* to only occur with nouns:

```
<#simplepos>
     a skos:Collection ;
     skos:member <#N> .

example:N a skos:Concept ;
    skos:notation "N" ;
    skos:prefLabel "Noun" .

example:gender a skos:Collection ;
    skos:member example:masculine, example:feminine, example:neuter ;
    fsd:constrain example:N .
```

The same can be expressed in our legacy format as follows. Note that we left out the definition for the three genders in the RDF example for brevity.

```
<set xml:id="simplepos" type="closed" xmlns="http://ilk.uvt.nl/folia">
   <class xml:id="N" label="Noun" />
```

(continues on next page)

(continued from previous page)

```
   <subset xml:id="gender" type="closed">
     <class xml:id="masculine" label="masculine" />
     <class xml:id="feminine" label="feminine" />
     <class xml:id="neuter" label="neuter" />
     <constrain id="N" />
   </subset>
</set>
```

Multiple constrain relations may be specified, but one has to be aware that this then counts as a conjunction or intersection. What we often see instead when multiple relations is the use of a `fsd:Constraint` class, which acts as a collection of contrain relations and can explicitly express the *type* (`fsd:constraintType`) of matching to apply to the constraints. The type be any of the following:

- `"any"` - Only of of the constrain relations must match for the constraint to pass
- `"all"` - All constrain relations must match for the constraint to pass
- `"none"` - None of the constrain relations must match for the constraint to pass

The other main purpose of the `fsd:Constraint` class is to avoid repetition, as it allows a complex contraint to be referenced from multiple locations. Consider the following example, first in our legacy format:

```
<set xml:id="simplepos" type="closed" xmlns="http://ilk.uvt.nl/folia">
   <class xml:id="N" label="Noun" />
   <class xml:id="A" label="Adjective" />
   <class xml:id="V" label="Verb" />
   <subset xml:id="gender" type="closed">
     <class xml:id="masculine" label="masculine" />
     <class xml:id="feminine" label="feminine" />
     <class xml:id="neuter" label="neuter" />
     <constrain id="constraint.1" />
   </subset>
   <subset xml:id="case" type="closed">
     <class xml:id="nom" label="nominative" />
     <class xml:id="gen" label="genitive" />
     <class xml:id="dat" label="dative" />
     <class xml:id="acc" label="accusative" />
     <constrain id="constraint.1" />
   </subset>
   <constraint xml:id="constraint.1" type="any">
     <constrain id="N" />
     <constrain id="A" />
   </constraint>
</set>
```

In RDF, the constraint would be formulated as follows:

```
example:constraint.1 a fsd:Constraint ;
    fsd:constraintType "any" ;
    fsd:constrain example:N ;
    fsd:constrain example:A .
```

A `fsd::constrain` relation may be used within sets (`skos:Collection`), classes (`skos:Concept`) as well as constraints (`fsd:Constraint`). Similary, a `fsd:constrain` relation may point to either of the three. All this combined allows for complex nesting logic.

The following example shows a more complete set definition with various kinds of constraints, we show it both in legacy XML as well as turtle RDF:

```xml
<set xml:id="simplepos" type="closed" xmlns="http://ilk.uvt.nl/folia">
   <class xml:id="N" label="Noun">
     <constrain id="constraint.2" />
   </class>

   <class xml:id="A" label="Adjective">
     <constrain id="constraint.2" />
   </class>

   <class xml:id="V" label="Verb">
     <constrain id="tense" />
     <constrain id="number" />
   </class>

   <subset xml:id="gender" type="closed">
     <class xml:id="m" label="masculine" />
     <class xml:id="f" label="feminine" />
     <class xml:id="n" label="neuter" />
     <constrain id="constraint.1" />
   </subset>

   <subset xml:id="case" type="closed">
     <class xml:id="nom" label="nominative" />
     <class xml:id="gen" label="genitive" />
     <class xml:id="dat" label="dative" />
     <class xml:id="acc" label="accusative" />
     <constrain id="constraint.1" />
   </subset>

   <subset xml:id="number" type="closed">
     <class xml:id="s" label="singular" />
     <class xml:id="p" label="plural" />
   </subset>

   <subset xml:id="tense" type="closed">
     <class xml:id="present" label="present" />
     <class xml:id="past" label="past" />
     <constrain id="V" />
   </subset>

   <constraint xml:id="constraint.1" type="any">
     <!-- This is a constraint expressing which classes the subset using this constraint is valid -->
     <constrain id="N" />
     <constrain id="A" />
   </constraint>

   <constraint xml:id="constraint.2" type="all">
     <!-- This is a constraint expressing which subsets are required by the class using it-->
     <constrain id="gender" />
     <constrain id="case" />
     <constrain id="number" />
   </constraint>
</set>
```

```
@prefix fsd: <http://folia.science.ru.nl/setdefinition#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix simplepos: <http://folia.science.ru.nl/setdefinition/simplepos#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

simplepos:Set a skos:Collection ;
    skos:member simplepos:A,
        simplepos:N,
        simplepos:Subset.case,
        simplepos:Subset.gender,
        simplepos:Subset.number,
        simplepos:Subset.tense,
        simplepos:V ;
    skos:notation "simplepos" .

simplepos:Subset.tense a skos:Collection ;
    fsd:constrain simplepos:V ;
    skos:member simplepos:past,
        simplepos:present ;
    skos:notation "tense" .

simplepos:acc a skos:Concept ;
    fsd:sequenceNumber 4 ;
    skos:notation "acc" ;
    skos:prefLabel "accusative" .

simplepos:dat a skos:Concept ;
    fsd:sequenceNumber 3 ;
    skos:notation "dat" ;
    skos:prefLabel "dative" .

simplepos:f a skos:Concept ;
    fsd:sequenceNumber 2 ;
    skos:notation "f" ;
    skos:prefLabel "feminine" .

simplepos:gen a skos:Concept ;
    fsd:sequenceNumber 2 ;
    skos:notation "gen" ;
    skos:prefLabel "genitive" .

simplepos:m a skos:Concept ;
    fsd:sequenceNumber 1 ;
    skos:notation "m" ;
    skos:prefLabel "masculine" .

simplepos:n a skos:Concept ;
    fsd:sequenceNumber 3 ;
    skos:notation "n" ;
    skos:prefLabel "neuter" .

simplepos:nom a skos:Concept ;
    fsd:sequenceNumber 1 ;
    skos:notation "nom" ;
```

(continues on next page)

(continued from previous page)

```
    skos:prefLabel "nominative" .

simplepos:p a skos:Concept ;
    fsd:sequenceNumber 2 ;
    skos:notation "p" ;
    skos:prefLabel "plural" .

simplepos:past a skos:Concept ;
    fsd:sequenceNumber 2 ;
    skos:notation "past" ;
    skos:prefLabel "past" .

simplepos:present a skos:Concept ;
    fsd:sequenceNumber 1 ;
    skos:notation "present" ;
    skos:prefLabel "present" .

simplepos:s a skos:Concept ;
    fsd:sequenceNumber 1 ;
    skos:notation "s" ;
    skos:prefLabel "singular" .

simplepos:A a skos:Concept ;
    fsd:constrain simplepos:constraint.2 ;
    fsd:sequenceNumber 2 ;
    skos:notation "A" ;
    skos:prefLabel "Adjective" .

simplepos:N a skos:Concept ;
    fsd:constrain simplepos:constraint.2 ;
    fsd:sequenceNumber 1 ;
    skos:notation "N" ;
    skos:prefLabel "Noun" .

simplepos:Subset.case a skos:Collection ;
    fsd:constrain simplepos:constraint.1 ;
    skos:member simplepos:acc,
        simplepos:dat,
        simplepos:gen,
        simplepos:nom ;
    skos:notation "case" .

simplepos:Subset.gender a skos:Collection ;
    fsd:constrain simplepos:constraint.1 ;
    skos:member simplepos:f,
        simplepos:m,
        simplepos:n ;
    skos:notation "gender" .

simplepos:Subset.number a skos:Collection ;
    skos:member simplepos:p,
        simplepos:s ;
    skos:notation "number" .

simplepos:V a skos:Concept ;
    fsd:constrain simplepos:number,
```

(continues on next page)

(continued from previous page)

```
        simplepos:tense ;
    fsd:sequenceNumber 3 ;
    skos:notation "V" ;
    skos:prefLabel "Verb" .

simplepos:constraint.1 a fsd:Constraint ;
    fsd:constrain simplepos:A,
        simplepos:N ;
    fsd:constraintType "any" .

simplepos:constraint.2 a fsd:Constraint ;
    fsd:constrain simplepos:Subset.case,
        simplepos:Subset.gender,
        simplepos:Subset.number ;
    fsd:constraintType "all" .


```

3.6 SKOS

SKOS allows for more expressions to be made, and of course the full power of open linked data is available up to be used with FoLiA Set Definitions. The previous subsections layed out the minimal requirements for FoLiA Set Definitions using the SKOS model.

The use of `skos:OrderedCollection` is currently not supported yet, `skos:Collection` is mandatory. Ordering of classes (SKOS Concepts) can currently be indicated through a separate `fsd:sequenceNumber` property.

FoLiA Set Definitions must be *complete*, that is to say that all sets (SKOS collections) and classes (SKOS concepts) must be fully defined in one and the same set definition file.

Note: The file need not be static but can be dynamically generated server-side; which must be *publicly* available from a URL. A set definition must contain one and only one primary set (SKOS collection), all other sets must be subsets (SKOS collections that are a member of the primary set, no deeper nesting is supported).

See also:

- *Annotation Declarations*
- *Features*

CHAPTER 4

Annotation Types

FoLiA defines various XML elements to represent document structure and various annotations, we can divide these XML elements into several generic annotation groups. In each of these categories, FoLiA defines specific elements for specific annotation types. This is a deliberate limit on the extensibility of FoLiA in favour of specificity; i.e. you can't just add your own annotation type. If a particular annotation type is not properly accommodated yet, contact the FoLiA developers and we will see how we can extend FoLiA.

For good measure, we again emphasise that this is a limitation on annotation types only, not on the vocabulary the annotation types make use of, which is deliberately separated from the FoLiA standard itself. The next section will elaborate on this.

Below are the categories and underlying annotation types, you can click each for exhaustive information (but please finish this introductory chapter first):

- *Structure Annotation* – This category encompasses annotation types that define the structure of a document, e.g. paragraphs, sentences, words, sections like chapters, lists, tables, etc... These types are not strictly considered linguistic annotation and equivalents are also commonly found in other document formats such as HTML, TEI, MarkDown, LaTeX, and others. For FoLiA it provides the necessary structural basis that linguistic annotation can build on.
  - *Token Annotation* – `<w>` – This annotation type introduces a tokenisation layer for the document. The terms **token** and **word** are used interchangeably in FoLiA as FoLiA itself does not commit to a specific tokenisation paradigm. Tokenisation is a prerequisite for the majority of linguistic annotation types offered by FoLiA and it is one of the most fundamental types of Structure Annotation. The words/tokens are typically embedded in other types of structure elements, such as sentences or paragraphs.
  - *Division Annotation* – `<div>` – Structure annotation representing some kind of division, typically used for chapters, sections, subsections (up to the set definition). Divisions may be nested at will, and may include almost all kinds of other structure elements.
  - *Paragraph Annotation* – `<p>` – Represents a paragraph and holds further structure annotation such as sentences.
  - *Head Annotation* – `<head>` – The `head` element is used to provide a header or title for the structure element in which it is embedded, usually a division (`<div>`)
  - *List Annotation* – `<list>` – Structure annotation for enumeration/itemisation, e.g. bulleted lists.
  - *Figure Annotation* – `<figure>` – Structure annotation for including pictures, optionally captioned, in documents.

  - *Vertical Whitespace* – `<whitespace>` – Structure annotation introducing vertical whitespace
  - *Linebreak* – `<br>` – Structure annotation representing a single linebreak and with special facilities to denote pagebreaks.
  - *Sentence Annotation* – `<s>` – Structure annotation representing a sentence. Sentence detection is a common stage in NLP alongside tokenisation.
  - *Event Annotation* – `<event>` – Structural annotation type representing events, often used in new media contexts for things such as tweets, chat messages and forum posts (as defined by a user-defined set definition). Note that a more linguistic kind of event annotation can be accomplished with *Entity Annotation* or even *Time Segmentation* rather than this one.
  - *Quote Annotation* – `<quote>` – Structural annotation used to explicitly mark quoted speech, i.e. that what is reported to be said and appears in the text in some form of quotation marks.
  - *Note Annotation* – `<note>` – Structural annotation used for notes, such as footnotes or warnings or notice blocks.
  - *Reference Annotation* – `<ref>` – Structural annotation for referring to other annotation types. Used e.g. for referring to bibliography entries (citations) and footnotes.
  - *Table Annotation* – `<table>` – Structural annotation type for creating a simple tabular environment, i.e. a table with rows, columns and cells and an optional header.
  - *Part Annotation* – `<part>` – The structure element `part` is a fairly abstract structure element that should only be used when a more specific structure element is not available. Most notably, the part element should never be used for representation of morphemes or phonemes! Part can be used to divide a larger structure element, such as a division, or a paragraph into arbitrary subparts.
  - *Utterance Annotation* – `<utt>` – An utterance is a structure element that may consist of words or sentences, which in turn may contain words. The opposite is also true, a sentence may consist of multiple utterances. Utterances are often used in the absence of sentences in a speech context, where neat grammatical sentences can not always be distinguished.
  - *Entry Annotation* – `<entry>` – FoLiA has a set of structure elements that can be used to represent collections such as glossaries, dictionaries, thesauri, and wordnets. *Entry annotation* defines the entries in such collections, *Term annotation* defines the terms, and *Definition Annotation* provides the definitions.
  - *Term Annotation* – `<term>` – FoLiA has a set of structure elements that can be used to represent collections such as glossaries, dictionaries, thesauri, and wordnets. *Entry annotation* defines the entries in such collections, *Term annotation* defines the terms, and *Definition Annotation* provides the definitions.
  - *Definition Annotation* – `<def>` – FoLiA has a set of structure elements that can be used to represent collections such as glossaries, dictionaries, thesauri, and wordnets. *Entry annotation* defines the entries in such collections, *Term annotation* defines the terms, and *Definition Annotation* provides the definitions.
  - *Example Annotation* – `<ex>` – FoLiA has a set of structure elements that can be used to represent collections such as glossaries, dictionaries, thesauri, and wordnets. *Examples annotation* defines examples in such collections.
  - *Hidden Token Annotation* – `<hiddenw>` – This annotation type allows for a hidden token layer in the document. Hidden tokens are ignored for most intents and purposes but may serve a purpose when annotations on implicit tokens is required, for example as targets for syntactic movement annotation.

- *Content Annotation* – This category groups text content and phonetic content, the former being one of the most frequent elements in FoLiA and used to associate text (or a phonetic transcription) with a structural element.
  - *Text Annotation* – `<t>` – Text annotation associates actual textual content with structural elements, without it a document would be textless. FoLiA treats it as an annotation like any other.

   - *Phonetic Annotation/Content* – `<ph>` – This is the phonetic analogy to text content (`<t>`) and allows associating a phonetic transcription with any structural element, it is often used in a speech context. Note that for actual segmentation into phonemes, FoLiA has another related type: `Phonological Annotation`
   - *Raw Content* – `<content>` – This associates raw text content which can not carry any further annotation. It is used in the context of *Gap Annotation*

- *Inline Annotation* – This category encompasses (linguistic) annotation types describing a single structural element. Examples are Part-of-Speech Annotation or Lemmatisation, which often describe a single token.
   - *Part-of-Speech Annotation* – `<pos>` – Part-of-Speech Annotation, one of the most common types of linguistic annotation. Assigns a lexical class to words.
   - *Lemmatisation* – `<lemma>` – Lemma Annotation, one of the most common types of linguistic annotation. Represents the canonical form of a word.
   - *Domain/topic Annotation* – `<domain>` – Domain/topic Annotation. A form of inline annotation used to assign a certain domain or topic to a structure element.
   - *Sense Annotation* – `<sense>` – Sense Annotation allows to assign a lexical semantic sense to a word.
   - *Error Detection Annotation (DEPRECATED)* – `<errordetection>` – This annotation type is deprecated in favour of *Observation Annotation* and only exists for backward compatibility.
   - *Subjectivity Annotation (DEPRECATED)* – `<subjectivity>` – This annotation type is deprecated in favour of *Sentiment Annotation* and only exists for backward compatibility.
   - *Language Annotation* – `<lang>` – Language Annotation simply identifies the language a part of the text is in. Though this information is often part of the metadata, this form is considered an actual annotation.

- *Span Annotation* – This category encompasses (linguistic) annotation types that span one or more structural elements. Examples are (Named) Entities or Multi-word Expressions, Dependency Relations, and many others. FoLiA implements these as a stand-off layer that refers back to the structural elements (often words/tokens). The layer itself is embedded in a structural level of a wider scope (such as a sentence).
   - *Syntactic Annotation* – `<su>` – Assign grammatical categories to spans of words. Syntactic units are nestable and allow representation of complete syntax trees that are usually the result of consistuency parsing.
   - *Chunking* – `<chunk>` – Assigns shallow grammatical categories to spans of words. Unlike syntax annotation, chunks are not nestable. They are often produced by a process called Shallow Parsing, or alternatively, chunking.
   - *Entity Annotation* – `<entity>` – Entity annotation is a broad and common category in FoLiA. It is used for specifying all kinds of multi-word expressions, including but not limited to named entities. The set definition used determines the vocabulary and therefore the precise nature of the entity annotation.
   - *Dependency Annotation* – `<dependency>` – Dependency relations are syntactic relations between spans of tokens. A dependency relation takes a particular class and consists of a single head component and a single dependent component.
   - *Time Segmentation* – `<timesegment>` – FoLiA supports time segmentation to allow for more fine-grained control of timing information by associating spans of words/tokens with exact timestamps. It can provide a more linguistic alternative to *Event Annotation*.
   - *Coreference Annotation* – `<coreferencechain>` – Relations between words that refer to the same referent (anaphora) are expressed in FoLiA using Coreference Annotation. The co-reference relations are expressed by specifying the entire chain in which all links are coreferent.
   - *Semantic Role Annotation* – `<semrole>` – This span annotation type allows for the expression of semantic roles, or thematic roles. It is often used together with *Predicate Annotation*

   - *Predicate Annotation* – `<predicate>` – Allows annotation of predicates, this annotation type is usually used together with Semantic Role Annotation. The types of predicates are defined by a user-defined set definition.
   - *Observation Annotation* – `<observation>` – Observation annotation is used to make an observation pertaining to one or more word tokens. Observations offer a an external qualification on part of a text. The qualification is expressed by the class, in turn defined by a set. The precise semantics of the observation depends on the user-defined set.
   - *Sentiment Annotation* – `<sentiment>` – Sentiment analysis marks subjective information such as sentiments or attitudes expressed in text. The sentiments/attitudes are defined by a user-defined set definition.
   - *Statement Annotation* – `<statement>` – Statement annotation, sometimes also refered to as attribution, allows to decompose statements into the source of the statement, the content of the statement, and the way these relate, provided these are made explicit in the text.
   - *Modality Annotation* – `<modality>` – Modality annotation is used to describe the relationship between cue word(s) and the scope it covers. It is primarily used for the annotation of negation, but also for the annotation of factuality, certainty and truthfulness:.

- *Subtoken Annotation* – This category contains morphological annotation and phonological annotation, i.e. the segmentation of a word into morphemes and phonemes, and recursively so if desired. It is a special category that mixes characteristics from structure annotation (the `morpheme` and `phoneme` elements are very structure-like) and also from span annotation, as morphemes and phonemes are embedded in an annotation layer and refer back to the text/phonetic content they apply to. Like words/tokens, these elements may also be referenced from `wref` elements.
   - *Morphological Annotation* – `<morpheme>` – Morphological Annotation allows splitting a word/token into morphemes, morphemes itself may be nested. It is embedded within a layer `morphology` which can be embedded within word/tokens.
   - *Phonological Annotation* – `<phoneme>` – The smallest unit of annotatable speech in FoLiA is the phoneme level. The phoneme element is a form of structure annotation used for phonemes. Alike to morphology, it is embedded within a layer `phonology` which can be embedded within word/tokens.

- *Text Markup Annotation* – The text content element (`<t>`) allows within its scope elements of a this category; these are **Text Markup** elements, they always contain textual content and apply a certain markup to certain spans of the text. One of it's common uses is for styling (emphasis, underlines, etc.). Text markup elements may be nested.
   - *Style Annotation* – `<t-style>` – This is a text markup annotation type for applying styling to text. The actual styling is defined by the user-defined set definition and can for example included classes such as italics, bold, underline
   - *Hyphenation* – `<t-hbr>` – This is a text-markup annotation form that indicates where in the original text a linebreak was inserted and a word was hyphenised.
   - *Horizontal Whitespace* – `<t-hspace>` – Markup annotation introducing horizontal whitespace

- *Higher-order Annotation* – Higher-order Annotation groups a very diverse set of annotation types that are considered *annotations on annotations*
   - *Correction Annotation* – `<correction>` – Corrections are one of the most complex annotation types in FoLiA. Corrections can be applied not just over text, but over any type of structure annotation, inline annotation or span annotation. Corrections explicitly preserve the original, and recursively so if corrections are done over other corrections.
   - *Gap Annotation* – `<gap>` – Sometimes there are parts of a document you want to skip and not annotate at all, but include as is. This is where gap annotation comes in, the user-defined set may indicate the kind of gap. Common omissions in books are for example front-matter and back-matter, i.e. the cover.
   - *Relation Annotation* – `<relation>` – FoLiA provides a facility to relate arbitrary parts of your document with other parts of your document, or even with parts of other FoLiA documents or

external resources, even in other formats. It thus allows linking resources together. Within this context, the `xref` element is used to refer to the linked FoLiA elements.

  - *Span Relation Annotation* – `<spanrelation>` – Span relations are a stand-off extension of relation annotation that allows for more complex relations, such as word alignments that include many-to-one, one-to-many or many-to-many alignments. One of its uses is in the alignment of multiple translations of (parts) of a text.
  - *Metric Annotation* – `<metric>` – Metric Annotation is a form of higher-order annotation that allows annotation of some kind of measurement. The type of measurement is defined by the class, which in turn is defined by the set as always. The metric element has a `value` attribute that stores the actual measurement, the value is often numeric but this needs not be the case.
  - *String Annotation* – `<str>` – This is a form of higher-order annotation for selecting an arbitrary substring of a text, even untokenised, and allows further forms of higher-order annotation on the substring. It is also tied to a form of text markup annotation.
  - *Alternative Annotation* – `<alt>` – This form of higher-order annotation encapsulates alternative annotations, i.e. annotations that are posed as an alternative option rather than the authoratitive chosen annotation
  - *Comment Annotation* – `<comment>` – This is a form of higher-order annotation that allows you to associate comments with almost all other annotation elements
  - *Description Annotation* – `<desc>` – This is a form of higher-order annotation that allows you to associate descriptions with almost all other annotation elements
  - *External Annotation* – `<external>` – External annotation makes a reference to an external FoLiA document whose structure is inserted at the exact place the external element occurs.

4.1 Content Annotation

This category groups text content and phonetic content, the former being one of the most frequent elements in FoLiA and used to associate text (or a phonetic transcription) with a structural element.

FoLiA defines the following types of content annotation:

- *Content Annotation* – This category groups text content and phonetic content, the former being one of the most frequent elements in FoLiA and used to associate text (or a phonetic transcription) with a structural element.
  - *Text Annotation* – `<t>` – Text annotation associates actual textual content with structural elements, without it a document would be textless. FoLiA treats it as an annotation like any other.
  - *Phonetic Annotation/Content* – `<ph>` – This is the phonetic analogy to text content (`<t>`) and allows associating a phonetic transcription with any structural element, it is often used in a speech context. Note that for actual segmentation into phonemes, FoLiA has another related type: `Phonological Annotation`
  - *Raw Content* – `<content>` – This associates raw text content which can not carry any further annotation. It is used in the context of *Gap Annotation*

4.1.1 Text Annotation

Text annotation associates actual textual content with structural elements, without it a document would be textless. FoLiA treats it as an annotation like any other.

Specification

Annotation Category *Content Annotation*

Declaration `<text-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since the beginning, revised since v0.6

Element `<t>`

API Class `TextContent` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.
- `xlink:href` – Turns this element into a hyperlink to the specified URL
- `xlink:type` – The type of link (you'll want to use `simple` in almost all cases).

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<br>` (*Linebreak*)

Valid Context `<current>` (*Correction Annotation*), `<def>` (*Definition Annotation*), `<div>` (*Division Annotation*), `<entry>` (*Entry Annotation*), `<event>` (*Event Annotation*), `<ex>` (*Example Annotation*), `<figure>` (*Figure Annotation*), `<head>` (*Head Annotation*), `<hiddenw>` (*Hidden Token Annotation*), `<list>` (*List Annotation*), `<morpheme>` (*Morphological Annotation*), `<new>` (*Correction Annotation*), `<note>` (*Note Annotation*), `<original>` (*Correction Annotation*), `<p>` (*Paragraph Annotation*), `<part>` (*Part Annotation*), `<phoneme>` (*Phonological Annotation*), `<quote>` (*Quote Annotation*), `<ref>` (*Reference Annotation*), `<s>` (*Sentence Annotation*), `<str>` (*String Annotation*), `<suggestion>` (*Correction Annotation*), `<term>` (*Term Annotation*), `<utt>` (*Utterance Annotation*), `<w>` (*Token Annotation*)

Explanation

Text is considered an annotation like any other rather than a given in FoLiA, but it is ubiquitous in almost all FoLiA documents, as a document without text is a rare occurrence. Text content is always represented by the `<t>` element and can be associated with *Structure Annotation* and *Subtoken Annotation*. Consider text associated with a words in a sentence:

```
<s xml:id="s.1">
    <w xml:id="s.1.w.1">
        <t>Hello</t>
    </w>
    <w xml:id="s.1.w.2">
        <t>world</t>
    </w>
</s>
```

FoLiA is not just a format for holding tokenised text, although tokenisation is a prerequisite for most all kinds of linguistic annotation. We can associate text content with a sentence as such:

```
<s xml:id="s.1">
    <t>Hello world</t>
</s>
```

Untokenised FoLiA documents with text on higher structural levels are in fact common input to FoLiA-aware tokenisers.

As FoLiA's representation of structure is hierarchical, you can nest various structure elements, but at the same time you can also associate text with structure elements on different levels, so specifying text on *both* the sentence and word level is valid too:

```
<s xml:id="s.1">
    <t>Hello world</t>
    <w xml:id="s.1.w.1">
        <t>Hello</t>
    </w>
    <w xml:id="s.1.w.2">
        <t>world</t>
    </w>
</s>
```

We call the association of text content on multiple structural levels **text redundancy**, it has its uses in preserving the untokenised original text, and facilating the job for parsers and tools.

If this kind of redundancy is used (it is not mandatory!), you may optionally point back to the text content of its parent structure element by specifying the `offset` attribute:

```
<p xml:id="example.p.1">
   <t>This is a paragraph containing only one sentence.</t>
   <s xml:id="example.p.1.s.1">
       <t offset="0">This is a paragraph containing only one sentence.</t>
       <w xml:id="example.p.1.s.1.w.1">
               <t offset="0">This</t>
       </w>
       <w xml:id="example.p.1.s.1.w.2">
               <t offset="5">is</t>
       </w>
       ...
       <w xml:id="example.p.1.s.1.w.8" space="no">
               <t offset="40">sentence</t>
```

(continues on next page)

(continued from previous page)

```
        </w>
        <w xml:id="example.p.1.s.1.w.9">
                <t offset="48">.</t>
        </w>
    </s>
</p>
```

Note: Offsets in FoLiA are always zero indexed (i.e, the first offset is zero, not one) and count unicode codepoints (as opposed to bytes). Offsets always refer to a specific **‘normalized form <http://www.unicode.org/reports/tr15/‘_** of the text: Unicode Normal Form Composed (NFC). This affects how certain characters (notably those with diacritics) are encoded. FoLiA libraries should take care of this for you automatically.

Offsets can be used to refer back from deeper text-content elements. This does imply that there are some challenges to solve: First of all, by default, the offset refers to the first structural parent of whatever text-supporting element the text content (`<t>`) is a member of. If a level is missing we have to explicitly specify this reference using the `ref` attribute. We show this in the following example, where there is no text content for the sentence, and we refer directly to the paragraph’s text:

```
<p xml:id="example.p.1">
   <t>Hello. This is a sentence. Bye!</t>
   <s xml:id="example.p.1.s.1">
       <w xml:id="example.p.1.s.1.w.1">
        <t ref="example.p.1" offset="7">This</t>
       </w>
       <w xml:id="example.p.1.s.1.w.2">
        <t ref="example.p.1" offset="12">is</t>
       </w>
       <w xml:id="example.p.1.s.1.w.3">
        <t ref="example.p.1" offset="15">a</t>
       </w>
       <w xml:id="example.p.1.s.1.w.4" space="no">
        <t ref="example.p.1" offset="17">sentence</t>
       </w>
       <w xml:id="example.p.1.s.1.w.5">
        <t ref="example.p.1" offset="25">.</t>
       </w>
   </s>
</p>
```

Text content is by default expected to be untokenised for higher-level structure; in `w` structure elements it by definition is tokenised, as that is precisely what provides the tokenisation layer. Text content elements may *never* be empty nor contain only whitespace or non-printable characters, in such circumstances you simply omit the text-content element altogether.

The notion of text redundancy can be useful but also creates room for error, the text on a higher level may not correspond with the text on a deeper level, as in the following *erroneous example*:

```
<s xml:id="s.1">
    <t>Goodbye world</t>
    <w xml:id="s.1.w.1">
        <t>Hello</t>
    </w>
    <w xml:id="s.1.w.2">
        <t>world</t>
```

(continues on next page)

(continued from previous page)

```
    </w>
</s>
```

FoLiA validators (since version 1.5) will not accept this and produce a *text consistency error*, so this is invalid FoLiA and should be rejected. Similar text consistency errors occur if you specify offsets that are incorrect.

Whitespace

Leading and trailing whitespace within a text content element is not significant (since version 2.4.1 but with backward effect). Double whitespace is collapsed to a single. As whitespace we consider spaces, tabs, newlines and carriage returns, so all of the following snippets have the identical text `to be or not to be` and the offset for `To` is 0:

```
<t>To be or not to be</t>

<t> To be or not to be</t>

<t>     To be or not to be</t>

<t>To be or not to be </t>

<t>
 To be or not to be</t>

<t>
 To be     or not to be</t>

<t>To be
   or not to be</t>

<t>
 To
 be
 or
 not
 to
 be</t>
```

This same principle applies to *Text Markup Annotation*, the following three are semantically identical:

```
<t>To <t-style class="bold">be</t-style> or not to be</t>

<t>To <t-style class="bold"> be </t-style> or not to be</t>

<t>
   To
   <t-style class="bold">be</t-style>
   or not to be
</t>
```

If you want to encode linebreaks, you need to explicitly use *Linebreak* (`<br/>`), as otherwise it will not be significant:

```
<t>To be<br/>
   or not to be</t>
```

Whitespace before explicit linebreaks is insignificant (since FoLiA v2.5.1), so the following two examples are identical to the one above:

```
<t>To be  <br/>
   or not to be</t>
```

```
<t>
   To be
   <br/>
   or not to be
</t>
```

As mentioned before, empty text is explicitly forbidden in FoLiA. All of the following are identical semantically, and all will produce an empty text error:

```
<t></t>

<t/>

<t>   </t>

<t>
</t>
```

The rule here is, empty text is no text at all, so you should omit the `<t>` element entirely in such cases.

Note: The rules regarding whitespace prior to FoLiA v2.5 and v2.4.1 were different and not as well-defined yet.

- prior to FoLiA v2.4.1 all whitespace and linebreaks were interpreted as significant
- since FoLiA v2.4.1 leading and trailing whitespace was stripped, but not all whitespace was collapsed yet.

FoLiA validators will be forgiving when checking the text consistency and offsets in older FoLiA documents. The new rules will be applied first, but fallbacks wil test again older rules in such cases, retaining backward compatibility.

Note: FoLiA (since v2.5) and TEI are comparable in the way they treat XML whitespace. TEI has an elaborate article on the subject that may provide further insight.

Preserving whitespace (advanced)

What if you **DO** explicitly want to encode a double space, an initial space or a trailing space? Though generally not recommended, this may be needed if you want to stay true to the untokenised original in a very strict sense. The You can set the `xml:space="preserve"` attribute on any text content or text markup element to indicate that you want to preserve the spaces as-is. Consider the following distinct examples:

```
<t>To be or not to be</t>

<t xml:space="preserve">To be     or not to be</t>
```

Without `xml:space="preserve"`, the texts would be identical. This attribute is automatically inherited by child elements, you will need to set `xml:space="default"` if you want to revert to the normal behaviour when nesting text markup.

Note that even when preserving spaces, FoLiA does not accept empty (whitespace-only) text nodes.

Instead of using `xml:space="preserve"`, you are encouraged to use the more explicit *Horizontal Whitespace* using the `<t-hspace/>` element:

```
<t>To be<t-hspace class="long" />or not to be</t>
```

Note: FoLiA does not accept XML CDATA in text content or text markup elements. It will be treated as it if were normal text. CDATA only makes sense when used with *Gap Annotation*.

Text classes (advanced)

It is possible to associate **multiple text content elements** with the same structural element, and thus associating multiple texts with the same element. You may wonder what could possibly be the point of such extra complexity. But there is a clear use case when dealing with for example corrections, or wanting to associate the text version just after a processing step such as Optical Character Recognition or any another kind of normalisation.

Text annotation, like most forms of annotations in FoLiA, is bound to the same paradigm of sets and classes. You can assign a `class` to your text content. And FoLiA allows you to associate multiple text content elements of different classes in the same structural element. Text content that has no explicitly associated class obtains the `current` class by default and is the only situation in which FoLiA actually predefines a class for a set. We call it `current` because it is considered the most current and up-to-date text layer, and the default unless explicitly specified otherwise. We allow you to omit it as it is so common and for most FoLiA documents you will not make use of multiple text classes and only use a single one.

Like all annotations, text annotation needs to be explicitly declared, declaring a `set` is only needed if you assign custom classes, otherwise a built-in set that defines `current` will be used automatically.

Orthographical corrections (see also *Correction Annotation*) are challenging because they can be applied to text content and thus change the text. Corrections are often applied on the token level, but you may want them propagated to the text content of sentences or paragraphs whilst at the same time wanting to retain the text how it originally was. This can be accomplished by introducing text content of a different class.

Below is an example illustrating the usage of multiple classes, three to be precise: the default `current` class showing the normal text, an `original` class showing text prior to correction, and a `ocroutput` class showing the text as produced by an OCR engine. To show the flexibility, offsets are added, but these are of course always optional. Note that when an offset is specified, it always refers to a text-content element of the same class! We first give an example where the correction is implicit:

```
<p xml:id="example.p.1">
   <t>Hello. This is a sentence. Bye!</t>
   <t class="original">Hello. This iz a sentence. Bye!</t>
   <t class="ocroutput">Hell0 Th1s iz a sentence, Bye1</t>
   <s xml:id="example.p.1.s.1">
       <t offset="7">This is a sentence.</t>
       <t class="original" offset="7">This is a sentence.</t>
       <t class="ocroutput" offset="6">Th1s iz a sentence,</t>
       <w xml:id="example.p.1.s.1.w.1">
        <t offset="0">This</t>
        <t class="ocroutput" offset="0">Th1s</t>
       </w>
       <w xml:id="example.p.1.s.1.w.2">
          <t offset="5">is</t>
          <t offset="5" class="original">iz</t>
          <t offset="5" class="ocroutput">iz</t>
       </w>
       <w xml:id="example.p.1.s.1.w.3">
        <t offset="8">a</t>
```

(continues on next page)

(continued from previous page)

```
        <t offset="8" class="original">a</t>
        <t offset="8" class="ocroutput">a</t>
       </w>
       <w xml:id="example.p.1.s.1.w.4" space="no">
        <t offset="10">sentence</t>
       </w>
       <w xml:id="example.p.1.s.1.w.5">
        <t offset="48">.</t>
        <t offset="48" class="original">.</t>
        <t offset="48" class="ocroutput">,</t>
       </w>
   </s>
</p>
```

Next, we give an example in which the correction is explicit, making use of *Correction Annotation*, which is one of the most complex annotation types in FoLiA. We leave out the ocr text class:

```
<p xml:id="example.p.1">
  <t>Hello. This is a sentence. Bye!</t>
  <t class="original">Hello. This iz a sentence. Bye!</t>
  <s xml:id="example.p.1.s.1">
    <t offset="7">This is a sentence.</t>
    <t class="original" offset="7">This is a sentence.</t>
    <w xml:id="example.p.1.s.1.w.1">
      <t offset="0">This</t>
    </w>
    <w xml:id="example.p.1.s.1.w.2">
      <correction>
      <new>
        <t offset="5">is</t>
      </new>
      <original>
        <t offset="5" class="original">iz</t>
      </original>
      </correction>
    </w>
    <w xml:id="example.p.1.s.1.w.3">
      <t offset="8">a</t>
    </w>
    <w xml:id="example.p.1.s.1.w.4" space="no">
      <t offset="10">sentence</t>
    </w>
    <w xml:id="example.p.1.s.1.w.5">
      <t offset="48">.</t>
    </w>
  </s>
</p>
```

See also:

- *Correction Annotation*
- *String Annotation*

Text class attribute (advanced)

So as we have just seen, FoLiA allows for multiple text content elements on the same structural elements, these other text content elements must carry a different class. This indicates an alternative text for the same element and is used for instance for pre-OCR vs. post-OCR or pre-normalisation vs. post-normalisation distinctions, or for transliterations.

When adding linguistic annotations on a structure element that has multiple text representations, it may be desirable to explicitly state which text class was used in establishing the annotation. This is done with the `textclass` attribute on any token or span annotation element. By default, this attribute is omitted, which implies it points to the default `current` text class.

Consider the following Part-of-Speech and lemma annotation on a word with two text classes, one representing the spelling as it occurs in the document, and one representing a more contemporary spelling. The following example makes it explicit that the PoS and lemma annotations are based on the latter text class.

```
<w class="WORD" xml:id="s.1.w.3">
     <t>aengename</t>
     <t class="contemporary">aangename</t>
     <pos class="ADJ" textclass="contemporary" />
     <lemma class="aangenaam" textclass="contemporary" />
</w>
```

Note that if you want to add another PoS annotation or lemma that is derived from another textclass, you will need to add those as an *alternative* (See *Alternative Annotation*), as the usual restrictions apply, there can be only one of each of a given set.

For span annotation, you can apply the `textclass` attribute in a similar fashion:

```
<entities>
  <entity class="per" textclass="contemporary">
    <wref id="s.1.w.5" t="John"/>
    <wref id="s.1.w.6" t="Doe"/>
  </entity>
</entities>
```

4.1.2 Phonetic Annotation/Content

This is the phonetic analogy to text content (`<t>`) and allows associating a phonetic transcription with any structural element, it is often used in a speech context. Note that for actual segmentation into phonemes, FoLiA has another related type: `Phonological Annotation`

Specification

Annotation Category *Content Annotation*

Declaration `<phon-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since v0.12

Element `<ph>`

API Class `PhonContent` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*)

Valid Context `<current>` (*Correction Annotation*), `<def>` (*Definition Annotation*), `<div>` (*Division Annotation*), `<event>` (*Event Annotation*), `<ex>` (*Example Annotation*), `<head>` (*Head Annotation*), `<hiddenw>` (*Hidden Token Annotation*), `<list>` (*List Annotation*), `<morpheme>` (*Morphological Annotation*), `<new>` (*Correction Annotation*), `<note>` (*Note Annotation*), `<original>` (*Correction Annotation*), `<p>` (*Paragraph Annotation*), `<part>` (*Part Annotation*), `<phoneme>` (*Phonological Annotation*), `<ref>` (*Reference Annotation*), `<s>` (*Sentence Annotation*), `<str>` (*String Annotation*), `<suggestion>` (*Correction Annotation*), `<term>` (*Term Annotation*), `<utt>` (*Utterance Annotation*), `<w>` (*Token Annotation*)

Explanation

Written text is always contained in the text content element (`<t>`, see *Text Annotation*), for phonology there is a similar counterpart that behaves almost identically: `<ph>`. This element holds a phonetic or phonological transcription. It is used in a very similar fashion:

```
<utt src="helloworld.mp3"  begintime="..." endtime="...">
    <ph>hel o  w ld</ph>
    <w xml:id="example.utt.1.w.1" begintime="..." endtime="...">
        <ph>hel o </ph>
    </w>
    <w xml:id="example.utt.1.w.2" begintime="..." endtime="...">
        <ph>w ld</ph>
    </w>
</utt>
```

Like the *Text Annotation*, the `<ph>` element supports the `offset` attribute, referring to the offset in the phonetic transcription for the parent structure. The first index being zero. It also support multiple classes (analogous to text classes), the implicit default and *predefined* class being `current`. You could imagine using this for different notation systems (IPA , SAMPA, pinyin, etc...).

Phonetic transcription and text content can also go together without problem:

```
<utt>
    <ph>hel o  w ld</ph>
    <t>hello world</t>
    <w xml:id="example.utt.1.w.1">
        <ph offset="0">hel o </ph>
        <t offset="0">hello</t>
    </w>
    <w xml:id="example.utt.1.w.2">
        <ph offset="8">w ld</ph>
        <t offset="6">world</t>
    </w>
</utt>
```

Note: You should still use the normal *Text Annotation* for a normal textual transcription of the speech. This annotation type is reserved for phonetic/phonological transcriptions.

See also:

If you want to actually do segmentation into *phonemes*, see *Phonological Annotation*.

Example

A simple example document:

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <phon-annotation>
                         <annotator processor="p1" />
          </phon-annotation>
          <utterance-annotation>
                         <annotator processor="p1" />
          </utterance-annotation>
          <token-annotation>
                         <annotator processor="p1" />
          </token-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <speech xml:id="example.speech">
    <utt xml:id="example.utt.1" src="helloworld.mp3"  begintime="00:00:01.000" endtime="00:00:02.000">
        <ph>hel o  w ld</ph>
        <w xml:id="example.utt.1.w.1" begintime="00:00:00.000" endtime="00:00:01.000">
            <ph>hel o </ph>
        </w>
```

(continues on next page)

(continued from previous page)

```
        <w xml:id="example.utt.1.w.2" begintime="00:00:01.000" endtime="00:00:02.000">
            <ph>w ld</ph>
        </w>
    </utt>
  </speech>
</FoLiA>
```

4.1.3 Raw Content

This associates raw text content which can not carry any further annotation. It is used in the context of *Gap Annotation*

Specification

Annotation Category *Content Annotation*

Declaration `<rawcontent-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since the beginning, but revised and made a proper annotation type in v2.0

Element `<content>`

API Class `Content` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data <comment> (*Comment Annotation*), <desc> (*Description Annotation*)

Valid Context <gap> (*Gap Annotation*)

Explanation

The content element associates raw text content with an element, it is specifically used in the context of *Gap Annotation*. The content can carry no further annotations.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <division-annotation set="https://raw.githubusercontent.com/
↪LanguageMachines/uctodata/master/setdefinitions/divisions.foliaset.xml">
                         <annotator processor="p1" />
                  </division-annotation>
          <gap-annotation set="adhoc">
                         <annotator processor="p1" />
                  </gap-annotation>
          <rawcontent-annotation>
                         <annotator processor="p1" />
                  </rawcontent-annotation>
          <description-annotation>
                         <annotator processor="p1" />
                  </description-annotation>
          <paragraph-annotation>
                         <annotator processor="p1" />
                  </paragraph-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
     <gap class="frontmatter">
        <desc>This is the cover of the book</desc>
        <content>
<![CDATA[

            SHOW WHITE AND THE SEVEN DWARFS


                by the Brothers Grimm

                    first edition


            Copyright(c) blah blah
]]>
        </content>
```

(continues on next page)

(continued from previous page)

```
    </gap>
    <div xml:id="example.div.1" class="chapter" n="1">
        <t>In the <t-gap class="illegible" /> there was a princess...</t>
    </div>
  </text>
</FoLiA>
```

4.2 Higher-order Annotation

Higher-order Annotation groups a very diverse set of annotation types that are considered *annotations on annotations*

FoLiA defines the following types of higher-order annotation:

- *Higher-order Annotation* – Higher-order Annotation groups a very diverse set of annotation types that are considered *annotations on annotations*
  - *Correction Annotation* – `<correction>` – Corrections are one of the most complex annotation types in FoLiA. Corrections can be applied not just over text, but over any type of structure annotation, inline annotation or span annotation. Corrections explicitly preserve the original, and recursively so if corrections are done over other corrections.
  - *Gap Annotation* – `<gap>` – Sometimes there are parts of a document you want to skip and not annotate at all, but include as is. This is where gap annotation comes in, the user-defined set may indicate the kind of gap. Common omissions in books are for example front-matter and back-matter, i.e. the cover.
  - *Relation Annotation* – `<relation>` – FoLiA provides a facility to relate arbitrary parts of your document with other parts of your document, or even with parts of other FoLiA documents or external resources, even in other formats. It thus allows linking resources together. Within this context, the `xref` element is used to refer to the linked FoLiA elements.
  - *Span Relation Annotation* – `<spanrelation>` – Span relations are a stand-off extension of relation annotation that allows for more complex relations, such as word alignments that include many-to-one, one-to-many or many-to-many alignments. One of its uses is in the alignment of multiple translations of (parts) of a text.
  - *Metric Annotation* – `<metric>` – Metric Annotation is a form of higher-order annotation that allows annotation of some kind of measurement. The type of measurement is defined by the class, which in turn is defined by the set as always. The metric element has a `value` attribute that stores the actual measurement, the value is often numeric but this needs not be the case.
  - *String Annotation* – `<str>` – This is a form of higher-order annotation for selecting an arbitrary substring of a text, even untokenised, and allows further forms of higher-order annotation on the substring. It is also tied to a form of text markup annotation.
  - *Alternative Annotation* – `<alt>` – This form of higher-order annotation encapsulates alternative annotations, i.e. annotations that are posed as an alternative option rather than the authoratitive chosen annotation
  - *Comment Annotation* – `<comment>` – This is a form of higher-order annotation that allows you to associate comments with almost all other annotation elements
  - *Description Annotation* – `<desc>` – This is a form of higher-order annotation that allows you to associate descriptions with almost all other annotation elements
  - *External Annotation* – `<external>` – External annotation makes a reference to an external FoLiA document whose structure is inserted at the exact place the external element occurs.

4.2.1 Correction Annotation

Corrections are one of the most complex annotation types in FoLiA. Corrections can be applied not just over text, but over any type of structure annotation, inline annotation or span annotation. Corrections explicitly preserve the original, and recursively so if corrections are done over other corrections.

Specification

Annotation Category *Higher-order Annotation*

Declaration `<correction-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since v0.4

Element `<correction>`

API Class `Correction` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.

- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<current>` (*Correction Annotation*), `<desc>` (*Description Annotation*), `<errordetection>` (*Error Detection Annotation (DEPRECATED)*), `<metric>` (*Metric Annotation*), `<new>` (*Correction Annotation*), `<original>` (*Correction Annotation*), `<suggestion>` (*Correction Annotation*)

Valid Context `<alt>` (*Alternative Annotation*), `<chunking>` (*Chunking*), `<coreferences>` (*Coreference Annotation*), `<current>` (*Correction Annotation*), `<def>` (*Definition Annotation*), `<dependencies>` (*Dependency Annotation*), `<div>` (*Division Annotation*), `<entities>` (*Entity Annotation*), `<entry>` (*Entry Annotation*), `<event>` (*Event Annotation*), `<ex>` (*Example Annotation*), `<figure>` (*Figure Annotation*), `<head>` (*Head Annotation*), `<hiddenw>` (*Hidden Token Annotation*), `<br>` (*Linebreak*), `<list>` (*List Annotation*), `<modalities>` (*Modality Annotation*), `<morpheme>` (*Morphological Annotation*), `<morphology>` (*Morphological Annotation*), `<new>` (*Correction Annotation*), `<note>` (*Note Annotation*), `<observations>` (*Observation Annotation*), `<original>` (*Correction Annotation*), `<p>` (*Paragraph Annotation*), `<part>` (*Part Annotation*), `<phoneme>` (*Phonological Annotation*), `<phonology>` (*Phonological Annotation*), `<quote>` (*Quote Annotation*), `<ref>` (*Reference Annotation*), `<semroles>` (*Semantic Role Annotation*), `<s>` (*Sentence Annotation*), `<sentiments>` (*Sentiment Annotation*), `<spanrelations>` (*Span Relation Annotation*), `<statements>` (*Statement Annotation*), `<str>` (*String Annotation*), `<suggestion>` (*Correction Annotation*), `<syntax>` (*Syntactic Annotation*), `<table>` (*Table Annotation*), `<term>` (*Term Annotation*), `<timing>` (*Time Segmentation*), `<utt>` (*Utterance Annotation*), `<whitespace>` (*Vertical Whitespace*), `<w>` (*Token Annotation*)

Explanation & Examples

Correction annotation is arguably one of the most complex annotation forms in FoLiA. It is a form of *Higher-order Annotation* which allows to annotate corrections on many types of annotation, including correction of text (i.e. spelling correction), of *Inline Annotation*, *Span Annotation* and even over *Structure Annotation*.

All corrections are annotated using the `<correction>` element. The following example shows a spelling correction of the misspelled word *treee* to its corrected form *tree*.

```xml
<w xml:id="example.p.1.s.1.w.1">
    <correction xml:id="TEST-000000001.p.1.s.1.w.1.c.1"
     class="spelling">
        <new>
            <t>tree</t>
        </new>
        <original>
            <t>treee</t>
        </original>
    </correction>
</w>
```

The class indicates the kind of correction, according to a user-defined set definition (see *Set Definitions (Vocabulary)*). The `<new>` element holds the actual content of the correction. The `<original>` element holds the content prior to correction. In this example, what we are correcting is the actual textual content (*Text Annotation*, `<t>`).

Corrections can be nested and we want to retain a full back-log. The following example illustrates the word *treee* that has been first mis-corrected to *three* and subsequently corrected again to *tree*:

```xml
<?xml version='1.0' encoding='utf-8'?>
<FoLiA xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="http://ilk.uvt.nl/folia"␣
↪xml:id="page1263" version="2.0.0">
  <metadata type="native">
    <annotations>
      <token-annotation>
          <annotator processor="p1" />
      </token-annotation>
      <sentence-annotation>
          <annotator processor="p1" />
      </sentence-annotation>
      <text-annotation>
          <annotator processor="p1" />
      </text-annotation>
      <correction-annotation set="https://raw.githubusercontent.com/proycon/folia/
↪master/setdefinitions/spellingcorrection.foliaset.xml">
          <annotator processor="johndoe" />
          <annotator processor="janedoe" />
      </correction-annotation>
    </annotations>
    <provenance>
       <processor xml:id="p1" name="proycon" type="manual" />
       <processor xml:id="johndoe" name="johndoe" type="manual" />
       <processor xml:id="janedoe" name="janedoe" type="manual" />
    </provenance>
  </metadata>
  <text xml:id="example.text">
    <s xml:id="example.s.1">
        <t>Watch that tree</t>
        <w xml:id="example.s.1.w.1">
            <t>Watch</t>
        </w>
        <w xml:id="example.s.1.w.2">
            <t>that</t>
        </w>
        <w xml:id="example.s.1.w.3">
          <correction xml:id="example.correction.2" class="spelling"
            processor="janedoe" confidence="1.0">
              <new>
                  <t>tree</t>
              </new>
              <original>
                <correction xml:id="example.correction.1"
                  class="spelling" processor="johndoe" confidence="0.6">
                  <new>
                      <t>three</t>
                  </new>
                  <original>
                      <t>treee</t>
                  </original>
```

(continues on next page)

(continued from previous page)

```
                </correction>
              </original>
          </correction>
        </w>
    </s>
  </text>
</FoLiA>
```

In the examples above what we corrected was the actual textual content (<t>). However, it is also possible to correct other annotations in exactly the same way. The next example corrects a part-of-speech tag:

```
<?xml version='1.0' encoding='utf-8'?>
<FoLiA xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="http://ilk.uvt.nl/folia"␣
↪xml:id="page1263" version="2.0.0">
  <metadata type="native">
    <annotations>
      <token-annotation>
          <annotator processor="p1" />
      </token-annotation>
      <text-annotation>
          <annotator processor="p1" />
      </text-annotation>
      <pos-annotation set="adhoc">
          <annotator processor="p1" />
      </pos-annotation>
      <correction-annotation set="https://raw.githubusercontent.com/proycon/folia/
↪master/setdefinitions/spellingcorrection.foliaset.xml">
          <annotator processor="p1" />
      </correction-annotation>
      <sentence-annotation>
          <annotator processor="p1" />
      </sentence-annotation>
    </annotations>
    <provenance>
       <processor xml:id="p1" name="proycon" type="manual" />
       <processor xml:id="johndoe" name="johndoe" type="manual" />
    </provenance>
  </metadata>
  <text xml:id="example.text">
    <s xml:id="example.s.1">
        <t>Watch that tree</t>
        <w xml:id="example.s.1.w.1">
          <t>Watch</t>
          <pos class="verb" />
        </w>
        <w xml:id="example.s.1.w.2">
          <t>that</t>
          <pos class="determiner" />
        </w>
        <w xml:id="example.s.1.w.3">
          <t>tree</t>
          <correction xml:id="example.correction.2" class="spelling"
            processor="p1" confidence="1.0">
              <new>
                  <pos class="noun" />
              </new>
```

(continues on next page)

(continued from previous page)

```
                <original>
                    <pos class="verb" />
                </original>
            </correction>
          </w>
      </s>
    </text>
</FoLiA>
```

Error detection

See also:

The correction of an error implies the detection of an error. In some cases, detection comes without correction and without suggestions for correction, for instance when the generation of correction suggestions is postponed to a later processing stage. You can use *Observation Annotation* to mark errors.

Suggestions for correction

The `<correction>` element can also be used in such situations in which you want to list *suggestions for correction*, but not yet commit to any single one. You may for example want to postpone this actual selection to another module or human annotator. The output of a speller check is typically a suggestion for correction. Recall that the actual correction is always included in the `<new>` tag, non-committing suggestions are included in the `<suggestion>` tag. All suggestions may take an ID and may specify an annotator/processor.

Suggestions never take sets or classes by themselves, the class and set pertain to the correction as a whole, and apply to all suggestions within. This implies that you will need *multiple* correction elements if you want to make suggestions of very distinct types. The following example shows two suggestions for spelling correction:

```
<?xml version='1.0' encoding='utf-8'?>
<FoLiA xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="http://ilk.uvt.nl/folia"␣
↪xml:id="page1263" version="2.0.0">
  <metadata type="native">
    <annotations>
      <token-annotation>
          <annotator processor="p1" />
      </token-annotation>
      <sentence-annotation>
          <annotator processor="p1" />
      </sentence-annotation>
      <text-annotation>
          <annotator processor="p1" />
      </text-annotation>
      <correction-annotation set="https://raw.githubusercontent.com/proycon/folia/
↪master/setdefinitions/spellingcorrection.foliaset.xml">
          <annotator processor="spellingcorrector" />
      </correction-annotation>
    </annotations>
    <provenance>
       <processor xml:id="p1" name="proycon" type="manual" />
       <processor xml:id="spellingcorrector" name="spellingcorrector" />
    </provenance>
  </metadata>
  <text xml:id="example.text">
    <s xml:id="example.s.1">
```

(continues on next page)

(continued from previous page)

```
        <w xml:id="example.s.1.w.1">
          <t>Watch</t>
        </w>
        <w xml:id="example.s.1.w.2">
          <t>that</t>
        </w>
        <w xml:id="example.s.1.w.3">
          <t>treee</t>
          <correction xml:id="example.correction.1" class="spelling" processor=
↪"spellingcorrector">
              <suggestion confidence="0.8">
                  <t>tree</t>
              </suggestion>
              <suggestion confidence="0.2">
                  <t>three</t>
              </suggestion>
          </correction>
        </w>
    </s>
  </text>
</FoLiA>
```

In the situation above we have a possible correction with two suggestions, none of which has been selected yet. The actual text remains unmodified so there are no `<new>` or `<original>` tags.

When an actual correction is made, the correction element changes. It may still retain the list of suggestions. In the following example, a human annotator named John Doe took one of the suggestions and made the actual correction:

```
<w>
    <correction xml:id="example.correction.1" class="spelling" processor="johndoe">
        <new>
            <t>tree</t>
        </new>
        <suggestion confidence="0.8">
            <t>tree</t>
        </suggestion>
        <suggestion confidence="0.2">
            <t>three</t>
        </suggestion>
        <original>
            <t>treee</t>
        </original>
    </correction>
</w>
```

Structural corrections: Merges, splits and swaps

Sometimes in the context of spelling correction, one wants to merge multiple tokens into one single new token, or the other way around; split one token into multiple new ones. The FoLiA format does not allow you to simply create new tokens and reassign identifiers. Identifiers are by definition permanent and should never change, as this would break backward compatibility. So such a change is therefore by definition a correction, and one uses the `<correction>` element to merge and split tokens.

We will first demonstrate a merge of two tokens (*on line*) into one (*online*). The original tokens are always retained within the `<original>` element. First a peek at the XML prior to merging:

```
<s xml:id="example.p.1.s.1">
    <w xml:id="example.p.1.s.1.w.1">
        <t>on</t>
    </w>
    <w xml:id="example.p.1.s.1.w.2">
        <t>line</t>
    </w>
</s>
```

And after merging:

```
<s xml:id="example.p.1.s.1">
 <correction xml:id="example.p.1.s.1.c.1" class="merge">
    <new>
        <w xml:id="example.p.1.s.1.w.1-2">
            <t>online</t>
        </w>
    </new>
    <original>
        <w xml:id="example.p.1.s.1.w.1">
            <t>on</t>
        </w>
        <w xml:id="example.p.1.s.1.w.2">
            <t>line</t>
        </w>
    </original>
 </correction>
</s>
```

Note that the correction element here is a member of the sentence (`<s>`), rather than the word token (`<w>`) as in all previous examples. The class, as always, is just a fictitious example and users can assign their own according to their own sets.

Now we will look at a split, the reverse of the above situation. Prior to splitting, assume we have:

```
<s xml:id="example.p.1.s.1">
 <w xml:id="example.p.1.s.1.w.1">
    <t>online</t>
 </w>
</s>
```

After splitting:

```
<s xml:id="example.p.1.s.1">
 <correction xml:id="example.p.1.s.1.c.1" class="split">
    <new>
        <w xml:id="example.p.1.s.1.w.1_1">
            <t>on</t>
        </w>
        <w xml:id="example.p.1.s.1.w.1_2">
            <t>line</t>
        </w>
    </new>
    <original>
        <w xml:id="example.p.1.s.1.w.1">
            <t>online</t>
        </w>
    </original>
```

(continues on next page)

(continued from previous page)

```
 </correction>
</s>
```

The same principle as used for merges and splits can also be used for performing *swap* corrections:

```
<s xml:id="example.p.1.s.1">
 <correction xml:id="example.p.1.s.1.c.1" class="swap">
    <new>
        <w xml:id="example.p.1.s.1.w.2_1">
            <t>on</t>
        </w>
        <w xml:id="example.p.1.s.1.w.1_2">
            <t>line</t>
        </w>
    </new>
    <original>
        <w xml:id="example.p.1.s.1.w.1">
            <t>line</t>
        </w>
        <w xml:id="example.p.1.s.1.w.2">
            <t>on</t>
        </w>
    </original>
 </correction>
</s>
```

Note that in such a swap situation, the identifiers of the swapped tokens tokens are new. They are essentially copies of the originals. Likewise, any token annotations you want to preserve explicitly need to be copies.

Insertions and Deletions

Insertions are words that are omitted in the original and have to be inserted in correction, while deletions are words that are erroneously inserted in the original and have to be removed in correction. FoLiA deals with these in a similar way to merges, splits and swaps. For deletions, the `<new>` element is simply empty. In the following example the word *the* was duplicated and removed in correction:

```
<s xml:id="example.p.1.s.1">
 <w xml:id="example.p.1.s.1.w.1">
    <t>the</t>
 </w>
 <correction xml:id="example.p.1.s.1.c.1" class="duplicate">
    <new/>
    <original>
        <w xml:id="example.p.1.s.1.w.2">
            <t>the</t>
        </w>
    </original>
 </correction>
 <w xml:id="example.p.1.s.1.w.3">
    <t>man</t>
 </w>
</s>
```

For insertions, the `<original>` element is empty:

```
<s xml:id="example.p.1.s.1">
 <w xml:id="example.p.1.s.1.w.1">
    <t>the</t>
 </w>
 <correction xml:id="example.p.1.s.1.c.1" class="duplicate">
    <new>
        <w xml:id="example.p.1.s.1.w.1_1">
            <t>old</t>
        </w>
    </new>
    <original />
 </correction>
 <w xml:id="example.p.1.s.1.w.2">
    <t>man</t>
 </w>
</s>
```

Although we limited our discussion to merges, splits, insertions and deletions applied to words/tokens, they may be applied to any other structural element just as well.

Suggestions for correction: structural changes

The earlier described suggestions for correction can be extended to merges, splits, insertions and deletions as well. This is done by embedding the newly suggested structure in `<suggestion>` elements. The current version of the structure is moved to within the scope of a `<current>` element.

We illustrate the splitting of online to on line as a suggestion for correction:

```
<s xml:id="example.p.1.s.1">
 <correction xml:id="example.p.1.s.1.c.1" class="split">
    <current>
        <w xml:id="example.p.1.s.1.w.1">
            <t>online</t>
        </w>
    </current>
    <suggestion>
        <w xml:id="example.p.1.s.1.w.1_1">
            <t>on</t>
        </w>
        <w xml:id="example.p.1.s.1.w.1_2">
            <t>line</t>
        </w>
    </suggestion>
 </correction>
</s>
```

Special cases are insertions and deletions. In case of suggested insertions, the current element is empty (but always present!), in case of deletions, the suggestion element is empty (but always present!).

For non-structural suggestions for correction, we simply have multiple correction elements if there are suggestions for correction of different classes. When structural changes are proposed, however, this is not possible, as there can be only one `<current>` element. The remedy here is to nest corrections, a current element may hold a correction with its own current element, and so on.

We can use suggestions for correction on any structural level; so we can for instance embed entire sentences or paragraphs within a suggestion. However, this quickly becomes very verbose and redundant as all the lower levels are copied for each suggestion. Common structural changes, as we have seen, are splits and merges. The `<suggestion>` element has a special additional facility to signal splits and merges, using the `split` and merge attribute, the value of which points to the ID (or IDs, space delimited) of the elements to split or merge with. When applied to sentences, splits and merges often coincide with an insertion of punctuation (for a sentence split), or deletion of redundant punctuation (for a sentence merge). The following two examples illustrate both these cases:

```
<p xml:id="correctionexample.p.2">
    <s xml:id="correctionexample.p.2.s.1">
        <w xml:id="correctionexample.p.2.s.1.w.1"><t>I</t></w>
        <w xml:id="correctionexample.p.2.s.1.w.2"><t>think</t></w>
        <correction xml:id="correctionexample.p.2.correction.1" class=
↪"redundantpunctuation">
            <suggestion merge="correctionexample.p.2.s.2" />
            <current>
                <w xml:id="correctionexample.p.2.s.1.w.3"><t>.</t></w>
            </current>
        </correction>
    </s>
    <s xml:id="correctionexample.p.2.s.2">
        <w xml:id="correctionexample.p.2.s.2.w.1"><t>and</t></w>
        <w xml:id="correctionexample.p.2.s.2.w.2"><t>therefore</t></w>
        <w xml:id="correctionexample.p.2.s.2.w.3"><t>I</t></w>
        <w xml:id="correctionexample.p.2.s.2.w.4"><t>am</t></w>
        <w xml:id="correctionexample.p.2.s.2.w.5"><t>.</t></w>
    </s>
</p>
```

```
<p xml:id="correctionexample.p.2">
    <s xml:id="correctionexample.p.2.s.1">
        <w xml:id="correctionexample.p.2.s.1.w.1"><t>I</t></w>
        <w xml:id="correctionexample.p.2.s.1.w.2"><t>go</t></w>
        <w xml:id="correctionexample.p.2.s.1.w.3"><t>home</t></w>
        <correction xml:id="correctionexample.p.2.correction.1" class=
↪"missingpunctuation">
            <suggestion split="correctionexample.p.2.s.1">
                <w xml:id="correctionexample.p.2.s.1.w.3a"><t>.</t></w>
            </suggestion>
            <current />
        </correction>
        <w xml:id="correctionexample.p.2.s.1.w.4">
          <t>you</t>
          <correction xml:id="correctionexample.p.2.correction.2" class=
↪"capitalizationerror">
            <suggestion>
              <t>You</t>
            </suggestion>
          </correction>
        </w>
        <w xml:id="correctionexample.p.2.s.1.w.5"><t>welcome</t></w>
        <w xml:id="correctionexample.p.2.s.1.w.6"><t>me</t></w>
        <w xml:id="correctionexample.p.2.s.1.w.7"><t>.</t></w>
    </s>
</p>
```

In the second example, we also add an additional non-structural suggestion for correction, suggesting to capitalize the first word of what is suggested to become a new sentence.

4.2.2 Gap Annotation

Sometimes there are parts of a document you want to skip and not annotate at all, but include as is. This is where gap annotation comes in, the user-defined set may indicate the kind of gap. Common omissions in books are for example front-matter and back-matter, i.e. the cover.

Specification

Structure Element

Annotation Category *Higher-order Annotation*

Declaration `<gap-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since the beginning

Element `<gap>`

API Class `Gap` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.

- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data <comment> (*Comment Annotation*), <content> (*Raw Content*), <desc> (*Description Annotation*), <metric> (*Metric Annotation*), <part> (*Part Annotation*)

Valid Context <div> (*Division Annotation*), <event> (*Event Annotation*), <head> (*Head Annotation*), <p> (*Paragraph Annotation*), <quote> (*Quote Annotation*), <s> (*Sentence Annotation*), <term> (*Term Annotation*), <utt> (*Utterance Annotation*)

Text markup Element

Element <t-gap>

API Class TextMarkupGap (FoLiApy API Reference)

Required Attributes

Optional Attributes

- xml:id – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- set – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The set must be referred to also in the *Annotation Declarations* for this annotation type.
- class – The class of the annotation, i.e. the annotation tag in the vocabulary defined by set.
- processor – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- annotator – This is an older alternative to the processor attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- annotatortype – This is an older alternative to the processor attribute, without support for full provenance. It is used together with annotator and specific the type of the annotator, either manual for human annotators or auto for automated systems.
- confidence – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- datetime – The date and time when this annotation was recorded, the format is YYYY-MM-DDThh:mm:ss (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- n – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- src – Points to a file or full URL of a sound or video file. This attribute is inheritable.

- begintime – A timestamp in HH:MM:SS.MMM format, indicating the begin time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- endtime – A timestamp in HH:MM:SS.MMM format, indicating the end time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- speaker – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.
- xlink:href – Turns this element into a hyperlink to the specified URL
- xlink:type – The type of link (you'll want to use simple in almost all cases).

Accepted Data <comment> (*Comment Annotation*), <desc> (*Description Annotation*), <br> (*Linebreak*)

Valid Context

Explanation

Sometimes there are parts of a document you want to skip and not annotate, but include as is. For this purpose the <gap> element should be used. Gaps may have a particular class indicating the kind of gap it is, defined by a user-defined set. Common omissions are for example front-matter and back-matter, text that is illegible/inaudible or in a foreign language. Again, the semantics depend on your set.

Although a gap skips over content, you may still want to explicitly add the raw content, this is done with the <content> element (see *Raw Content*). As this concerns raw content, it can not be annotated any further and we use XML CDATA type here to include it verbatim.

The following example shows the the use of <gap>:

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <division-annotation set="https://raw.githubusercontent.com/
↪LanguageMachines/uctodata/master/setdefinitions/divisions.foliaset.xml">
                         <annotator processor="p1" />
                 </division-annotation>
          <gap-annotation set="adhoc">
                         <annotator processor="p1" />
                 </gap-annotation>
          <rawcontent-annotation>
                         <annotator processor="p1" />
                 </rawcontent-annotation>
          <description-annotation>
                         <annotator processor="p1" />
```

(continues on next page)

(continued from previous page)

```xml
                  </description-annotation>
          <paragraph-annotation>
                        <annotator processor="p1" />
                  </paragraph-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
     <gap class="frontmatter">
        <desc>This is the cover of the book</desc>
        <content>
<![CDATA[

            SHOW WHITE AND THE SEVEN DWARFS


                by the Brothers Grimm

                    first edition


            Copyright(c) blah blah
]]>
        </content>
     </gap>
     <div xml:id="example.div.1" class="chapter" n="1">
         <t>In the <t-gap class="illegible" /> there was a princess...</t>
     </div>
  </text>
</FoLiA>
```

The gap element comes in two flavours, there is not just the aforementioned structural elements but there is also a text markup element (see *Text Markup Annotation*). This is the text markup element <t-gap> and it offers a more fine-grained variant for use in untokenised text. It indicates a gap in the textual content and is also shown in the above example. Either text is not available or there is a deliberate blank for, for example, fill-in exercises. It is recommended to provide a textual value when possible, but this is not required.

If you find that you want to mark your whole text content as being a <t-gap>, then this is a sure sign you should use the structural element <gap> instead.

Note: Both elements are the same annotation type so share the same declaration.

Text Redundancy

In cases of *text redundancy* (see *Text Annotation*), the <t-gap> element may take an ID reference attribute that refers to a gap element, as shown in the following example:

```xml
<s>
  <t>to <t-gap id="gap.1" class="fillin">be</t-gap> or not to be</t>
  <w><t>to</t></w>
  <gap xml:id="gap.1" class="fillin"><content>be</content></gap>
  <w><t>or</t></w>
```

(continues on next page)

(continued from previous page)

```
  <w><t>not</t></w>
  <w><t>to</t></w>
  <w><t>be</t></w>
</s>
```

4.2.3 Relation Annotation

FoLiA provides a facility to relate arbitrary parts of your document with other parts of your document, or even with parts of other FoLiA documents or external resources, even in other formats. It thus allows linking resources together. Within this context, the `xref` element is used to refer to the linked FoLiA elements.

Specification

Annotation Category *Higher-order Annotation*

Declaration `<relation-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Revised since v0.8, renamed from alignment in v2.0

Element `<relation>`

API Class `Relation` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be

an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).

- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.
- `xlink:href` – Turns this element into a hyperlink to the specified URL
- `xlink:type` – The type of link (you'll want to use `simple` in almost all cases).

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<metric>` (*Metric Annotation*)

Valid Context `<chunk>` (*Chunking*), `<coreferencechain>` (*Coreference Annotation*), `<coreferencelink>` (*Coreference Annotation*), `<def>` (*Definition Annotation*), `<dependency>` (*Dependency Annotation*), `<div>` (*Division Annotation*), `<entity>` (*Entity Annotation*), `<entry>` (*Entry Annotation*), `<event>` (*Event Annotation*), `<ex>` (*Example Annotation*), `<figure>` (*Figure Annotation*), `<head>` (*Head Annotation*), `<hiddenw>` (*Hidden Token Annotation*), `<br>` (*Linebreak*), `<list>` (*List Annotation*), `<modality>` (*Modality Annotation*), `<morpheme>` (*Morphological Annotation*), `<note>` (*Note Annotation*), `<observation>` (*Observation Annotation*), `<p>` (*Paragraph Annotation*), `<part>` (*Part Annotation*), `<phoneme>` (*Phonological Annotation*), `<predicate>` (*Predicate Annotation*), `<quote>` (*Quote Annotation*), `<ref>` (*Reference Annotation*), `<semrole>` (*Semantic Role Annotation*), `<s>` (*Sentence Annotation*), `<sentiment>` (*Sentiment Annotation*), `<spanrelation>` (*Span Relation Annotation*), `<statement>` (*Statement Annotation*), `<str>` (*String Annotation*), `<su>` (*Syntactic Annotation*), `<table>` (*Table Annotation*), `<term>` (*Term Annotation*), `<timesegment>` (*Time Segmentation*), `<utt>` (*Utterance Annotation*), `<whitespace>` (*Vertical Whitespace*), `<w>` (*Token Annotation*)

Explanation

Note: In versions of FoLiA prior to 2.0, this annotation type was called *alignments*

FoLiA provides a facility to link parts of your document with other parts of your document, or even with parts of other FoLiA documents or external resources. These are called *relations* and are implemented using the `<relation>` element. Within this context, the `<xref>` element is used to cross-link to the related FoLiA elements.

Consider the two following aligned sentences from excerpts of two **separate** FoLiA documents in different languages:

```
<s xml:id="example-english.p.1.s.1">
  <t>The Dalai Lama greeted him.</t>
  <relation class="french-translation" xlink:href="doc-french.xml"
    xlink:type="simple">
     <xref id="doc-french.p.1.s.1" t="Le Dalai Lama le saluait."
     type="s" />
  </relation>
</s>
```

```
<s xml:id="example-french.p.1.s.1">
  <t>Le Dalai Lama le saluait.</t>
  <relation class="english-translation" xlink:href="doc-english.xml"
    xlink:type="simple">
      <xref id="doc-english.p.1.s.1" t="The Dalai Lama greeted him."
       type="s" />
  </relation>
  <relation class="dutch-translation" xlink:href="doc-dutch.xml"
     xlink:type="simple">
      <xref id="doc-dutch.p.1.s.1" t="De Dalai Lama begroette hem."
       type="s" />
  </relation>
</s>
```

It is the job of the `<relation>` element to point to the relevant resource, whereas the `<xref>` element points to a specific point *inside* the referenced resource. The `xlink:href` attribute is used to link to the target document, if any. If the relation is within the same document then it should simply be omitted. The `type` attribute on `<xref>` specifies the type of element the relation points too, i.e. its value is equal to the tagname it points to. The `t` attribute to the `<xref>` element is merely optional and this overhead is added simply to facilitate the job of limited FoLiA parsers and provides a quick reference to the target text for both parsers and human users.

Although the above example has a single relation reference (`<xref>`), it is not forbidden to specify multiple references within the `<relation>` block, effectively referring to a span of multiple elements at the target.

By default, relations are between FoLiA documents. It is, however, also possible to point to resources in different formats. This has to be made explicit using the `format` attribute on the `<relation>` element. The value of the `format` attribute is a MIME type and defaults to `text/folia+xml` (naming follows RFC 3032). In the following example align a section (`<div>`) with the original HTML document from which the FoLiA document is arrived, and where the section is expressed with an HTML anchor (`<a>`) tag.

```
<div class="section">
 <t>lorum ipsum etc.</t>
 <relation class="original" xlink:href="http://somewhere/original.html"
    xlink:type="simple" format="text/html">
    <xref id="section2" type="a" />
 </relation>
</div>
```

See also:

For more complex many-to-many relations, see *Span Relation Annotation*, an extension of the current annotation type.

Translations

relation Annotation and *Span Relation Annotation* are an excellent tool for specifying translations. For situations in which relations seem overkill, a simple multi-document mechanism is available. This mechanism is based purely on convention: It assumes that structural elements that are translations simply share the same ID. This approach is quite feasible when used on higher-level structural elements, such as divisions, paragraphs, events or entries.

Example

The following example shows *Entity Annotation* with relations to Wikipedia.

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" xmlns:xlink="http://www.w3.org/1999/xlink"␣
↪version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl" format="text/turtle">
                         <annotator processor="p1" />
                  </token-annotation>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
          </sentence-annotation>
          <paragraph-annotation>
                         <annotator processor="p1" />
          </paragraph-annotation>
          <entity-annotation set="https://raw.githubusercontent.com/proycon/folia/
↪master/setdefinitions/namedentities.foliaset.ttl" format="text/turtle">
                         <annotator processor="p1" />
                  </entity-annotation>
          <relation-annotation set="adhoc">
                         <annotator processor="p1" />
          </relation-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <p xml:id="example.p.1">
      <s xml:id="example.p.1.s.1">
         <t>The Dalai Lama currently lives in Dharamsala in India.</t>
         <w xml:id="example.p.1.s.1.w.1" class="WORD">
            <t>The</t>
         </w>
         <w xml:id="example.p.1.s.1.w.2" class="WORD">
            <t>Dalai</t>
         </w>
         <w xml:id="example.p.1.s.1.w.3" class="WORD">
            <t>Lama</t>
         </w>
         <w xml:id="example.p.1.s.1.w.4" class="WORD">
            <t>currently</t>
         </w>
         <w xml:id="example.p.1.s.1.w.5" class="WORD">
            <t>lives</t>
         </w>
         <w xml:id="example.p.1.s.1.w.6" class="WORD">
```

(continues on next page)

(continued from previous page)

```xml
                <t>in</t>
            </w>
            <w xml:id="example.p.1.s.1.w.7" class="WORD">
                <t>Dharamsala</t>
            </w>
            <w xml:id="example.p.1.s.1.w.8" class="WORD">
                <t>in</t>
            </w>
            <w xml:id="example.p.1.s.1.w.9" class="WORD" space="no">
                <t>India</t>
            </w>
            <w xml:id="example.p.1.s.1.w.10" class="PUNCTUATION">
                <t>.</t>
            </w>
            <entities>
                <entity xml:id="example.p.1.s.1.entity.1" class="per">
                    <relation class="wikipedia" xlink:href="https://en.wikipedia.org/
↪wiki/Dalai_Lama" xlink:type="simple" format="text/html" />
                    <wref id="example.p.1.s.1.w.2" t="Dalai" />
                    <wref id="example.p.1.s.1.w.3" t="Lama" />
                </entity>
                <entity xml:id="example.p.1.s.1.entity.2" class="loc.city">
                    <relation class="wikipedia" xlink:href="https://en.wikipedia.org/
↪wiki/Dharamsala" xlink:type="simple" format="text/html" />
                    <wref id="example.p.1.s.1.w.7" t="Dharamsala" />
                </entity>
                <entity xml:id="example.p.1.s.1.entity.3" class="loc.country">
                    <relation class="wikipedia" xlink:href="https://en.wikipedia.org/
↪wiki/India" xlink:type="simple" format="text/html" />
                    <wref id="example.p.1.s.1.w.9" t="India" />
                </entity>
            </entities>
        </s>
      </p>
  </text>
</FoLiA>
```

The following example shows relations within strings in a document (See also *String Annotation*):

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
             <annotator processor="p1" />
          </text-annotation>
          <paragraph-annotation>
             <annotator processor="p1" />
          </paragraph-annotation>
          <string-annotation>
             <annotator processor="p1" />
          </string-annotation>
          <relation-annotation>
             <annotator processor="p1" />
          </relation-annotation>
      </annotations>
```

(continues on next page)

(continued from previous page)

```
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
     <p xml:id="example.p.1">
        <t><t-str id="example.p.1.str.1">Hello.</t-str> This is a sentence. Bye!</t>
        <t class="ocroutput"><t-str id="example.p.1.str.2">Hell0</t-str> Th1s iz a␣
↪sentence, Bye1</t>

        <str xml:id="example.p.1.str.1">
            <t offset="0">Hello.</t>
            <relation>
                <xref id="example.p.1.str.2" type="str" />
            </relation>
        </str>

        <str xml:id="example.p.1.str.2">
            <t class="ocroutput" offset="0">Hell0</t>
            <relation>
                <xref id="example.p.1.str.1" type="str" />
            </relation>
        </str>
     </p>
  </text>
</FoLiA>
```

4.2.4 Span Relation Annotation

Span relations are a stand-off extension of relation annotation that allows for more complex relations, such as word alignments that include many-to-one, one-to-many or many-to-many alignments. One of its uses is in the alignment of multiple translations of (parts) of a text.

Specification

Annotation Category *Higher-order Annotation*

Declaration `<spanrelation-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History since v0.8, renamed from complexalignment in v2.0

Element `<spanrelation>`

API Class `SpanRelation` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.

- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<metric>` (*Metric Annotation*), `<relation>` (*Relation Annotation*)

Valid Context `<spanrelations>` (*Span Relation Annotation*)

Explanation & Examples

Please ensure you are familiar with *Relation Annotation* first, as this is an extension for that annotation type.

Note: In versions of FoLiA prior to 2.0, this annotation type was called *complex alignments*

Under span relations we count more complex relations such as many-to-one, one-to-many and many-to-many relations between arbitrary elements. The element `<spanrelation>` behaves similarly to a span annotation element, operating in a stand-off fashion. This element groups `<relation>` elements together, effectively creating a many-to-many relation. The following example illustrates an example similar to the one above. All this takes place within the <spanrelations> annotation layer.

Consider the following example:

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" xmlns:xlink="http://www.w3.org/1999/xlink"␣
↪version="2.0" xml:id="example-english">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
             <annotator processor="p1" />
          </token-annotation>
          <text-annotation>
             <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
             <annotator processor="p1" />
          </sentence-annotation>
          <relation-annotation set="ad-hoc-translation-set">
             <annotator processor="p1" />
          </relation-annotation>
          <spanrelation-annotation>
             <annotator processor="p1" />
          </spanrelation-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example-english.text">
    <s xml:id="example-english.p.1.s.1">
      <t>The Dalai Lama greeted him.</t>
      <w xml:id="example-english.p.1.s.1.w.1"><t>The</t></w>
      <w xml:id="example-english.p.1.s.1.w.2"><t>Dalai</t></w>
      <w xml:id="example-english.p.1.s.1.w.3"><t>Lama</t></w>
      <w xml:id="example-english.p.1.s.1.w.4"><t>greeted</t></w>
      <w xml:id="example-english.p.1.s.1.w.5" space="no"><t>him</t></w>
      <w xml:id="example-english.p.1.s.1.w.6"><t>.</t></w>
      <spanrelations>
        <spanrelation>
          <relation class="original">
            <xref id="example-english.p.1.s.1.w.2" t="Dalai" type="w"/>
            <xref id="example-english.p.1.s.1.w.3" t="Lama" type="w"/>
          </relation>
          <relation class="french" xlink:href="doc-french.xml" xlink:type="simple">
            <xref id="example-french.p.1.s.1.w.2" t="Dalai" type="w"/>
            <xref id="example-french.p.1.s.1.w.3" t="Lama" type="w"/>
          </relation>
        </spanrelation>
      </spanrelations>
    </s>
  </text>
</FoLiA>
```

Here <xref> is used instead of the <wref> element we know from *Span Annotation*. as despite similarities relations are technically not exactly span annotation elements. You can in fact relate to anything that can carry an ID, within the same document and across multiple documents. Moreover, the notion of relations is not limited to just words, and it can be used for more than specifying translations.

The first `<relation>` element has no xlink reference, and therefore simply refers to the current document. The second relation element links to the foreign document. This notation is powerful as it allows you to specify a large number of relations in a concise matter. Consider the next example in which we added German and Italian, effectively specifying what can be perceived as 16 relationships over four different documents:

```
<s xml:id="example-english.p.1.s.1">
  <t>The Dalai Lama greeted him.</t>
  <w xml:id="example-english.p.1.s.1.w.1"><t>The</t></w>
  <w xml:id="example-english.p.1.s.1.w.2"><t>Dalai</t></w>
  <w xml:id="example-english.p.1.s.1.w.3"><t>Lama</t></w>
  <w xml:id="example-english.p.1.s.1.w.4"><t>greeted</t></w>
  <w xml:id="example-english.p.1.s.1.w.5"><t>him</t></w>
  <w xml:id="example-english.p.1.s.1.w.6"><t>.</t></w>
  <spanrelations>
    <spanrelation>
      <relation class="english-translation">
        <xref id="example-english.p.1.s.1.w.2" t="Dalai" type="w"/>
        <xref id="example-english.p.1.s.1.w.3" t="Lama" type="w"/>
      </relation>
      <relation class="french-translation"
       xlink:href="doc-french.xml"
       xlink:type="simple">
        <xref id="example-french.p.1.s.1.w.2" t="Dalai" type="w"/>
        <xref id="example-french.p.1.s.1.w.3" t="Lama" type="w"/>
      </relation>
      <relation class="german-translation"
       xlink:href="doc-german.xml"
       xlink:type="simple">
        <xref id="example-german.p.1.s.1.w.2" t="Dalai" type="w"/>
        <xref id="example-german.p.1.s.1.w.3" t="Lama" type="w"/>
      </relation>
      <relation class="italian-translation"
       xlink:href="doc-italian.xml"
       xlink:type="simple">
        <xref id="example-italian.p.1.s.1.w.2" t="Dalai" type="w"/>
        <xref id="example-italian.p.1.s.1.w.3" t="Lama" type="w"/>
      </relation>
    </spanrelation>
  </spanrelations>
</s>
```

Now you can even envision a FoLiA document that does not hold actual content, but acts merely as a document containing all relations between for example different translations of the document. Allowing for all relations to be contained in a single document rather than having to be made explicit in each language version.

The `<spanrelation>` element itself may also take a set, which is *independent* from the alignment set. They therefore also have two separate declarations.

4.2.5 Metric Annotation

Metric Annotation is a form of higher-order annotation that allows annotation of some kind of measurement. The type of measurement is defined by the class, which in turn is defined by the set as always. The metric element has a `value` attribute that stores the actual measurement, the value is often numeric but this needs not be the case.

Specification

Annotation Category *Higher-order Annotation*

Declaration `<metric-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History since v0.9

Element `<metric>`

API Class `Metric` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing

a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*)

Valid Context `<chunk>` (*Chunking*), `<coreferencechain>` (*Coreference Annotation*), `<coreferencelink>` (*Coreference Annotation*), `<correction>` (*Correction Annotation*), `<current>` (*Correction Annotation*), `<def>` (*Definition Annotation*), `<dependency>` (*Dependency Annotation*), `<div>` (*Division Annotation*), `<domain>` (*Domain/topic Annotation*), `<entity>` (*Entity Annotation*), `<entry>` (*Entry Annotation*), `<errordetection>` (*Error Detection Annotation (DEPRECATED)*), `<event>` (*Event Annotation*), `<ex>` (*Example Annotation*), `<figure>` (*Figure Annotation*), `<gap>` (*Gap Annotation*), `<head>` (*Head Annotation*), `<hiddenw>` (*Hidden Token Annotation*), `<lang>` (*Language Annotation*), `<lemma>` (*Lemmatisation*), `<br>` (*Linebreak*), `<list>` (*List Annotation*), `<modality>` (*Modality Annotation*), `<morpheme>` (*Morphological Annotation*), `<new>` (*Correction Annotation*), `<note>` (*Note Annotation*), `<observation>` (*Observation Annotation*), `<original>` (*Correction Annotation*), `<p>` (*Paragraph Annotation*), `<part>` (*Part Annotation*), `<phoneme>` (*Phonological Annotation*), `<pos>` (*Part-of-Speech Annotation*), `<predicate>` (*Predicate Annotation*), `<quote>` (*Quote Annotation*), `<ref>` (*Reference Annotation*), `<relation>` (*Relation Annotation*), `<semrole>` (*Semantic Role Annotation*), `<sense>` (*Sense Annotation*), `<s>` (*Sentence Annotation*), `<sentiment>` (*Sentiment Annotation*), `<spanrelation>` (*Span Relation Annotation*), `<statement>` (*Statement Annotation*), `<str>` (*String Annotation*), `<subjectivity>` (*Subjectivity Annotation (DEPRECATED)*), `<suggestion>` (*Correction Annotation*), `<su>` (*Syntactic Annotation*), `<table>` (*Table Annotation*), `<term>` (*Term Annotation*), `<timesegment>` (*Time Segmentation*), `<utt>` (*Utterance Annotation*), `<whitespace>` (*Vertical Whitespace*), `<w>` (*Token Annotation*)

Feature subsets (extra attributes)

- `value`

Explanation

The `<metric>` element allows annotation of some kind of measurement. The type of measurement is defined by the *class*, which in turn is user-defined by the set as always. The metric element has a `value` attribute that stores the actual measurement, the value is often numeric but this needs not be the case. It is a higher-level annotation element that may be used with any kind of annotation.

An example of measurements associated with a word/token:

```xml
<w xml:id="example.p.1.s.1.w.2">
    <t>boot</t>
    <metric class="charlength" value="4" />
    <metric class="frequency" value="0.00232" />
</w>
```

The next example shows measurements associated with a span annotation element, in this case to add geolocation information:

```xml
<entity class="location">
    <wref id="w3" t="New" />
    <wref id="w4" t="York" />
    <metric class="latitude" value="40.71274" />
    <metric class="longitude" value="-74.005974" />
</entity>
```

The next example demonstrates a full FoLiA document with metric annotation on a Figure, but it may be more appropriate to use *Submetadata* for this instead:

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <division-annotation set="https://raw.githubusercontent.com/
↪LanguageMachines/uctodata/master/setdefinitions/divisions.foliaset.xml">
                         <annotator processor="p1" />
                 </division-annotation>
          <head-annotation>
                         <annotator processor="p1" />
                 </head-annotation>
          <figure-annotation>
                         <annotator processor="p1" />
                 </figure-annotation>
          <metric-annotation set="adhoc-figure">
                         <annotator processor="p1" />
                 </metric-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
     <div xml:id="example.div.1" class="chapter" n="1">
         <head>
             <t>Frits Philips</t>
         </head>
         <figure xml:id="example.figure.1" n="1" src="https://upload.wikimedia.org/
↪wikipedia/commons/f/f8/Standbeeld_Frits_Philips.jpg">
           <metric class="photographer" value="Robert de Greef" />
           <metric class="city" value="Eindhoven" />
           <metric class="depicted" value="Frits Philips" />
           <metric class="license" value="CC-BY-SA 3.0" />
           <caption><t>Standbeeld van Frits Philips in Eindhoven</t></caption>
         </figure>
     </div>
  </text>
</FoLiA>
```

4.2.6 String Annotation

This is a form of higher-order annotation for selecting an arbitrary substring of a text, even untokenised, and allows further forms of higher-order annotation on the substring. It is also tied to a form of text markup annotation.

Specification

Annotation Category *Higher-order Annotation*

Declaration `<string-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History since v0.9.1

Element `<str>`

API Class `String` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<correction>` (*Correction Annotation*), `<desc>` (*Description Annotation*), `<metric>` (*Metric Annotation*), `<ph>` (*Phonetic Annotation/Content*), `<relation>` (*Relation Annotation*), `<t>` (*Text Annotation*)

Valid Context <current> (*Correction Annotation*), <def> (*Definition Annotation*), <entry> (*Entry Annotation*), <event> (*Event Annotation*), <ex> (*Example Annotation*), <figure> (*Figure Annotation*), <head> (*Head Annotation*), <hiddenw> (*Hidden Token Annotation*), <list> (*List Annotation*), <morpheme> (*Morphological Annotation*), <new> (*Correction Annotation*), <note> (*Note Annotation*), <original> (*Correction Annotation*), <p> (*Paragraph Annotation*), <phoneme> (*Phonological Annotation*), <quote> (*Quote Annotation*), <ref> (*Reference Annotation*), <s> (*Sentence Annotation*), <suggestion> (*Correction Annotation*), <term> (*Term Annotation*), <utt> (*Utterance Annotation*), <w> (*Token Annotation*)

Explanation

The <str> element is available in FoLiA to allow annotations on untokenised substrings. It is a higher-order annotation element that refers to a substring of the text-content (<t>) element on the same level, but is specified outside from it.

Explicitly denoting substrings in this fashion is needed when you want to associate further annotations with a substring. Consider the following example:

```
<p xml:id="example.p.1">
   <t>Hello. This is a sentence. Bye!</t>
   <str xml:id="example.p.1.str.1">
       <t offset="0">Hello</t>
       <desc>This is a word of greeting</desc>
   </str>
</p>
```

In substrings, using an offset attribute on the text-content element enables substrings to be properly positioned with respect to their *parent* text.

The <str> element has a text markup (*Text Markup Annotation*) counterpart called <t-str>. Both share the same declaration. The text markup variant can be used in the scope of the text content itself and may be more intuitive, but it is also less flexible, as it does not allow further annotations in its scope and can not be used when substrings are overlapping, unlike <str>. Consider the following example:

```
<p xml:id="example.p.1">
   <t><t-str id="example.p.1.str.1">Hello</t-str>. This is a sentence. Bye!</t>
   <str xml:id="example.p.1.str.1">
       <t offset="0">Hello</t>
       <desc>This is a word of greeting</desc>
   </str>
</p>
```

In the above example, the id parameter (distinct from xml:id!) on <t-str> is a reference to the <str> element, showing how the two elements can be used in combination.

One of the features of <str> is that you can put *Inline Annotation* in its scope, so you can associate e.g. PoS tags and lemmas with substrings in special cases where you might need to do this. Do note that this is **NOT** a substitute or alternative for proper tokenisation (*Token Annotation*), nor *Morphological Annotation*!

String elements are a form of higher-order annotation, they are similar to structure annotation but carry several distinct properties. Unlike structure elements, substring order does not matter and substrings may overlap. The difference between *Token Annotation* (<w>) and string annotation (<str>) has to be clearly understood, the former refers to actual tokens and supports further token annotation, the latter to untokenised or differently tokenised substrings.The

Of course, the <str> elements themselves may carry a class, associated with a user-defined set.

Textclasses (advanced)

If you are familiar with *Text classes (advanced)*, then it is good to know that this principle of course extends to within substrings as well. Consider the following example with three text layers, from each of them the same substring has been extracted:

```
<p xml:id="example.p.1">
   <t>Hello. This is a sentence. Bye!</t>
   <t class="normalised">Hello. This iz a sentence. Bye!</t>
   <t class="ocroutput">Hell0 Th1s iz a sentence, Bye1</t>

   <str xml:id="example.p.1.str.1">
       <t class="ocroutput" offset="0">Hell0</t>
   </str>

   <str xml:id="example.p.1.str.2">
       <t class="normalised" offset="0">Hello.</t>
   </str>

   <str xml:id="example.p.1.str.3">
       <t offset="0">Hello.</t>
   </str>
</p>
```

Instead of three separate substrings, we can also opt for a single one. Which solution is right for you depends on your own use case:

```
<p xml:id="example.p.1">
   <t>Hello. This is a sentence. Bye!</t>
   <t class="normalised">Hello. This iz a sentence. Bye!</t>
   <t class="ocroutput">Hell0 Th1s iz a sentence, Bye1</t>

   <str xml:id="example.p.1.str.1">
       <t class="ocroutput" offset="0">Hell0</t>
       <t class="normalised" offset="0">Hello</t>
       <t offset="0">Hello.</t>
   </str>
</p>
```

Or, if you do want separate strings but you also want to make the relation between them very explicit, then you can resort to *Relation Annotation* as shown in the next example:

```
<p xml:id="example.p.1">
   <t>Hello. This is a sentence. Bye!</t>
   <t class="ocroutput">Hell0 Th1s iz a sentence, Bye1</t>

   <str xml:id="example.p.1.str.1">
       <t class="ocroutput" offset="0">Hell0</t>
       <alignment>
           <aref id="example.p.1.str.2" type="str" />
       </alignment>
   </str>

   <str xml:id="example.p.1.str.2">
       <t offset="0">Hello.</t>
       <alignment>
           <aref id="example.p.1.str.1" type="str" />
       </alignment>
```

(continues on next page)

(continued from previous page)

```
  </str>
</p>
```

The <str> element is powerful when combined with alignments, as this allows the user to relate multiple alternative (pseudo-)tokenisations. This is also the limit as to what you can do with differing tokenisations in FoLiA, as FoLiA only supports one authoritative tokenisation.

Example

The following examples combines various aspects discussed in this section:

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
             <annotator processor="p1" />
          </text-annotation>
          <paragraph-annotation>
             <annotator processor="p1" />
          </paragraph-annotation>
          <string-annotation>
             <annotator processor="p1" />
          </string-annotation>
          <relation-annotation>
             <annotator processor="p1" />
          </relation-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
     <p xml:id="example.p.1">
        <t><t-str id="example.p.1.str.1">Hello.</t-str> This is a sentence. Bye!</t>
        <t class="ocroutput"><t-str id="example.p.1.str.2">Hell0</t-str> Th1s iz a␣
↪sentence, Bye1</t>

        <str xml:id="example.p.1.str.1">
            <t offset="0">Hello.</t>
            <relation>
                <xref id="example.p.1.str.2" type="str" />
            </relation>
        </str>

        <str xml:id="example.p.1.str.2">
            <t class="ocroutput" offset="0">Hell0</t>
            <relation>
                <xref id="example.p.1.str.1" type="str" />
            </relation>
        </str>
     </p>
  </text>
</FoLiA>
```

4.2.7 Alternative Annotation

This form of higher-order annotation encapsulates alternative annotations, i.e. annotations that are posed as an alternative option rather than the authoratitive chosen annotation

Specification

Annotation Category *Higher-order Annotation*

Declaration `<alternative-annotation>` *(note: there is never a set associated with this annotation type)

Version History Since the beginning, may carry set and classes since v2.0

Element `<alt>`

API Class `Alternative` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing

a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<correction>` (*Correction Annotation*), `<desc>` (*Description Annotation*), `<morphology>` (*Morphological Annotation*), `<phonology>` (*Phonological Annotation*)

Valid Context `<def>` (*Definition Annotation*), `<div>` (*Division Annotation*), `<entry>` (*Entry Annotation*), `<event>` (*Event Annotation*), `<ex>` (*Example Annotation*), `<figure>` (*Figure Annotation*), `<head>` (*Head Annotation*), `<hiddenw>` (*Hidden Token Annotation*), `<br>` (*Linebreak*), `<list>` (*List Annotation*), `<morpheme>` (*Morphological Annotation*), `<note>` (*Note Annotation*), `<p>` (*Paragraph Annotation*), `<part>` (*Part Annotation*), `<phoneme>` (*Phonological Annotation*), `<quote>` (*Quote Annotation*), `<ref>` (*Reference Annotation*), `<s>` (*Sentence Annotation*), `<table>` (*Table Annotation*), `<term>` (*Term Annotation*), `<utt>` (*Utterance Annotation*), `<whitespace>` (*Vertical Whitespace*), `<w>` (*Token Annotation*)

Element `<altlayers>`

API Class `AlternativeLayers` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.

- speaker – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data <comment> (*Comment Annotation*), <desc> (*Description Annotation*)

Valid Context <def> (*Definition Annotation*), <div> (*Division Annotation*), <entry> (*Entry Annotation*), <event> (*Event Annotation*), <ex> (*Example Annotation*), <figure> (*Figure Annotation*), <head> (*Head Annotation*), <hiddenw> (*Hidden Token Annotation*), <br> (*Linebreak*), <list> (*List Annotation*), <morpheme> (*Morphological Annotation*), <note> (*Note Annotation*), <p> (*Paragraph Annotation*), <part> (*Part Annotation*), <phoneme> (*Phonological Annotation*), <quote> (*Quote Annotation*), <ref> (*Reference Annotation*), <s> (*Sentence Annotation*), <table> (*Table Annotation*), <term> (*Term Annotation*), <utt> (*Utterance Annotation*), <whitespace> (*Vertical Whitespace*), <w> (*Token Annotation*)

Introduction

The FoLiA format does not just allow for a single authoritative annotation per token; it allows the representation of *alternative* annotations. There is a specific form for *Inline Annotation* and a form for *Span Annotation*; both share the same declaration <alternative-annotation> with which a set may be associated.

Alternative Inline Annotation

Alternative inline 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 a single set of alternatives.

Each alternative preferably is given a unique identifier. 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" confidence="1.0">
      <desc>furniture</desc>
    </sense>
    <alt xml:id="example.p.1.s.1.w.1.alt.1">
        <domain class="finance" />
        <sense class="r_n-5919" confidence="0.6">
            <desc>financial institution</desc>
        </sense>
    </alt>
    <alt xml:id="example.p.1.s.1.w.1.alt.2">
        <domain class="geology" />
        <sense class="r_n-5920" confidence="0.1">
            <desc>river bank</desc>
        </sense>
```

(continues on next page)

(continued from previous page)

```
    </alt>
</w>
```

Sometimes, an alternative is concerned only with a portion of the annotations. By default, annotations not mentioned are applicable to the alternative as well, unless the alternative is set as being *exclusive*. Consider the following expanded example in which we added a part-of-speech tag and a lemma.

```
<w xml:id="example.p.1.s.1.w.1">
    <t>bank</t>
    <domain class="furniture" />
    <sense class="r_n-5918" confidence="1.0">
      <desc>furniture</desc>
    </sense>
    <pos class="n" />
    <lemma class="bank" />
    <alt xml:id="example.p.1.s.1.w.1.alt.1">
        <domain class="finance" />
        <sense class="r_n-5919" confidence="0.6">
            <desc>financial institution</desc>
        </sense>
    </alt>
    <alt xml:id="example.p.1.s.1.w.1.alt.2">
        <domain class="geology" />
        <sense class="r_n-5920" confidence="0.1">
            <desc>river bank</desc>
        </sense>
    </alt>
    <alt xml:id="example.p.1.s.1.w.1.alt.2" exclusive="yes">
        <t>bank</t>
        <domain class="navigation" />
        <sense class="r_n-1234">
            <desc>to turn</desc>
        </sense>
        <pos class="v" />
        <lemma class="bank" />
    </alt>
</w>
```

The first two alternatives are inclusive, which is the default. This means that the pos tag `n` and the lemma `bank` apply to them as well. The last alternative is set as exclusive, using the `exclusive` attribute. It has been given a different pos tag and the lemma and even the text content has necessarily been repeated even though it is equal to the higher-level annotation, otherwise there would be no lemma nor text associated with the exclusive alternative.

Alternatives can be used as a great way of postponing actual annotation, due to their non-authoritative nature. When used in this way, they can be regarded as *options*. They can be used even when there are no authoritative annotations of the type. Consider the following example in which domain and sense annotations are presented as alternatives and there is no authoritative annotation of these types whatsoever:

```
<w xml:id="example.p.1.s.1.w.1">
    <t>bank</t>
    <alt xml:id="example.p.1.s.1.w.1.alt.1">
        <domain class="finance" />
        <sense class="r_n-5919" confidence="0.6">
            <desc>financial institution</desc>
        </sense>
    </alt>
```

(continues on next page)

(continued from previous page)

```
    <alt xml:id="example.p.1.s.1.w.1.alt.2">
        <domain class="geology" />
        <sense class="r_n-5920" confidence="0.1">
            <desc>river bank</desc>
        </sense>
    </alt>
</w>
```

Alternative Span Annotation

With inline annotations one can specify an unbounded number of alternative annotations. This functionality is available for *Span Annotation* as well, but due to the different nature of span annotations this happens in a slightly different way.

Where we used `<alt>` for token annotations, we now use `<altlayers>` for span annotations. Under this element several alternative layers can be presented. Analogous to `<alt>`, any layers grouped together are assumed to be somehow dependent. Multiple `<altlayers>` can be added to introduce independent alternatives. Each alternative may be associated with a unique identifier.

Below is an example of a sentence that is chunked in two ways:

```
<s xml:id="example.p.1.s.1">
  <t>The Dalai Lama greeted him.</t>
  <w xml:id="example.p.1.s.1.w.1"><t>The</t></w>
  <w xml:id="example.p.1.s.1.w.2"><t>Dalai</t></w>
  <w xml:id="example.p.1.s.1.w.3"><t>Lama</t></w>
  <w xml:id="example.p.1.s.1.w.4"><t>greeted</t></w>
  <w xml:id="example.p.1.s.1.w.5"><t>him</t></w>
  <w xml:id="example.p.1.s.1.w.6"><t>.</t></w>
  <chunking>
    <chunk xml:id="example.p.1.s.1.chunk.1">
        <wref id="example.p.1.s.1.w.1" t="The" />
        <wref id="example.p.1.s.1.w.2" t="Dalai" />
        <wref id="example.p.1.s.1.w.3" t="Lama" />
    </chunk>
    <chunk xml:id="example.p.1.s.1.chunk.2">
        <wref id="example.p.1.s.1.w.4" t="greeted" />
    </chunk>
    <chunk xml:id="example.p.1.s.1.chunk.3">
        <wref id="example.p.1.s.1.w.5" t="him" />
        <wref id="example.p.1.s.1.w.6" t="." />
    </chunk>
  </chunking>
  <altlayers xml:id="example.p.1.s.1.alt.1">
       <chunking>
        <chunk xml:id="example.p.1.s.1.alt.1.chunk.1" confidence="0.001">
            <wref id="example.p.1.s.1.w.1" t="The" />
            <wref id="example.p.1.s.1.w.2" t="Dalai" />
        </chunk>
        <chunk xml:id="example.p.1.s.1.alt.1.chunk.2" confidence="0.001">
            <wref id="example.p.1.s.1.w.2" t="Lama" />
            <wref id="example.p.1.s.1.w.4" t="greeted" />
        </chunk>
        <chunk xml:id="example.p.1.s.1.alt.1.chunk.3" confidence="0.001">
            <wref id="example.p.1.s.1.w.5" t="him" />
            <wref id="example.p.1.s.1.w.6" t="." />
```

(continues on next page)

(continued from previous page)

```
        </chunk>
      </chunking>
  </altlayers>
</s>
```

The support for alternatives and the fact that multiple layers (including those of different types) cannot be nested in a single inline structure, should make clear why FoLiA uses a stand-off notation alongside an inline notation.

4.2.8 Comment Annotation

This is a form of higher-order annotation that allows you to associate comments with almost all other annotation elements

Specification

Annotation Category *Higher-order Annotation*

Declaration `<comment-annotation>` *(note: there is never a set associated with this annotation type)

Version History Since v1.3

Element `<comment>`

API Class `Comment` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing

a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*)

Valid Context `<alt>` (*Alternative Annotation*), `<altlayers>` (*Alternative Annotation*), `<chunk>` (*Chunking*), `<chunking>` (*Chunking*), `<comment>` (*Comment Annotation*), `<content>` (*Raw Content*), `<coreferencechain>` (*Coreference Annotation*), `<coreferences>` (*Coreference Annotation*), `<coreferencelink>` (*Coreference Annotation*), `<correction>` (*Correction Annotation*), `<current>` (*Correction Annotation*), `<def>` (*Definition Annotation*), `<dependencies>` (*Dependency Annotation*), `<dependency>` (*Dependency Annotation*), `<desc>` (*Description Annotation*), `<div>` (*Division Annotation*), `<domain>` (*Domain/topic Annotation*), `<entities>` (*Entity Annotation*), `<entity>` (*Entity Annotation*), `<entry>` (*Entry Annotation*), `<errordetection>` (*Error Detection Annotation (DEPRECATED)*), `<event>` (*Event Annotation*), `<ex>` (*Example Annotation*), `<external>` (*External Annotation*), `<figure>` (*Figure Annotation*), `<gap>` (*Gap Annotation*), `<head>` (*Head Annotation*), `<hiddenw>` (*Hidden Token Annotation*), `<t-hbr>` (*Hyphenation*), `<lang>` (*Language Annotation*), `<lemma>` (*Lemmatisation*), `<br>` (*Linebreak*), `<list>` (*List Annotation*), `<metric>` (*Metric Annotation*), `<modalities>` (*Modality Annotation*), `<modality>` (*Modality Annotation*), `<morpheme>` (*Morphological Annotation*), `<morphology>` (*Morphological Annotation*), `<new>` (*Correction Annotation*), `<note>` (*Note Annotation*), `<observation>` (*Observation Annotation*), `<observations>` (*Observation Annotation*), `<original>` (*Correction Annotation*), `<p>` (*Paragraph Annotation*), `<part>` (*Part Annotation*), `<ph>` (*Phonetic Annotation/Content*), `<phoneme>` (*Phonological Annotation*), `<phonology>` (*Phonological Annotation*), `<pos>` (*Part-of-Speech Annotation*), `<predicate>` (*Predicate Annotation*), `<quote>` (*Quote Annotation*), `<ref>` (*Reference Annotation*), `<relation>` (*Relation Annotation*), `<semrole>` (*Semantic Role Annotation*), `<semroles>` (*Semantic Role Annotation*), `<sense>` (*Sense Annotation*), `<s>` (*Sentence Annotation*), `<sentiment>` (*Sentiment Annotation*), `<sentiments>` (*Sentiment Annotation*), `<spanrelation>` (*Span Relation Annotation*), `<spanrelations>` (*Span Relation Annotation*), `<statement>` (*Statement Annotation*), `<statements>` (*Statement Annotation*), `<str>` (*String Annotation*), `<subjectivity>` (*Subjectivity Annotation (DEPRECATED)*), `<suggestion>` (*Correction Annotation*), `<su>` (*Syntactic Annotation*), `<syntax>` (*Syntactic Annotation*), `<table>` (*Table Annotation*), `<term>` (*Term Annotation*), `<t>` (*Text Annotation*), `<t-correction>` (*Correction Annotation*), `<t-error>` (*Error Detection Annotation (DEPRECATED)*), `<t-gap>` (*Gap Annotation*), `<t-hspace>` (*Horizontal Whitespace*), `<t-lang>` (*Language Annotation*), `<t-ref>` (*Reference Annotation*), `<t-str>` (*String Annotation*), `<t-style>` (*Style Annotation*), `<t-whitespace>` (*Vertical Whitespace*), `<timesegment>` (*Time Segmentation*), `<timing>` (*Time Segmentation*), `<utt>` (*Utterance Annotation*), `<whitespace>` (*Vertical Whitespace*), `<w>` (*Token Annotation*)

Explanation

Comments is a simple higher-order annotation element that may be used with any annotation. It holds text that comments the annotation. Multiple comments are allowed per annotation.

An alternative to these FoLiA-specific comments, which are considered actual annotations, is standard XML comments. Standard XML comments, however, are not considered actual annotations and most likely won't be interpreted by any tools.

Example

Consider the following example in the context of *Sense Annotation*.

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
                         <annotator processor="p1" />
                  </token-annotation>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
          </sentence-annotation>
          <paragraph-annotation>
                         <annotator processor="p1" />
          </paragraph-annotation>
          <sense-annotation set="wordnet"> <!-- an ad-hoc set -->
                         <annotator processor="p1" />
                  </sense-annotation>
          <description-annotation>
                         <annotator processor="p1" />
                  </description-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <p xml:id="example.p.1">
      <s xml:id="example.p.1.s.2">
         <t>I show an example.</t>
         <w xml:id="example.p.1.s.2.w.1" class="WORD">
            <t>I</t>
         </w>
         <w xml:id="example.p.1.s.2.w.2" class="WORD">
            <t>show</t>
            <sense class="show%2:39:02::">
                                <desc>give an exhibition of to an interested audience
↪</desc>
            </sense>
         </w>
         <w xml:id="example.p.1.s.2.w.3" class="WORD">
            <t>an</t>
         </w>
         <w xml:id="example.p.1.s.2.w.4" class="WORD" space="no">
            <t>example</t>
            <sense class="example%1:09:00::">
                <desc>an item of information that is typical of a class or group)</
↪desc>
            </sense>
         </w>
         <w xml:id="example.p.1.s.2.w.5" class="PUNCTUATION">
            <t>.</t>
         </w>
      </s>
    </p>
```

(continues on next page)

(continued from previous page)

```
  </text>
</FoLiA>
```

4.2.9 Description Annotation

This is a form of higher-order annotation that allows you to associate descriptions with almost all other annotation elements

Specification

Annotation Category *Higher-order Annotation*

Declaration `<description-annotation>` *(note: there is never a set associated with this annotation type)

Version History Since the beginning

Element `<desc>`

API Class `Description` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data <comment> (*Comment Annotation*), <desc> (*Description Annotation*)

Valid Context <alt> (*Alternative Annotation*), <altlayers> (*Alternative Annotation*), <chunk> (*Chunking*), <chunking> (*Chunking*), <comment> (*Comment Annotation*), <content> (*Raw Content*), <coreferencechain> (*Coreference Annotation*), <coreferences> (*Coreference Annotation*), <coreferencelink> (*Coreference Annotation*), <correction> (*Correction Annotation*), <current> (*Correction Annotation*), <def> (*Definition Annotation*), <dependencies> (*Dependency Annotation*), <dependency> (*Dependency Annotation*), <desc> (*Description Annotation*), <div> (*Division Annotation*), <domain> (*Domain/topic Annotation*), <entities> (*Entity Annotation*), <entity> (*Entity Annotation*), <entry> (*Entry Annotation*), <errordetection> (*Error Detection Annotation (DEPRECATED)*), <event> (*Event Annotation*), <ex> (*Example Annotation*), <external> (*External Annotation*), <figure> (*Figure Annotation*), <gap> (*Gap Annotation*), <head> (*Head Annotation*), <hiddenw> (*Hidden Token Annotation*), <t-hbr> (*Hyphenation*), <lang> (*Language Annotation*), <lemma> (*Lemmatisation*), <br> (*Linebreak*), <list> (*List Annotation*), <metric> (*Metric Annotation*), <modalities> (*Modality Annotation*), <modality> (*Modality Annotation*), <morpheme> (*Morphological Annotation*), <morphology> (*Morphological Annotation*), <new> (*Correction Annotation*), <note> (*Note Annotation*), <observation> (*Observation Annotation*), <observations> (*Observation Annotation*), <original> (*Correction Annotation*), <p> (*Paragraph Annotation*), <part> (*Part Annotation*), <ph> (*Phonetic Annotation/Content*), <phoneme> (*Phonological Annotation*), <phonology> (*Phonological Annotation*), <pos> (*Part-of-Speech Annotation*), <predicate> (*Predicate Annotation*), <quote> (*Quote Annotation*), <ref> (*Reference Annotation*), <relation> (*Relation Annotation*), <semrole> (*Semantic Role Annotation*), <semroles> (*Semantic Role Annotation*), <sense> (*Sense Annotation*), <s> (*Sentence Annotation*), <sentiment> (*Sentiment Annotation*), <sentiments> (*Sentiment Annotation*), <spanrelation> (*Span Relation Annotation*), <spanrelations> (*Span Relation Annotation*), <statement> (*Statement Annotation*), <statements> (*Statement Annotation*), <str> (*String Annotation*), <subjectivity> (*Subjectivity Annotation (DEPRECATED)*), <suggestion> (*Correction Annotation*), <su> (*Syntactic Annotation*), <syntax> (*Syntactic Annotation*), <table> (*Table Annotation*), <term> (*Term Annotation*), <t> (*Text Annotation*), <t-correction> (*Correction Annotation*), <t-error> (*Error Detection Annotation (DEPRECATED)*), <t-gap> (*Gap Annotation*), <t-hspace> (*Horizontal Whitespace*), <t-lang> (*Language Annotation*), <t-ref> (*Reference Annotation*), <t-str> (*String Annotation*), <t-style> (*Style Annotation*), <t-whitespace> (*Vertical Whitespace*), <timesegment> (*Time Segmentation*), <timing> (*Time Segmentation*), <utt> (*Utterance Annotation*), <whitespace> (*Vertical Whitespace*), <w> (*Token Annotation*)

Explanation

This is one of the simplest forms of higher-order annotation. Any annotation element may hold a desc element containing in its body a human readable description for the annotation. Only one description is allowed per annotation.

Example

Consider the following example in the context of *Sense Annotation*.

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
                         <annotator processor="p1" />
                     </token-annotation>
          <text-annotation>
```

(continues on next page)

(continued from previous page)

```
                        <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                        <annotator processor="p1" />
          </sentence-annotation>
          <paragraph-annotation>
                        <annotator processor="p1" />
          </paragraph-annotation>
          <sense-annotation set="wordnet"> <!-- an ad-hoc set -->
                        <annotator processor="p1" />
                  </sense-annotation>
          <description-annotation>
                        <annotator processor="p1" />
                  </description-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <p xml:id="example.p.1">
      <s xml:id="example.p.1.s.2">
         <t>I show an example.</t>
         <w xml:id="example.p.1.s.2.w.1" class="WORD">
            <t>I</t>
         </w>
         <w xml:id="example.p.1.s.2.w.2" class="WORD">
            <t>show</t>
            <sense class="show%2:39:02::">
                               <desc>give an exhibition of to an interested audience
↪</desc>
            </sense>
         </w>
         <w xml:id="example.p.1.s.2.w.3" class="WORD">
            <t>an</t>
         </w>
         <w xml:id="example.p.1.s.2.w.4" class="WORD" space="no">
            <t>example</t>
            <sense class="example%1:09:00::">
                <desc>an item of information that is typical of a class or group)</
↪desc>
            </sense>
         </w>
         <w xml:id="example.p.1.s.2.w.5" class="PUNCTUATION">
            <t>.</t>
         </w>
      </s>
    </p>
  </text>
</FoLiA>
```

4.2.10 External Annotation

External annotation makes a reference to an external FoLiA document whose structure is inserted at the exact place the external element occurs.

Specification

Annotation Category *Higher-order Annotation*

Declaration `<external-annotation>` *(note: there is never a set associated with this annotation type)

Version History Since v2.4.0

Element `<external>`

API Class `External` (FoLiApy API Reference)

Required Attributes

- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*)

Valid Context `<def>` (*Definition Annotation*), `<div>` (*Division Annotation*), `<entry>` (*Entry Annotation*), `<event>` (*Event Annotation*), `<ex>` (*Example Annotation*), `<figure>` (*Figure Annotation*), `<head>` (*Head Annotation*), `<hiddenw>` (*Hidden Token Annotation*), `<br>` (*Linebreak*), `<list>` (*List Annotation*), `<note>` (*Note Annotation*), `<p>` (*Paragraph Annotation*), `<part>` (*Part Annotation*), `<quote>` (*Quote Annotation*), `<ref>` (*Reference Annotation*), `<s>` (*Sentence Annotation*), `<table>` (*Table Annotation*), `<term>` (*Term Annotation*), `<utt>` (*Utterance Annotation*), `<whitespace>` (*Vertical Whitespace*), `<w>` (*Token Annotation*)

Explanation

This annotation type type is used to split a larger document into multiple smaller ones, and link from the parent document to the external child documents. It is a type of higher-order annotation that is inserted at a certain place in the parent structure. The parent document would be functionally equivalent if the structure of the external child documents were inserted at the point the `<external>` element occurs.

The `<external>` element is valid in most structural elements. It is **not** a mechanism to create stand-off annotation documents. Each external document must also be a valid FoLiA document in its own right.

The `src` attribute can refer to a local file path (relative or absolute) or a remote URL.

Example

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <external-annotation>
                         <annotator processor="p1" />
                 </external-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <external src="chapter1.folia.xml" />
    <external src="chapter2.folia.xml" />
  </text>
</FoLiA>
```

4.3 Inline Annotation

This category encompasses (linguistic) annotation types describing a single structural element. Examples are Part-of-Speech Annotation or Lemmatisation, which often describe a single token.

These annotation types are encoded in an inline fashion in FoLiA, i.e. they appear within the structural element to which they apply (often words/tokens but not necessarily so) and make use of the hierarchical nature of XML.

FoLiA defines the following types of inline annotation:

- *Inline Annotation* – This category encompasses (linguistic) annotation types describing a single structural element. Examples are Part-of-Speech Annotation or Lemmatisation, which often describe a single token.
  - *Part-of-Speech Annotation* – `<pos>` – Part-of-Speech Annotation, one of the most common types of linguistic annotation. Assigns a lexical class to words.
  - *Lemmatisation* – `<lemma>` – Lemma Annotation, one of the most common types of linguistic annotation. Represents the canonical form of a word.
  - *Domain/topic Annotation* – `<domain>` – Domain/topic Annotation. A form of inline annotation used to assign a certain domain or topic to a structure element.
  - *Sense Annotation* – `<sense>` – Sense Annotation allows to assign a lexical semantic sense to a word.
  - *Error Detection Annotation (DEPRECATED)* – `<errordetection>` – This annotation type is deprecated in favour of *Observation Annotation* and only exists for backward compatibility.
  - *Subjectivity Annotation (DEPRECATED)* – `<subjectivity>` – This annotation type is deprecated in favour of *Sentiment Annotation* and only exists for backward compatibility.
  - *Language Annotation* – `<lang>` – Language Annotation simply identifies the language a part of the text is in. Though this information is often part of the metadata, this form is considered an actual annotation.

4.3.1 Part-of-Speech Annotation

Part-of-Speech Annotation, one of the most common types of linguistic annotation. Assigns a lexical class to words.

Specification

Annotation Category *Inline Annotation*

Declaration `<pos-annotation set="...">` *(note: set is mandatory)*

Version History Since the beginning

Element `<pos>`

API Class `PosAnnotation` (FoLiApy API Reference)

Required Attributes

- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.

- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `textclass` – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to `current`. See *Text class attribute (advanced)*.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<metric>` (*Metric Annotation*)

Valid Context

Feature subsets (extra attributes)

- `head`

Explanation & Examples

Part-of-Speech annotation allows the annotation of lexical categories using the `pos` element. The following example shows a simple part-of-speech annotation. In this example , we declare PoS annotation to use the tagset from the brown corpus (although we do not have an actual set definition for it).

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation />
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
                         <annotator processor="p1" />
                  </token-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
          </sentence-annotation>
          <paragraph-annotation>
                         <annotator processor="p1" />
          </paragraph-annotation>
          <pos-annotation set="brown"> <!-- This is an ad-hoc set declaration as it␣
↪is no URL and therefore not really defined -->
                         <annotator processor="p1" />
          </pos-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <s xml:id="example.p.1.s.2">
     <w xml:id="example.p.1.s.2.w.1" class="WORD">
        <t>This</t>
        <pos class="DT"/>
     </w>
     <w xml:id="example.p.1.s.2.w.2" class="WORD">
        <t>is</t>
        <pos class="VBZ"/>
     </w>
     <w xml:id="example.p.1.s.2.w.3" class="WORD">
        <t>an</t>
        <pos class="AT"/>
     </w>
     <w xml:id="example.p.1.s.2.w.4" class="WORD" space="no">
        <t>example</t>
        <pos class="NN"/>
     </w>
     <w xml:id="example.p.1.s.2.w.5" class="PUNCTUATION">
        <t>.</t>
        <pos class="."/>
     </w>
    </s>
  </text>
</FoLiA>
```

Lexical annotation can take more complex forms than assignment of a single part-of-speech tag. There may for example be numerous features associated with the part-of-speech tag, such as gender, number, case, tense, mood, etc... FoLiA introduces a special paradigm for dealing with such features. This is described in *Features*, please ensure you are familiar with this before reading the remainder of this section.

Two scenarios can be envisioned, one in which the class of the pos element encodes all features, and one in which it is the foundation upon which is expanded. Which one is used is entirely up to the defined set.

Option one:

```
<w xml:id="example.p.1.s.1.w.2">
    <t>boot</t>
    <pos head="N" class="N(singular)">
        <feat subset="number" class="singular" />
        <feat subset="gender" class="none" />
        <feat subset="case" class="none" />
    </pos>
</w>
```

In FoLiA, this attribute head is a *predefined subset* for PoS-annotation, i.e. the subset is commonly used and has clear semantics; however, it still needs to be defined in the set definition. We can use such *predefined subsets* as XML attributes.

Option two:

```
<w xml:id="example.p.1.s.1.w.2">
    <t>boot</t>
    <pos class="N">
        <feat subset="number" class="singular" />
        <feat subset="gender" class="none" />
        <feat subset="case" class="none" />
    </pos>
</w>
```

The last examples demonstrates a full FoLiA document with part-of-speech tagging with features:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="folia.xsl"?>
<FoLiA xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="http://ilk.uvt.nl/folia"␣
↪xml:id="example.deep" generator="libfolia-v1.5" version="2.0.0">
  <metadata type="native">
    <annotations>
      <text-annotation>
                           <annotator processor="p1" />
      </text-annotation>
      <sentence-annotation>
                           <annotator processor="p1" />
      </sentence-annotation>
      <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/folia1.4/setdefinitions/tokconfig-nld.foliaset.ttl">
                           <annotator processor="p2" />
      </token-annotation>
      <pos-annotation set="https://raw.githubusercontent.com/proycon/folia/master/
↪setdefinitions/frog-mbpos-cgn">
                           <annotator processor="p3.1" />
      </pos-annotation>
      <lemma-annotation set="https://raw.githubusercontent.com/proycon/folia/master/
↪setdefinitions/frog-mblem-nl">
                           <annotator processor="p3.2" />
      </lemma-annotation>
    </annotations>
    <provenance>
       <processor xml:id="p1" name="proycon" type="manual" />
       <processor xml:id="p2" name="ucto" version="0.14" />
       <processor xml:id="p3" name="frog" version="0.16" begindatetime="2016-11-
↪15T15:12:00">
```

(continues on next page)

(continued from previous page)

```
           <processor xml:id="p3.0" name="libfolia" version="1.14" type="generator" />
           <processor xml:id="p3.1" name="mbpos" version="1.0" />
           <processor xml:id="p3.2" name="mblem" version="1.1" />
       </processor>
    </provenance>
    <meta id="language">nld</meta>
  </metadata>
  <text xml:id="example.deep.text">
      <s xml:id="example.deep.p.1.s.1">
        <t>De Russen kennen Nova Zembla sinds de 11e of 12e eeuw, toen handelaars van Novgorod het eiland al aandeden.</t>
        <w xml:id="example.deep.p.1.s.1.w.1" class="WORD">
          <t>De</t>
          <pos class="LID(bep,stan,rest)" confidence="0.779762" head="LID">
            <feat class="bep" subset="lwtype"/>
            <feat class="stan" subset="naamval"/>
            <feat class="rest" subset="npagr"/>
          </pos>
          <lemma class="de"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.2" class="WORD">
          <t>Russen</t>
          <pos class="SPEC(deeleigen)" confidence="1" head="SPEC">
            <feat class="deeleigen" subset="spectype"/>
          </pos>
          <lemma class="Russen"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.3" class="WORD">
          <t>kennen</t>
          <pos class="WW(pv,tgw,mv)" confidence="0.833333" head="WW">
            <feat class="pv" subset="wvorm"/>
            <feat class="tgw" subset="pvtijd"/>
            <feat class="mv" subset="pvagr"/>
          </pos>
          <lemma class="kennen"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.4" class="WORD">
          <t>Nova</t>
          <pos class="SPEC(deeleigen)" confidence="1" head="SPEC">
            <feat class="deeleigen" subset="spectype"/>
          </pos>
          <lemma class="Nova"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.5" class="WORD">
          <t>Zembla</t>
          <pos class="SPEC(deeleigen)" confidence="1" head="SPEC">
            <feat class="deeleigen" subset="spectype"/>
          </pos>
          <lemma class="Zembla"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.6" class="WORD">
          <t>sinds</t>
          <pos class="VZ(init)" confidence="0.999078" head="VZ">
            <feat class="init" subset="vztype"/>
          </pos>
          <lemma class="sinds"/>
```

(continues on next page)

(continued from previous page)

```
        </w>
        <w xml:id="example.deep.p.1.s.1.w.7" class="WORD">
          <t>de</t>
          <pos class="LID(bep,stan,rest)" confidence="0.981886" head="LID">
            <feat class="bep" subset="lwtype"/>
            <feat class="stan" subset="naamval"/>
            <feat class="rest" subset="npagr"/>
          </pos>
          <lemma class="de"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.8" class="NUMBER-ORDINAL">
          <t>11e</t>
          <pos class="TW(rang,prenom,stan)" confidence="0.990632" head="TW">
            <feat class="rang" subset="numtype"/>
            <feat class="prenom" subset="positie"/>
            <feat class="stan" subset="naamval"/>
          </pos>
          <lemma class="11"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.9" class="WORD">
          <t>of</t>
          <pos class="VG(neven)" confidence="0.855677" head="VG">
            <feat class="neven" subset="conjtype"/>
          </pos>
          <lemma class="of"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.10" class="NUMBER-ORDINAL">
          <t>12e</t>
          <pos class="TW(rang,prenom,stan)" confidence="0.990632" head="TW">
            <feat class="rang" subset="numtype"/>
            <feat class="prenom" subset="positie"/>
            <feat class="stan" subset="naamval"/>
          </pos>
          <lemma class="12"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.11" class="WORD" space="no">
          <t>eeuw</t>
          <pos class="N(soort,ev,basis,zijd,stan)" confidence="0.999633" head="N">
            <feat class="soort" subset="ntype"/>
            <feat class="ev" subset="getal"/>
            <feat class="basis" subset="graad"/>
            <feat class="zijd" subset="genus"/>
            <feat class="stan" subset="naamval"/>
          </pos>
          <lemma class="eeuw"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.12" class="PUNCTUATION">
          <t>,</t>
          <pos class="LET()" confidence="1" head="LET"/>
          <lemma class=","/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.13" class="WORD">
          <t>toen</t>
          <pos class="VG(onder)" confidence="0.571429" head="VG">
            <feat class="onder" subset="conjtype"/>
          </pos>
```

(continues on next page)

(continued from previous page)

```
          <lemma class="toen"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.14" class="WORD">
          <t>handelaars</t>
          <pos class="N(soort,mv,basis)" confidence="0.99944" head="N">
            <feat class="soort" subset="ntype"/>
            <feat class="mv" subset="getal"/>
            <feat class="basis" subset="graad"/>
          </pos>
          <lemma class="handelaar"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.15" class="WORD">
          <t>van</t>
          <pos class="VZ(init)" confidence="0.999469" head="VZ">
            <feat class="init" subset="vztype"/>
          </pos>
          <lemma class="van"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.16" class="WORD">
          <t>Novgorod</t>
          <pos class="SPEC(deeleigen)" confidence="1" head="SPEC">
            <feat class="deeleigen" subset="spectype"/>
          </pos>
          <lemma class="Novgorod"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.17" class="WORD">
          <t>het</t>
          <pos class="LID(bep,stan,evon)" confidence="0.996855" head="LID">
            <feat class="bep" subset="lwtype"/>
            <feat class="stan" subset="naamval"/>
            <feat class="evon" subset="npagr"/>
          </pos>
          <lemma class="het"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.18" class="WORD">
          <t>eiland</t>
          <pos class="N(soort,ev,basis,onz,stan)" confidence="0.996804" head="N">
            <feat class="soort" subset="ntype"/>
            <feat class="ev" subset="getal"/>
            <feat class="basis" subset="graad"/>
            <feat class="onz" subset="genus"/>
            <feat class="stan" subset="naamval"/>
          </pos>
          <lemma class="eiland"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.19" class="WORD">
          <t>al</t>
          <pos class="BW()" confidence="0.90383" head="BW"/>
          <lemma class="al"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.20" class="WORD" space="no">
          <t>aandeden</t>
          <pos class="WW(pv,verl,mv)" confidence="0.999559" head="WW">
            <feat class="pv" subset="wvorm"/>
            <feat class="verl" subset="pvtijd"/>
            <feat class="mv" subset="pvagr"/>
```

(continues on next page)

(continued from previous page)

```
          </pos>
          <lemma class="aandoen"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.21" class="PUNCTUATION">
          <t>.</t>
          <pos class="LET()" confidence="1" head="LET"/>
          <lemma class="."/>
        </w>
      </s>
  </text>
</FoLiA>
```

4.3.2 Lemmatisation

Lemma Annotation, one of the most common types of linguistic annotation. Represents the canonical form of a word.

Specification

Annotation Category *Inline Annotation*

Declaration `<lemma-annotation set="...">` *(note: set is mandatory)*

Version History Since the beginning

Element `<lemma>`

API Class `LemmaAnnotation` (FoLiApy API Reference)

Required Attributes

- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.

- annotatortype – This is an older alternative to the processor attribute, without support for full provenance. It is used together with annotator and specific the type of the annotator, either manual for human annotators or auto for automated systems.
- confidence – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- datetime – The date and time when this annotation was recorded, the format is YYYY-MM-DDThh:mm:ss (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- n – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- textclass – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to current. See *Text class attribute (advanced)*.
- src – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- begintime – A timestamp in HH:MM:SS.MMM format, indicating the begin time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- endtime – A timestamp in HH:MM:SS.MMM format, indicating the end time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- speaker – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data <comment> (*Comment Annotation*), <desc> (*Description Annotation*), <metric> (*Metric Annotation*)

Valid Context

Example

The following example includes lemmas as well as part-of-speech tags:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="folia.xsl"?>
<FoLiA xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="http://ilk.uvt.nl/folia"␣
↪xml:id="example.deep" generator="libfolia-v1.5" version="2.0.0">
  <metadata type="native">
    <annotations>
      <text-annotation>
                         <annotator processor="p1" />
      </text-annotation>
      <sentence-annotation>
                         <annotator processor="p1" />
      </sentence-annotation>
```

(continues on next page)

(continued from previous page)

```
      <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/folia1.4/setdefinitions/tokconfig-nld.foliaset.ttl">
                         <annotator processor="p2" />
      </token-annotation>
      <pos-annotation set="https://raw.githubusercontent.com/proycon/folia/master/
↪setdefinitions/frog-mbpos-cgn">
                         <annotator processor="p3.1" />
      </pos-annotation>
      <lemma-annotation set="https://raw.githubusercontent.com/proycon/folia/master/
↪setdefinitions/frog-mblem-nl">
                         <annotator processor="p3.2" />
      </lemma-annotation>
    </annotations>
    <provenance>
       <processor xml:id="p1" name="proycon" type="manual" />
       <processor xml:id="p2" name="ucto" version="0.14" />
       <processor xml:id="p3" name="frog" version="0.16" begindatetime="2016-11-
↪15T15:12:00">
           <processor xml:id="p3.0" name="libfolia" version="1.14" type="generator" />
           <processor xml:id="p3.1" name="mbpos" version="1.0" />
           <processor xml:id="p3.2" name="mblem" version="1.1" />
       </processor>
    </provenance>
    <meta id="language">nld</meta>
  </metadata>
  <text xml:id="example.deep.text">
      <s xml:id="example.deep.p.1.s.1">
        <t>De Russen kennen Nova Zembla sinds de 11e of 12e eeuw, toen handelaars van␣
↪Novgorod het eiland al aandeden.</t>
        <w xml:id="example.deep.p.1.s.1.w.1" class="WORD">
          <t>De</t>
          <pos class="LID(bep,stan,rest)" confidence="0.779762" head="LID">
            <feat class="bep" subset="lwtype"/>
            <feat class="stan" subset="naamval"/>
            <feat class="rest" subset="npagr"/>
          </pos>
          <lemma class="de"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.2" class="WORD">
          <t>Russen</t>
          <pos class="SPEC(deeleigen)" confidence="1" head="SPEC">
            <feat class="deeleigen" subset="spectype"/>
          </pos>
          <lemma class="Russen"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.3" class="WORD">
          <t>kennen</t>
          <pos class="WW(pv,tgw,mv)" confidence="0.833333" head="WW">
            <feat class="pv" subset="wvorm"/>
            <feat class="tgw" subset="pvtijd"/>
            <feat class="mv" subset="pvagr"/>
          </pos>
          <lemma class="kennen"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.4" class="WORD">
          <t>Nova</t>
```

(continues on next page)

(continued from previous page)

```
          <pos class="SPEC(deeleigen)" confidence="1" head="SPEC">
            <feat class="deeleigen" subset="spectype"/>
          </pos>
          <lemma class="Nova"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.5" class="WORD">
          <t>Zembla</t>
          <pos class="SPEC(deeleigen)" confidence="1" head="SPEC">
            <feat class="deeleigen" subset="spectype"/>
          </pos>
          <lemma class="Zembla"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.6" class="WORD">
          <t>sinds</t>
          <pos class="VZ(init)" confidence="0.999078" head="VZ">
            <feat class="init" subset="vztype"/>
          </pos>
          <lemma class="sinds"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.7" class="WORD">
          <t>de</t>
          <pos class="LID(bep,stan,rest)" confidence="0.981886" head="LID">
            <feat class="bep" subset="lwtype"/>
            <feat class="stan" subset="naamval"/>
            <feat class="rest" subset="npagr"/>
          </pos>
          <lemma class="de"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.8" class="NUMBER-ORDINAL">
          <t>11e</t>
          <pos class="TW(rang,prenom,stan)" confidence="0.990632" head="TW">
            <feat class="rang" subset="numtype"/>
            <feat class="prenom" subset="positie"/>
            <feat class="stan" subset="naamval"/>
          </pos>
          <lemma class="11"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.9" class="WORD">
          <t>of</t>
          <pos class="VG(neven)" confidence="0.855677" head="VG">
            <feat class="neven" subset="conjtype"/>
          </pos>
          <lemma class="of"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.10" class="NUMBER-ORDINAL">
          <t>12e</t>
          <pos class="TW(rang,prenom,stan)" confidence="0.990632" head="TW">
            <feat class="rang" subset="numtype"/>
            <feat class="prenom" subset="positie"/>
            <feat class="stan" subset="naamval"/>
          </pos>
          <lemma class="12"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.11" class="WORD" space="no">
          <t>eeuw</t>
          <pos class="N(soort,ev,basis,zijd,stan)" confidence="0.999633" head="N">
```

(continues on next page)

(continued from previous page)

```
          <feat class="soort" subset="ntype"/>
          <feat class="ev" subset="getal"/>
          <feat class="basis" subset="graad"/>
          <feat class="zijd" subset="genus"/>
          <feat class="stan" subset="naamval"/>
        </pos>
        <lemma class="eeuw"/>
      </w>
      <w xml:id="example.deep.p.1.s.1.w.12" class="PUNCTUATION">
        <t>,</t>
        <pos class="LET()" confidence="1" head="LET"/>
        <lemma class=","/>
      </w>
      <w xml:id="example.deep.p.1.s.1.w.13" class="WORD">
        <t>toen</t>
        <pos class="VG(onder)" confidence="0.571429" head="VG">
          <feat class="onder" subset="conjtype"/>
        </pos>
        <lemma class="toen"/>
      </w>
      <w xml:id="example.deep.p.1.s.1.w.14" class="WORD">
        <t>handelaars</t>
        <pos class="N(soort,mv,basis)" confidence="0.99944" head="N">
          <feat class="soort" subset="ntype"/>
          <feat class="mv" subset="getal"/>
          <feat class="basis" subset="graad"/>
        </pos>
        <lemma class="handelaar"/>
      </w>
      <w xml:id="example.deep.p.1.s.1.w.15" class="WORD">
        <t>van</t>
        <pos class="VZ(init)" confidence="0.999469" head="VZ">
          <feat class="init" subset="vztype"/>
        </pos>
        <lemma class="van"/>
      </w>
      <w xml:id="example.deep.p.1.s.1.w.16" class="WORD">
        <t>Novgorod</t>
        <pos class="SPEC(deeleigen)" confidence="1" head="SPEC">
          <feat class="deeleigen" subset="spectype"/>
        </pos>
        <lemma class="Novgorod"/>
      </w>
      <w xml:id="example.deep.p.1.s.1.w.17" class="WORD">
        <t>het</t>
        <pos class="LID(bep,stan,evon)" confidence="0.996855" head="LID">
          <feat class="bep" subset="lwtype"/>
          <feat class="stan" subset="naamval"/>
          <feat class="evon" subset="npagr"/>
        </pos>
        <lemma class="het"/>
      </w>
      <w xml:id="example.deep.p.1.s.1.w.18" class="WORD">
        <t>eiland</t>
        <pos class="N(soort,ev,basis,onz,stan)" confidence="0.996804" head="N">
          <feat class="soort" subset="ntype"/>
```

(continues on next page)

(continued from previous page)

```
            <feat class="ev" subset="getal"/>
            <feat class="basis" subset="graad"/>
            <feat class="onz" subset="genus"/>
            <feat class="stan" subset="naamval"/>
          </pos>
          <lemma class="eiland"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.19" class="WORD">
          <t>al</t>
          <pos class="BW()" confidence="0.90383" head="BW"/>
          <lemma class="al"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.20" class="WORD" space="no">
          <t>aandeden</t>
          <pos class="WW(pv,verl,mv)" confidence="0.999559" head="WW">
            <feat class="pv" subset="wvorm"/>
            <feat class="verl" subset="pvtijd"/>
            <feat class="mv" subset="pvagr"/>
          </pos>
          <lemma class="aandoen"/>
        </w>
        <w xml:id="example.deep.p.1.s.1.w.21" class="PUNCTUATION">
          <t>.</t>
          <pos class="LET()" confidence="1" head="LET"/>
          <lemma class="."/>
        </w>
      </s>
  </text>
</FoLiA>
```

4.3.3 Domain/topic Annotation

Domain/topic Annotation. A form of inline annotation used to assign a certain domain or topic to a structure element.

Specification

Annotation Category *Inline Annotation*

Declaration `<domain-annotation set="...">` *(note: set is mandatory)*

Version History Since the beginning

Element `<domain>`

API Class `DomainAnnotation` (FoLiApy API Reference)

Required Attributes

- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `textclass` – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to `current`. See *Text class attribute (advanced)*.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<metric>` (*Metric Annotation*)

Valid Context

Example

The following document shows a domain annotation on the sentence level:

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
          </sentence-annotation>
          <paragraph-annotation>
                         <annotator processor="p1" />
          </paragraph-annotation>
          <domain-annotation set="topics"> <!-- an ad-hoc set -->
                         <annotator processor="p1" />
                  </domain-annotation>
          <lang-annotation set="iso638-3"> <!-- an ad-hoc set -->
                         <annotator processor="p1" />
                  </lang-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <p xml:id="example.p.1">
      <s xml:id="example.p.1.s.1">
         <t>I show an example:</t>
         <lang class="eng" />
      </s>
      <s xml:id="example.p.1.s.2">
         <t>        ,          .</t>
         <domain class="animals" />
         <lang class="rus" />
      </s>
    </p>
  </text>
</FoLiA>
```

4.3.4 Sense Annotation

Sense Annotation allows to assign a lexical semantic sense to a word.

Specification

Annotation Category *Inline Annotation*

Declaration `<sense-annotation set="...">` *(note: set is mandatory)*

Version History Since the beginning

Element `<sense>`

API Class `SenseAnnotation` (FoLiApy API Reference)

Required Attributes

- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `textclass` – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to `current`. See *Text class attribute (advanced)*.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry

no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<metric>` (*Metric Annotation*)

Valid Context

Feature subsets (extra attributes)

- `synset`

Explanation

In semantic sense annotation, the classes will correspond to some kind of lexical unit ID or synset ID. In vocabularies that make an explicit distinction between lexical units and synonym sets (synsets), you can use the `synset` *predefined subset* for notation of the latter. A simpler alternative is to just use two different sets similtaneously (two `sense` annotations per item).

You can use *Description Annotation* to optionally associate a human readable description with the sense annotation (or any other annotation for that matter).

Example

The following example shows sense annotations:

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
                         <annotator processor="p1" />
                  </token-annotation>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
          </sentence-annotation>
          <paragraph-annotation>
                         <annotator processor="p1" />
          </paragraph-annotation>
          <sense-annotation set="wordnet"> <!-- an ad-hoc set -->
                         <annotator processor="p1" />
                  </sense-annotation>
          <description-annotation>
                         <annotator processor="p1" />
                  </description-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <p xml:id="example.p.1">
      <s xml:id="example.p.1.s.2">
```

(continues on next page)

(continued from previous page)

```
         <t>I show an example.</t>
         <w xml:id="example.p.1.s.2.w.1" class="WORD">
            <t>I</t>
         </w>
         <w xml:id="example.p.1.s.2.w.2" class="WORD">
            <t>show</t>
            <sense class="show%2:39:02::">
                                    <desc>give an exhibition of to an interested audience
↪</desc>
            </sense>
         </w>
         <w xml:id="example.p.1.s.2.w.3" class="WORD">
            <t>an</t>
         </w>
         <w xml:id="example.p.1.s.2.w.4" class="WORD" space="no">
            <t>example</t>
            <sense class="example%1:09:00::">
                <desc>an item of information that is typical of a class or group)</
↪desc>
            </sense>
         </w>
         <w xml:id="example.p.1.s.2.w.5" class="PUNCTUATION">
            <t>.</t>
         </w>
      </s>
    </p>
  </text>
</FoLiA>
```

4.3.5 Error Detection Annotation (DEPRECATED)

This annotation type is deprecated, see *Observation Annotation* instead.

4.3.6 Subjectivity Annotation (DEPRECATED)

This annotation type is deprecated, see *Sentiment Annotation* instead.

4.3.7 Language Annotation

Language Annotation simply identifies the language a part of the text is in. Though this information is often part of the metadata, this form is considered an actual annotation.

Specification

Annotation Category *Inline Annotation*

Declaration `<lang-annotation set="...">` *(note: set is mandatory)*

Version History since v0.8.1

Element `<lang>`

API Class `LangAnnotation` (FoLiApy API Reference)

Required Attributes

- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `textclass` – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to `current`. See *Text class attribute (advanced)*.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they

use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<metric>` (*Metric Annotation*)

Valid Context

Text markup Element

Element `<t-lang>`

API Class `TextMarkupLanguage` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.

- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.
- xlink:href – Turns this element into a hyperlink to the specified URL
- xlink:type – The type of link (you'll want to use simple in almost all cases).

Accepted Data <comment> (*Comment Annotation*), <desc> (*Description Annotation*), <br> (*Linebreak*)

Valid Context

Explanation

Language identification is used to identify a certain structural element as being in a certain language, so it can be applied to the text as a whole or smaller elements within it. The language vocabulary is determined by the set definition.

The text markup variant (<t-lang>), can be used in non-tokenised contexts.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
          </sentence-annotation>
          <paragraph-annotation>
                         <annotator processor="p1" />
          </paragraph-annotation>
          <domain-annotation set="topics"> <!-- an ad-hoc set -->
                         <annotator processor="p1" />
                  </domain-annotation>
          <lang-annotation set="iso638-3"> <!-- an ad-hoc set -->
                         <annotator processor="p1" />
                  </lang-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <p xml:id="example.p.1">
      <s xml:id="example.p.1.s.1">
         <t>I show an example:</t>
         <lang class="eng" />
      </s>
```

(continues on next page)

(continued from previous page)

```
      <s xml:id="example.p.1.s.2">
        <t>        ,          .</t>
        <domain class="animals" />
        <lang class="rus" />
      </s>
    </p>
  </text>
</FoLiA>
```

4.4 Span Annotation

This category encompasses (linguistic) annotation types that span one or more structural elements. Examples are (Named) Entities or Multi-word Expressions, Dependency Relations, and many others. FoLiA implements these as a stand-off layer that refers back to the structural elements (often words/tokens). The layer itself is embedded in a structural level of a wider scope (such as a sentence).

Span annotation elements are always embedded in a **layer** element, this is an element that groups span annotations of a particular annotation type and set together. Each annotation type has its own layer element and the layer elements themselves are embedded, inline, in a structural element. So, say you want to do named entity annotation (a form of span annotation) over words, then *after* you defined the words, you can embed a layer (`<entities>`) containing the span annotation elements (`<entity>` in this example), which refer back to the words. Such a reference back is done with the `wref` element.

Consider the following example:

```
<s xml:id="example.p.1.s.1">
  <t>The Dalai Lama greeted him.</t>
  <w xml:id="example.p.1.s.1.w.1"><t>The</t></w>
  <w xml:id="example.p.1.s.1.w.2"><t>Dalai</t></w>
  <w xml:id="example.p.1.s.1.w.3"><t>Lama</t></w>
  <w xml:id="example.p.1.s.1.w.4"><t>greeted</t></w>
  <w xml:id="example.p.1.s.1.w.5"><t>him</t></w>
  <w xml:id="example.p.1.s.1.w.6"><t>.</t></w>
  <entities>
    <entity xml:id="example.p.1.s.1.entity.1" class="per">
        <wref id="example.p.1.s.1.w.2" t="Dalai" />
        <wref id="example.p.1.s.1.w.3" t="Lama" />
    </entity>
  </entities>
</s>
```

The next sentence may in turn have an `<entities>` layer as well. The design principle behind this is to keep information, even when it concerns span annotations, as local as possible rather than spread out of the document. This facilitates the job for streaming parsers and humans looking at the raw XML. Nevertheless, this is a convention which most FoLiA libraries adhere to, but is not a strict requirement. So it is still possible and valid to place your layer at any higher structural level, as long as all the elements you refer to are within its scope and all defined prior to the layer itself.

Note: As you might have seen, the `wref` element may carry a `t` attribute with the text of word/structure it refers to. This redundancy is merely to provide extra clarity to the person inspecting the XML and is not mandatory.

Note: The `wref` elements refers to words/tokens or sub-token annotations such as morphemes and phonemes. We do not use it to refer to higher-level structural elements!

Note: The order of the references should always correspond to the order of the tokens in the text. However, the references need not be strictly continuous; there may be gaps.

Depending on the type of span annotation, it is possible that the element may be nested. This is for example the case for *Syntactic Annotation*, where the nesting of syntactic units allows the building of syntax trees. Span annotation elements of a more complex nature may require or allow so-called **span role** elements. Span roles encapsulate references to the words and ascribe a more defined meaning to the span, for instance to mark the head or dependent in a dependency relation. Span role elements themselves never carry any classes and can only be used in the scope of a certain span annotation element, not standalone. They can still carry *Features*, though.

FoLiA defines the following types of span annotation:

- *Span Annotation* – This category encompasses (linguistic) annotation types that span one or more structural elements. Examples are (Named) Entities or Multi-word Expressions, Dependency Relations, and many others. FoLiA implements these as a stand-off layer that refers back to the structural elements (often words/tokens). The layer itself is embedded in a structural level of a wider scope (such as a sentence).
  - *Syntactic Annotation* – `<su>` – Assign grammatical categories to spans of words. Syntactic units are nestable and allow representation of complete syntax trees that are usually the result of consistuency parsing.
  - *Chunking* – `<chunk>` – Assigns shallow grammatical categories to spans of words. Unlike syntax annotation, chunks are not nestable. They are often produced by a process called Shallow Parsing, or alternatively, chunking.
  - *Entity Annotation* – `<entity>` – Entity annotation is a broad and common category in FoLiA. It is used for specifying all kinds of multi-word expressions, including but not limited to named entities. The set definition used determines the vocabulary and therefore the precise nature of the entity annotation.
  - *Dependency Annotation* – `<dependency>` – Dependency relations are syntactic relations between spans of tokens. A dependency relation takes a particular class and consists of a single head component and a single dependent component.
  - *Time Segmentation* – `<timesegment>` – FoLiA supports time segmentation to allow for more fine-grained control of timing information by associating spans of words/tokens with exact timestamps. It can provide a more linguistic alternative to *Event Annotation*.
  - *Coreference Annotation* – `<coreferencechain>` – Relations between words that refer to the same referent (anaphora) are expressed in FoLiA using Coreference Annotation. The co-reference relations are expressed by specifying the entire chain in which all links are coreferent.
  - *Semantic Role Annotation* – `<semrole>` – This span annotation type allows for the expression of semantic roles, or thematic roles. It is often used together with *Predicate Annotation*
  - *Predicate Annotation* – `<predicate>` – Allows annotation of predicates, this annotation type is usually used together with Semantic Role Annotation. The types of predicates are defined by a user-defined set definition.
  - *Observation Annotation* – `<observation>` – Observation annotation is used to make an observation pertaining to one or more word tokens. Observations offer a an external qualification on part of a text. The qualification is expressed by the class, in turn defined by a set. The precise semantics of the observation depends on the user-defined set.
  - *Sentiment Annotation* – `<sentiment>` – Sentiment analysis marks subjective information such as sentiments or attitudes expressed in text. The sentiments/attitudes are defined by a user-defined set definition.

- *Statement Annotation* – `<statement>` – Statement annotation, sometimes also refered to as attribution, allows to decompose statements into the source of the statement, the content of the statement, and the way these relate, provided these are made explicit in the text.
- *Modality Annotation* – `<modality>` – Modality annotation is used to describe the relationship between cue word(s) and the scope it covers. It is primarily used for the annotation of negation, but also for the annotation of factuality, certainty and truthfulness:.

4.4.1 Syntactic Annotation

Assign grammatical categories to spans of words. Syntactic units are nestable and allow representation of complete syntax trees that are usually the result of consistuency parsing.

Specification

Annotation Category *Span Annotation*

Declaration `<syntax-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since the beginning

Element `<su>`

API Class `SyntacticUnit` (FoLiApy API Reference)

Layer Element `<syntax>`

Span Role Elements

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.

- n – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- textclass – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to current. See *Text class attribute (advanced)*.
- src – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- begintime – A timestamp in HH:MM:SS.MMM format, indicating the begin time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- endtime – A timestamp in HH:MM:SS.MMM format, indicating the end time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- speaker – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data <comment> (*Comment Annotation*), <desc> (*Description Annotation*), <metric> (*Metric Annotation*), <relation> (*Relation Annotation*), <su> (*Syntactic Annotation*)

Valid Context <su> (*Syntactic Annotation*), <syntax> (*Syntactic Annotation*)

Explanation

Note: Please first ensure you are familiar with the general principles of *Span Annotation* to make sense of this annotation type.

Syntax annotation allows representation of a syntax tree, commonly the result of *constituency parsing*. This is a nested form of span annotation, in which nodes in the tree are represented by <su> (syntactic unit) elements. Each syntactic unit may carry a class in a user-defined set, determining the vocabulary of the syntax annotation.

It is recommended for each syntactic unit to have a unique identifier.

See also:

For dependency parsing, see *Dependency Annotation* instead.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
```

(continues on next page)

(continued from previous page)

```xml
                        <annotator processor="p1" />
                    </token-annotation>
            <text-annotation>
                        <annotator processor="p1" />
            </text-annotation>
            <sentence-annotation>
                        <annotator processor="p1" />
            </sentence-annotation>
            <paragraph-annotation>
                        <annotator processor="p1" />
            </paragraph-annotation>
            <syntax-annotation set="syntax"> <!-- an ad-hoc set -->
                        <annotator processor="p1" />
                    </syntax-annotation>
        </annotations>
        <provenance>
           <processor xml:id="p1" name="proycon" type="manual" />
        </provenance>
    </metadata>
    <text xml:id="example.text">
      <p xml:id="example.p.1">
          <s xml:id="example.p.1.s.1">
            <t>The Dalai Lama greeted him.</t>
            <w xml:id="example.p.1.s.1.w.1"><t>The</t></w>
            <w xml:id="example.p.1.s.1.w.2"><t>Dalai</t></w>
            <w xml:id="example.p.1.s.1.w.3"><t>Lama</t></w>
            <w xml:id="example.p.1.s.1.w.4"><t>greeted</t></w>
            <w xml:id="example.p.1.s.1.w.5" space="no"><t>him</t></w>
            <w xml:id="example.p.1.s.1.w.6"><t>.</t></w>
            <syntax>
              <su xml:id="example.p.1.s.1.su.1" class="s">
                <su xml:id="example.p.1.s.1.su.1_1" class="np">
                    <su xml:id="example.p.1.s.1.su.1_1_1" class="det">
                       <wref id="example.p.1.s.1.w.1" t="The" />
                    </su>
                    <su xml:id="example.p.1.s.1.su.1_1_2" class="pn">
                       <wref id="example.p.1.s.1.w.2" t="Dalai" />
                       <wref id="example.p.1.s.1.w.3" t="Lama" />
                    </su>
                </su>
                <su xml:id="example.p.1.s.1.su.1_2" class="vp">
                  <su xml:id="example.p.1.s.1.su.1_2_1" class="v">
                      <wref id="example.p.1.s.1.w.4" t="greeted" />
                  </su>
                  <su xml:id="example.p.1.s.1.su.1_2_2" class="pron">
                    <wref id="example.p.1.s.1.w.5" t="him" />
                  </su>
                </su>
              </su>
            </syntax>
          </s>
      </p>
    </text>
</FoLiA>
```

4.4.2 Chunking

Assigns shallow grammatical categories to spans of words. Unlike syntax annotation, chunks are not nestable. They are often produced by a process called Shallow Parsing, or alternatively, chunking.

Specification

Annotation Category *Span Annotation*

Declaration `<chunking-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since the beginning

Element `<chunk>`

API Class `Chunk` (FoLiApy API Reference)

Layer Element `<chunking>`

Span Role Elements

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `textclass` – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to `current`. See *Text class attribute (advanced)*.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.

- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<metric>` (*Metric Annotation*), `<relation>` (*Relation Annotation*)

Valid Context `<chunking>` (*Chunking*)

Explanation

Note: Please first ensure you are familiar with the general principles of *Span Annotation* to make sense of this annotation type.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
                           <annotator processor="p1" />
                    </token-annotation>
          <text-annotation>
                           <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                           <annotator processor="p1" />
          </sentence-annotation>
          <paragraph-annotation>
                           <annotator processor="p1" />
          </paragraph-annotation>
          <chunking-annotation set="chunkset"> <!-- an ad-hoc set -->
                           <annotator processor="p1" />
                    </chunking-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
```

(continues on next page)

(continued from previous page)

```
  </metadata>
  <text xml:id="example.text">
    <p xml:id="example.p.1">
        <s xml:id="example.p.1.s.1">
          <t>The Dalai Lama greeted him.</t>
          <w xml:id="example.p.1.s.1.w.1"><t>The</t></w>
          <w xml:id="example.p.1.s.1.w.2"><t>Dalai</t></w>
          <w xml:id="example.p.1.s.1.w.3"><t>Lama</t></w>
          <w xml:id="example.p.1.s.1.w.4"><t>greeted</t></w>
          <w xml:id="example.p.1.s.1.w.5" space="no"><t>him</t></w>
          <w xml:id="example.p.1.s.1.w.6"><t>.</t></w>
          <chunking>
            <chunk xml:id="example.p.1.s.1.chunk.1">
                <wref id="example.p.1.s.1.w.1" t="The" />
                <wref id="example.p.1.s.1.w.2" t="Dalai" />
                <wref id="example.p.1.s.1.w.3" t="Lama" />
            </chunk>
            <chunk xml:id="example.p.1.s.1.chunk.2">
                <wref id="example.p.1.s.1.w.4" t="greeted" />
            </chunk>
            <chunk xml:id="example.p.1.s.1.chunk.3">
                <wref id="example.p.1.s.1.w.5" t="him" />
                <wref id="example.p.1.s.1.w.6" t="." />
            </chunk>
          </chunking>
        </s>
    </p>
  </text>
</FoLiA>
```

4.4.3 Entity Annotation

Entity annotation is a broad and common category in FoLiA. It is used for specifying all kinds of multi-word expressions, including but not limited to named entities. The set definition used determines the vocabulary and therefore the precise nature of the entity annotation.

Specification

Annotation Category *Span Annotation*

Declaration `<entity-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since the beginning

Element `<entity>`

API Class `Entity` (FoLiApy API Reference)

Layer Element `<entities>`

Span Role Elements

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.

- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `textclass` – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to `current`. See *Text class attribute (advanced)*.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<metric>` (*Metric Annotation*), `<relation>` (*Relation Annotation*)

Valid Context `<entities>` (*Entity Annotation*)

Explanation

Note: Please first ensure you are familiar with the general principles of *Span Annotation* to make sense of this annotation type.

The `entities` layer offers a generic solution to encode various types of entities or *multi-word expressions*, including but not limited to named entities. The set used determines the precise semantics behind the entities.

This annotation type, being the simplest of all span annotations, is much used in FoLiA.

It is recommended, but not required, for each entity to have a unique identifier.

Examples

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl" format="text/turtle">
                              <annotator processor="p1" />
                       </token-annotation>
          <text-annotation>
                              <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                              <annotator processor="p1" />
          </sentence-annotation>
          <paragraph-annotation>
                              <annotator processor="p1" />
          </paragraph-annotation>
          <entity-annotation set="https://raw.githubusercontent.com/proycon/folia/
↪master/setdefinitions/namedentities.foliaset.ttl" format="text/turtle">
                              <annotator processor="p1" />
                       </entity-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <p xml:id="example.p.1">
      <s xml:id="example.p.1.s.1">
         <t>The Dalai Lama currently lives in Dharamsala in India.</t>
         <w xml:id="example.p.1.s.1.w.1" class="WORD">
            <t>The</t>
         </w>
         <w xml:id="example.p.1.s.1.w.2" class="WORD">
            <t>Dalai</t>
         </w>
         <w xml:id="example.p.1.s.1.w.3" class="WORD">
            <t>Lama</t>
         </w>
         <w xml:id="example.p.1.s.1.w.4" class="WORD">
            <t>currently</t>
         </w>
```

(continues on next page)

(continued from previous page)

```
            <w xml:id="example.p.1.s.1.w.5" class="WORD">
               <t>lives</t>
            </w>
            <w xml:id="example.p.1.s.1.w.6" class="WORD">
               <t>in</t>
            </w>
            <w xml:id="example.p.1.s.1.w.7" class="WORD">
               <t>Dharamsala</t>
            </w>
            <w xml:id="example.p.1.s.1.w.8" class="WORD">
               <t>in</t>
            </w>
            <w xml:id="example.p.1.s.1.w.9" class="WORD" space="no">
               <t>India</t>
            </w>
            <w xml:id="example.p.1.s.1.w.10" class="PUNCTUATION">
               <t>.</t>
            </w>
            <entities>
                <entity xml:id="example.p.1.s.1.entity.1" class="per">
                    <wref id="example.p.1.s.1.w.2" t="Dalai" />
                    <wref id="example.p.1.s.1.w.3" t="Lama" />
                </entity>
                <entity xml:id="example.p.1.s.1.entity.2" class="loc.city">
                    <wref id="example.p.1.s.1.w.7" t="Dharamsala" />
                </entity>
                <entity xml:id="example.p.1.s.1.entity.3" class="loc.country">
                    <wref id="example.p.1.s.1.w.9" t="India" />
                </entity>
            </entities>
         </s>
      </p>
   </text>
</FoLiA>
```

It is possible to associate inline annotations with span annotations, provided you declare the annotation type with groupannotations="yes". For entities, this is useful in case you have a more fine-grained tokenisation layer but want to associate certain information such as part-of-speech tags or lemmas with larger entities than tokens:

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
                         <annotator processor="p1" />
                  </token-annotation>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
          </sentence-annotation>
          <paragraph-annotation>
                         <annotator processor="p1" />
```

(continues on next page)

(continued from previous page)

```
          </paragraph-annotation>
          <entity-annotation groupannotations="yes">
                         <annotator processor="p1" />
                  </entity-annotation>
          <pos-annotation set="brown"> <!-- This is an ad-hoc set declaration as it is no URL and therefore not really defined -->
                         <annotator processor="p1" />
          </pos-annotation>
          <lemma-annotation set="english-adhoc"> <!-- This is an ad-hoc set declaration as it is no URL and therefore not really defined -->
                         <annotator processor="p1" />
          </lemma-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <p xml:id="example.p.1">
      <s xml:id="example.p.1.s.1">
         <t>The container-ship lost its cargo of bottle openers.</t>
         <w xml:id="example.p.1.s.1.w.1" class="WORD">
            <t>The</t>
            <pos class="AT" />
         </w>
         <w xml:id="example.p.1.s.1.w.2" class="WORD" space="no">
            <t>container</t>
         </w>
         <w xml:id="example.p.1.s.1.w.3" class="WORD" space="no">
            <t>-</t>
         </w>
         <w xml:id="example.p.1.s.1.w.4" class="WORD">
            <t>ship</t>
         </w>
         <w xml:id="example.p.1.s.1.w.5" class="WORD">
            <t>lost</t>
            <pos class="VBD" />
         </w>
         <w xml:id="example.p.1.s.1.w.6" class="WORD">
            <t>its</t>
            <pos class="PP$" />
         </w>
         <w xml:id="example.p.1.s.1.w.7" class="WORD">
            <t>cargo</t>
            <pos class="NN" />
         </w>
         <w xml:id="example.p.1.s.1.w.8" class="WORD">
             <t>of</t>
            <pos class="IN" />
         </w>
         <w xml:id="example.p.1.s.1.w.9" class="WORD">
            <t>bottle</t>
         </w>
         <w xml:id="example.p.1.s.1.w.10" class="WORD" space="no">
            <t>openers</t>
         </w>
```

(continues on next page)

(continued from previous page)

```
          <w xml:id="example.p.1.s.1.w.11" class="PUNCTUATION">
            <t>.</t>
          </w>
          <entities>
              <entity xml:id="example.p.1.s.1.entity.1">
                  <wref id="example.p.1.s.1.w.2" t="container" />
                  <wref id="example.p.1.s.1.w.3" t="-" />
                  <wref id="example.p.1.s.1.w.4" t="ship" />
                  <pos class="NN" />
                  <lemma class="container-ship" />
              </entity>
              <entity xml:id="example.p.1.s.1.entity.2">
                  <wref id="example.p.1.s.1.w.9" t="bottle" />
                  <wref id="example.p.1.s.1.w.10" t="openers" />
                  <pos class="NNS" />
                  <lemma class="bottle opener" />
              </entity>
          </entities>
       </s>
     </p>
   </text>
</FoLiA>
```

4.4.4 Dependency Annotation

Dependency relations are syntactic relations between spans of tokens. A dependency relation takes a particular class and consists of a single head component and a single dependent component.

Specification

Annotation Category *Span Annotation*

Declaration `<dependency-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Slightly revised since v0.8 (no `su` attribute on `hd`/`dep`)

Element `<dependency>`

API Class `Dependency` (FoLiApy API Reference)

Layer Element `<dependencies>`

Span Role Elements `<dep>` (`DependencyDependent`), `<hd>` (`Headspan`)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.

- class – The class of the annotation, i.e. the annotation tag in the vocabulary defined by set.
- processor – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- annotator – This is an older alternative to the processor attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- annotatortype – This is an older alternative to the processor attribute, without support for full provenance. It is used together with annotator and specific the type of the annotator, either manual for human annotators or auto for automated systems.
- confidence – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- datetime – The date and time when this annotation was recorded, the format is YYYY-MM-DDThh:mm:ss (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- n – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- textclass – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to current. See *Text class attribute (advanced)*.
- src – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- begintime – A timestamp in HH:MM:SS.MMM format, indicating the begin time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- endtime – A timestamp in HH:MM:SS.MMM format, indicating the end time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- speaker – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data <comment> (*Comment Annotation*), <desc> (*Description Annotation*), <metric> (*Metric Annotation*), <relation> (*Relation Annotation*)

Valid Context <dependencies> (*Dependency Annotation*)

Explanation

Note: Please first ensure you are familiar with the general principles of *Span Annotation* to make sense of this annotation type.

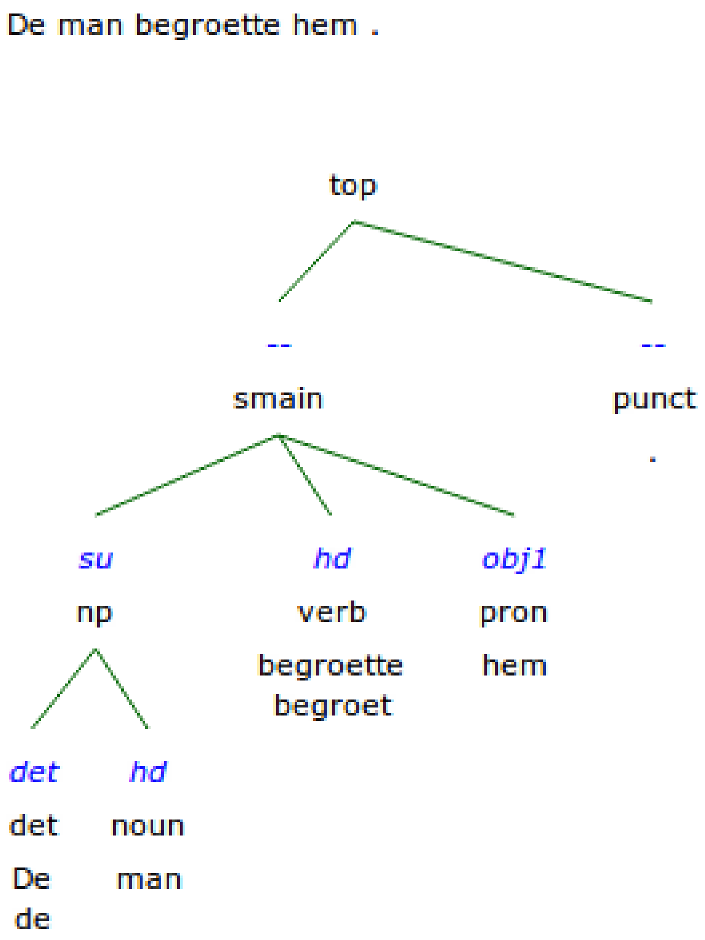
Dependency relations are syntactic relations between spans of tokens. A dependency relation takes a particular class and consists of a single head component and a single dependent component. In the sample *"He sees"*, there is syntactic dependency between the two words: *"sees"* is the head, and *"He"* is the dependant, and the relation can be qualified as something like *subject*, as the dependant is the subject of the head word. Each dependency relation is explicitly noted in FoLiA.

The element `<dependencies>` introduces this annotation layer. Within it, `<dependency>` elements describe all dependency pairs. The `<dependency>` element always contains two *span roles*: one head element (`<hd>`) and one dependent element (`<dep>`). Within these span roles, the words referenced in the usual stand-off fashion, using `<wref>`.

Example

In the example below, we show a Dutch sentence parsed with the Alpino Parser.

For a better understanding, The following figure illustrates the syntactic parse with the dependency relations (blue).



We show not only the dependency layer, but also the syntax layer (*Syntactic Annotation*) to which it is related.

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
```

(continues on next page)

(continued from previous page)

```
                              <annotator processor="p1" />
                       </token-annotation>
           <text-annotation>
                              <annotator processor="p1" />
           </text-annotation>
           <sentence-annotation>
                              <annotator processor="p1" />
           </sentence-annotation>
           <paragraph-annotation>
                              <annotator processor="p1" />
           </paragraph-annotation>
           <dependency-annotation set="alpino-dependencies"> <!-- an ad-hoc set -->
                              <annotator processor="p2" />
                       </dependency-annotation>
           <syntax-annotation set="alpino-constituents"> <!-- an ad-hoc set -->
                              <annotator processor="p2" />
                       </syntax-annotation>
       </annotations>
       <provenance>
          <processor xml:id="p1" name="proycon" type="manual" />
          <processor xml:id="p2" name="alpino" />
       </provenance>
   </metadata>
   <text xml:id="example.text">
     <p xml:id="example.p.1">
         <s xml:id="example.p.1.s.1">
           <t>De man begroette hem.</t>
           <w xml:id="example.p.1.s.1.w.1"><t>De</t></w>
           <w xml:id="example.p.1.s.1.w.2"><t>man</t></w>
           <w xml:id="example.p.1.s.1.w.3"><t>begroette</t></w>
           <w xml:id="example.p.1.s.1.w.4" space="no"><t>hem</t></w>
           <w xml:id="example.p.1.s.1.w.5"><t>.</t></w>
           <dependencies>
             <dependency xml:id="example.p.1.s.1.dependency.1" class="su">
                 <hd>
                     <wref id="example.p.1.s.1.w.3" t="begroette"/>
                 </hd>
                 <dep>
                     <wref id="example.p.1.s.1.w.2" t="man" />
                 </dep>
             </dependency>
             <dependency xml:id="example.p.1.s.1.dependency.3" class="obj1">
                 <hd>
                     <wref id="example.p.1.s.1.w.3" t="begroette"/>
                 </hd>
                 <dep>
                     <wref id="example.p.1.s.1.w.4" t="hem" />
                 </dep>
             </dependency>
             <dependency xml:id="example.p.1.s.1.dependency.2" class="det">
                 <hd>
                    <wref id="example.p.1.s.1.w.2" t="man" />
                 </hd>
                 <dep>
                     <wref id="example.p.1.s.1.w.1" t="De" />
                 </dep>
```

(continues on next page)

(continued from previous page)

```
            </dependency>
          </dependencies>
          <syntax>
            <su xml:id="example.p.1.s.1.su.1" class="top">
                <su xml:id="example.p.1.s.1.su.1_1" class="smain">
                    <su xml:id="example.p.1.s.1.su.1_1_1" class="np">
                        <su xml:id="example.p.1.s.1.su.1_1_1_1" class="top">
                            <wref id="example.p.1.s.1.w.1" t="De" />
                        </su>
                        <su xml:id="example.p.1.s.1.su.1_1_1_2" class="top">
                            <wref id="example.p.1.s.1.w.2" t="man" />
                        </su>
                    </su>
                    <su xml:id="example.p.1.s.1.su.1_1_2" class="verb">
                        <wref id="example.p.1.s.1.w.3" t="begroette" />
                    </su>
                    <su xml:id="example.p.1.s.1.su.1_1_3" class="pron">
                        <wref id="example.p.1.s.1.w.4" t="hem" />
                    </su>
                </su>
                <su xml:id="example.p.1.s.1.su.1_2" class="punct">
                    <wref id="example.p.1.s.1.w.5" t="." />
                </su>
            </su>
          </syntax>
        </s>
    </p>
  </text>
</FoLiA>
```

4.4.5 Time Segmentation

FoLiA supports time segmentation to allow for more fine-grained control of timing information by associating spans of words/tokens with exact timestamps. It can provide a more linguistic alternative to *Event Annotation*.

Specification

Annotation Category *Span Annotation*

Declaration `<timesegment-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since v0.8 but renamed since v0.9

Element `<timesegment>`

API Class `TimeSegment` (FoLiApy API Reference)

Layer Element `<timing>`

Span Role Elements

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.

- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `textclass` – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to `current`. See *Text class attribute (advanced)*.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<metric>` (*Metric Annotation*), `<relation>` (*Relation Annotation*)

Valid Context `<timing>` (*Time Segmentation*)

Feature subsets (extra attributes)

- `actor`
- `begindatetime`

- enddatetime

Explanation

FoLiA supports time segmentation using the `<timing>` layer and the `<timesegment>` span annotation element. This element is useful for speech, but can also be used for event annotation. We already saw events as structure annotation in *Event Annotation*, but for more fine-grained control of timing information a span annotation element in an offset layer is more suited.

Time segments may also be nested. The predefined and optional feature subset `begindatetime` and `enddatetime` can be used express the exact moment at which an event started or ended. These too are set-defined so the format shown here is just an example.

If you are only interested in a structural annotation of events, and a coarser level of annotation suffices, then use :ref:*event_annotation*.

If used in a **speech context**, all the generic speech attributes become available (See speech). This introduces `begintime` and `endtime`, which are different from the `begindatetime` and `enddatetime` feature subsets introduced by this annotation type! The generic attributes `begintime` and `endtime` are not defined by a set, but specify a time location in `HH:MM:SS.MMM` format which may refer to the location in an associated audio file. Audio files are associated using the `src` attribute, which is inherited by all lower elements, so we put it on the sentence here.

See also:

- *Event Annotation*
- speech

Example

The following example illustrates the usage of time segmentation for event annotation:

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
                         <annotator processor="p1" />
                 </token-annotation>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
          </sentence-annotation>
          <paragraph-annotation>
                         <annotator processor="p1" />
          </paragraph-annotation>
          <timesegment-annotation set="events"> <!-- an ad-hoc set -->
                         <annotator processor="p1" />
                 </timesegment-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
```

(continues on next page)

(continued from previous page)

```
    <p xml:id="example.p.1">
        <s xml:id="example.p.1.s.1">
          <w xml:id="example.p.1.s.1.w.1"><t>I</t></w>
          <w xml:id="example.p.1.s.1.w.2"><t>think</t></w>
          <w xml:id="example.p.1.s.1.w.3"><t>I</t></w>
          <w xml:id="example.p.1.s.1.w.4"><t>have</t></w>
          <w xml:id="example.p.1.s.1.w.5"><t>to</t></w>
          <w xml:id="example.p.1.s.1.w.6"><t>go</t></w>
          <w xml:id="example.p.1.s.1.w.7"><t>.</t></w>
          <timing>
            <timesegment class="utterance" begindatetime="2011-12-15T19:01"
              enddatetime="2011-12-15T19:03" actor="myself">
                <wref id="example.p.1.s.1.w.1" t="I" />
                <wref id="example.p.1.s.1.w.2" t="think" />
            </timesegment>
            <timesegment class="cough" begindatetime="2011-12-15T19:03"
              enddatetime="2011-12-15T19:05" actor="myself">
            </timesegment>
            <timesegment class="utterance" begindatetime="2011-12-15T19:05"
              enddatetime="2011-12-15T19:06" actor="myself">
                <wref id="example.p.1.s.1.w.3" t="I" />
                <wref id="example.p.1.s.1.w.4" t="have" />
                <wref id="example.p.1.s.1.w.5" t="to" />
                <wref id="example.p.1.s.1.w.6" t="go" />
            </timesegment>
          </timing>
        </s>
    </p>
  </text>
</FoLiA>
```

Example in a speech context

The following example illustrates the usage of time segmentation in a speech context. You have to be aware though, that the `begintime` and `endtime` attributes can also be directly associated with any structure elements in a speech context, making the use of this annotation type unnecessary or redundant if used this way.

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
                         <annotator processor="p1" />
                  </token-annotation>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <utterance-annotation>
                         <annotator processor="p1" />
          </utterance-annotation>
          <timesegment-annotation set="events"> <!-- an ad-hoc set -->
                         <annotator processor="p1" />
                  </timesegment-annotation>
```

(continues on next page)

(continued from previous page)

```
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <speech xml:id="example.speech">
    <utt src="ithinkihavetogo.mp3">
     <w xml:id="example.utt.1.w.1"><t>I</t></w>
     <w xml:id="example.utt.1.w.2"><t>think</t></w>
     <w xml:id="example.utt.1.w.3"><t>I</t></w>
     <w xml:id="example.utt.1.w.4"><t>have</t></w>
     <w xml:id="example.utt.1.w.5"><t>to</t></w>
     <w xml:id="example.utt.1.w.6"><t>go</t></w>
     <w xml:id="example.utt.1.w.7"><t>.</t></w>
     <timing>
      <timesegment begintime="00:00:00.000" endtime="00:00:00.250">
        <wref id="example.utt.1.w.1" t="I" />
      </timesegment>
      <timesegment begintime="00:00:00.250" endtime="00:00:00.500">
        <wref id="example.utt.1.w.2" t="think" />
      </timesegment>
      <timesegment begintime="00:00:00.500" endtime="00:00:00.750">
        <wref id="example.utt.1.w.3" t="I" />
      </timesegment>
      <timesegment begintime="00:00:00.750" endtime="00:00:01.000">
        <wref id="example.utt.1.w.4" t="have" />
      </timesegment>
      <timesegment begintime="00:00:01.000" endtime="00:00:01.250">
        <wref id="example.utt.1.w.5" t="to" />
      </timesegment>
      <timesegment begintime="00:00:01.250" endtime="00:00:01.500">
        <wref id="example.utt.1.w.6" t="go" />
      </timesegment>
     </timing>
    </utt>
  </speech>
</FoLiA>
```

4.4.6 Coreference Annotation

Relations between words that refer to the same referent (anaphora) are expressed in FoLiA using Coreference Annotation. The co-reference relations are expressed by specifying the entire chain in which all links are coreferent.

Specification

Annotation Category *Span Annotation*

Declaration `<coreference-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History since v0.9

Element `<coreferencechain>`

API Class `CoreferenceChain` (FoLiApy API Reference)

Layer Element `<coreferences>`

Span Role Elements `<coreferencelink>` (`CoreferenceLink`)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `textclass` – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to `current`. See *Text class attribute (advanced)*.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they

use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<coreferencelink>` (*Coreference Annotation*), `<desc>` (*Description Annotation*), `<metric>` (*Metric Annotation*), `<relation>` (*Relation Annotation*)

Valid Context `<coreferences>` (*Coreference Annotation*)

Explanation

Note: Please first ensure you are familiar with the general principles of *Span Annotation* to make sense of this annotation type.

Relations between words that refer to the same referent are expressed in FoLiA using the `<coreferencechain>` span annotation element and the `<coreferencelink>` span role within it for each instance.

The co-reference relations are expressed by specifying the entire chain in which all links are coreferent. The head of a coreferent may optionally be marked with the `<hd>` element, another span role.

As always, this annotation layer itself may be embedded on whatever level is preferred. The following example uses paragraph level, but you can for instance also embed it at sentence level or a global text level:

The `coreferencelink` may take three attributes, which are actually predefined feature subsets (See *Features*), their values depend on the set used and are thus user-definable and never predefined:

- `mod` - A subset that can be used to indicate that there is modality or negation in this coreference link.
- `time` - A subset used to indicate a time dependency. An example of a time dependency is seen in the sentence: *"Bert De Graeve, until recently CEO, will now take up a position as CFO"*. Here

"Bert De Graeve", "CEO" and "CFO" would all be part of the same coreference chain, and the second coreferencelink ("CEO") can be marked as being in the past using the "time" attribute. * `level` - A subset used that can indicate the level on which the coreference holds. A possible value suggestion could be `sense`, indicating that only on sense-level there is a coreference relation, as opposed to an actual reference.

Example

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
                          <annotator processor="p1" />
                  </token-annotation>
          <text-annotation>
                          <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                          <annotator processor="p1" />
          </sentence-annotation>
          <paragraph-annotation>
                          <annotator processor="p1" />
          </paragraph-annotation>
          <coreference-annotation set="adhoc"> <!-- an ad-hoc set -->
                          <annotator processor="p1" />
                  </coreference-annotation>
```

(continues on next page)

(continued from previous page)

```
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <p xml:id="example.p.1">
        <s xml:id="example.p.1.s.1">
          <t>The Dalai Lama greeted him.</t>
          <w xml:id="example.p.1.s.1.w.1"><t>The</t></w>
          <w xml:id="example.p.1.s.1.w.2"><t>Dalai</t></w>
          <w xml:id="example.p.1.s.1.w.3"><t>Lama</t></w>
          <w xml:id="example.p.1.s.1.w.4"><t>greeted</t></w>
          <w xml:id="example.p.1.s.1.w.5" space="no"><t>him</t></w>
          <w xml:id="example.p.1.s.1.w.6"><t>.</t></w>
        </s>
        <s xml:id="example.p.1.s.2">
          <t>He was happy to see him.</t>
          <w xml:id="example.p.1.s.2.w.1"><t>He</t></w>
          <w xml:id="example.p.1.s.2.w.2"><t>was</t></w>
          <w xml:id="example.p.1.s.2.w.3"><t>happy</t></w>
          <w xml:id="example.p.1.s.2.w.4"><t>to</t></w>
          <w xml:id="example.p.1.s.2.w.5"><t>see</t></w>
          <w xml:id="example.p.1.s.2.w.6" space="no"><t>him</t></w>
          <w xml:id="example.p.1.s.2.w.7"><t>.</t></w>
        </s>
        <s xml:id="example.p.1.s.3">
          <t>He smiled.</t>
          <w xml:id="example.p.1.s.3.w.1"><t>He</t></w>
          <w xml:id="example.p.1.s.3.w.2" space="no"><t>smiled</t></w>
          <w xml:id="example.p.1.s.3.w.3"><t>.</t></w>
        </s>
        <coreferences>
            <coreferencechain class="dalailama">
              <coreferencelink>
                  <wref id="example.p.1.s.1.w.1" t="The" />
                  <hd> <!-- extra span role to mark the head -->
                    <wref id="example.p.1.s.1.w.2" t="Dalai" />
                    <wref id="example.p.1.s.1.w.3" t="Lama" />
                  </hd>
              </coreferencelink>
              <coreferencelink>
                <wref id="example.p.1.s.2.w.1" t="he" />
              </coreferencelink>
            </coreferencechain>
            <coreferencechain class="dalailama">
              <coreferencelink>
                <wref id="example.p.1.s.1.w.5" t="him" />
              </coreferencelink>
              <coreferencelink>
                <wref id="example.p.1.s.2.w.6" t="him" />
              </coreferencelink>
              <coreferencelink>
                <wref id="example.p.1.s.3.w.1" t="He" />
              </coreferencelink>
            </coreferencechain>
```

(continues on next page)

(continued from previous page)

```
        </coreferences>
      </p>
    </text>
</FoLiA>
```

4.4.7 Semantic Role Annotation

This span annotation type allows for the expression of semantic roles, or thematic roles. It is often used together with *Predicate Annotation*

Specification

Annotation Category *Span Annotation*

Declaration `<semrole-annotation set="...">` *(note: set is mandatory)*

Version History since v0.9, revised since v1.3 (added predicates)

Element `<semrole>`

API Class `SemanticRole` (FoLiApy API Reference)

Layer Element `<semroles>`

Span Role Elements `<hd>` (`Headspan`)

Required Attributes

- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.

- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `textclass` – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to `current`. See *Text class attribute (advanced)*.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<metric>` (*Metric Annotation*), `<relation>` (*Relation Annotation*)

Valid Context `<predicate>` (*Predicate Annotation*), `<semroles>` (*Semantic Role Annotation*)

Explanation

Note: Please first ensure you are familiar with the general principles of *Span Annotation* to make sense of this annotation type.

Semantic roles are usually embedded within the `<predicate>` span annotation element (see *Predicate Annotation*, since FoLiA v1.3). This is a *separate* span annotation element, which itself may also take a class and has its own declaration. Such a class can for instance be used to describe frame semantics, such as FrameNet.

Semantic roles without predicates are also allowed, but less expressive as relations between the semantic roles are not explicit. The reverse also hold, you can do predicate annotation without semantic role labelling.

See also:

Predicate Annotation

Example

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
```

(continues on next page)

(continued from previous page)

```
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
                         <annotator processor="p1" />
                  </token-annotation>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
          </sentence-annotation>
          <paragraph-annotation>
                         <annotator processor="p1" />
          </paragraph-annotation>
          <semrole-annotation set="semroleset"> <!-- an ad-hoc set -->
                         <annotator processor="p1" />
                  </semrole-annotation>
          <predicate-annotation set="semroleset"> <!-- an ad-hoc set -->
                         <annotator processor="p1" />
                  </predicate-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <p xml:id="example.p.1">
        <s xml:id="example.p.1.s.1">
          <t>The Dalai Lama greeted him.</t>
          <w xml:id="example.p.1.s.1.w.1"><t>The</t></w>
          <w xml:id="example.p.1.s.1.w.2"><t>Dalai</t></w>
          <w xml:id="example.p.1.s.1.w.3"><t>Lama</t></w>
          <w xml:id="example.p.1.s.1.w.4"><t>greeted</t></w>
          <w xml:id="example.p.1.s.1.w.5" space="no"><t>him</t></w>
          <w xml:id="example.p.1.s.1.w.6"><t>.</t></w>
          <semroles>
           <predicate class="greet">
            <semrole class="agent">
                <wref id="example.p.1.s.1.w.2" />
                <wref id="example.p.1.s.1.w.3" />
            </semrole>
            <semrole class="patient">
                <wref id="example.p.1.s.1.w.5" />
            </semrole>
           </predicate>
          </semroles>
        </s>
    </p>
  </text>
</FoLiA>
```

4.4.8 Predicate Annotation

Allows annotation of predicates, this annotation type is usually used together with Semantic Role Annotation. The types of predicates are defined by a user-defined set definition.

Specification

Annotation Category *Span Annotation*

Declaration `<predicate-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History since v1.3

Element `<predicate>`

API Class `Predicate` (FoLiApy API Reference)

Layer Element `<None>`

Span Role Elements

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `textclass` – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to `current`. See *Text class attribute (advanced)*.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.

- speaker – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<metric>` (*Metric Annotation*), `<relation>` (*Relation Annotation*), `<semrole>` (*Semantic Role Annotation*)

Valid Context `<semroles>` (*Semantic Role Annotation*)

Explanation

Please see *Semantic Role Annotation* for an explanation of predicates in the context of semantic role labelling.

4.4.9 Observation Annotation

Observation annotation is used to make an observation pertaining to one or more word tokens. Observations offer a an external qualification on part of a text. The qualification is expressed by the class, in turn defined by a set. The precise semantics of the observation depends on the user-defined set.

Specification

Annotation Category *Span Annotation*

Declaration `<observation-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History since v1.3

Element `<observation>`

API Class `Observation` (FoLiApy API Reference)

Layer Element `<observations>`

Span Role Elements

Required Attributes

Optional Attributes

- xml:id – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- set – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The set must be referred to also in the *Annotation Declarations* for this annotation type.
- class – The class of the annotation, i.e. the annotation tag in the vocabulary defined by set.

- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `textclass` – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to `current`. See *Text class attribute (advanced)*.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<metric>` (*Metric Annotation*), `<relation>` (*Relation Annotation*)

Valid Context `<observations>` (*Observation Annotation*)

Explanation

Note: Please first ensure you are familiar with the general principles of *Span Annotation* to make sense of this annotation type.

The `<observation>` element is a span annotation element that makes an observation pertaining to one or more word tokens. It is embedded in an `observations` layer.

Observations offer a an external qualification on part of a text. The qualification is expressed by the class, in turn defined by a set. The precise semantics of the observation depends on the user-defined set.

The element may for example act to mark errors in the text or to capture observations from teachers/proofreaders.

Example

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
                         <annotator processor="p1" />
                  </token-annotation>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
          </sentence-annotation>
          <paragraph-annotation>
                         <annotator processor="p1" />
          </paragraph-annotation>
          <observation-annotation set="errordetection"> <!-- an ad-hoc set -->
                         <annotator processor="p2" />
                  </observation-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="student" type="manual" />
         <processor xml:id="p2" name="teacher" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <p xml:id="example.p.1">
        <s xml:id="example.p.1.s.1">
         <t>The Dalai Lama greated him.</t>
         <w xml:id="example.p.1.s.1.w.1"><t>The</t></w>
         <w xml:id="example.p.1.s.1.w.2"><t>Dalai</t></w>
         <w xml:id="example.p.1.s.1.w.3"><t>Lama</t></w>
         <w xml:id="example.p.1.s.1.w.4"><t>greated</t></w>
         <w xml:id="example.p.1.s.1.w.5" space="no"><t>him</t></w>
         <w xml:id="example.p.1.s.1.w.6"><t>.</t></w>
         <observations>
          <observation class="typo">
           <wref id="example.p.1.s.1.w.4"/>
          </observation>
         </observations>
        </s>
    </p>
  </text>
</FoLiA>
```

4.4.10 Sentiment Annotation

Sentiment analysis marks subjective information such as sentiments or attitudes expressed in text. The sentiments/attitudes are defined by a user-defined set definition.

Note: This annotation type is deprecated because it overlaps with modality annotation (_modality_annotation). Modality annotation is now preferred over sentiment annotation, as it is more generic.

Specification

Annotation Category *Span Annotation*

Declaration `<sentiment-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History since v1.3

Element `<sentiment>`

API Class `Sentiment` (FoLiApy API Reference)

Layer Element `<sentiments>`

Span Role Elements `<hd>` (`Headspan`), `<source>` (`Source`), `<target>` (`Target`)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `textclass` – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to `current`. See *Text class attribute (advanced)*.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.

- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<metric>` (*Metric Annotation*), `<relation>` (*Relation Annotation*)

Valid Context `<sentiments>` (*Sentiment Annotation*)

Feature subsets (extra attributes)

- `polarity`
- `strength`

Explanation

Note: Please first ensure you are familiar with the general principles of *Span Annotation* to make sense of this annotation type.

Sentiment analysis marks subjective information such as sentiments or attitudes expressed in text. The `<sentiment>` span annotation element is used to this end. It is embedded in a `<sentiments>` layer.

The `<sentiment>` element takes the following span roles:

- `<hd>` – (required) – The head of the sentiment; expresses the actual sentiment, it covers word spans such as "happy", "very satisfied", "highly dissappointed".
- `<source>` – (optional) – The source/holder of the sentiment, assuming it is explicitly expressed in the text.
- `<target>` – (optional) – The target/recipient of the sentiment, assuming it is explicitly expressed in the text.

The following feature subsets are predefined (see *Features*), whether they are actually used depends on the set, their values (classes) are set-dependent as well:

- `polarity` – Expresses the whether the sentiment is positive, neutral or negative.
- `strength` – Expresses the strength or intensity of the sentiment.

Besides these predefined features, FoLiA's feature mechanism can be used to associate other custom properties with any sentiment.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
                         <annotator processor="p1" />
                  </token-annotation>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
          </sentence-annotation>
          <sentiment-annotation set="sentiments"> <!-- an ad-hoc set -->
                         <annotator processor="p1" />
                  </sentiment-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <s xml:id="s1">
     <w xml:id="s1.w1"><t>He</t></w>
     <w xml:id="s1.w2"><t>is</t></w>
     <w xml:id="s1.w3"><t>happy</t></w>
     <w xml:id="s1.w4"><t>to</t></w>
     <w xml:id="s1.w5"><t>see</t></w>
     <w xml:id="s1.w6"><t>him</t></w>
     <w xml:id="s1.w7"><t>.</t></w>
     <sentiments>
      <sentiment class="emotion.joy" polarity="positive" strength="moderate">
        <source>
          <wref id="s1.w1" t="he" />
        </source>
        <target>
          <wref id="s1.w6" t="him" />
        </target>
        <hd>
          <wref id="s1.w3" t="happy" />
        </hd>
      </sentiment>
     </sentiments>
    </s>
  </text>
</FoLiA>
```

4.4.11 Statement Annotation

Statement annotation, sometimes also refered to as attribution, allows to decompose statements into the source of the statement, the content of the statement, and the way these relate, provided these are made explicit in the text.

Specification

Annotation Category *Span Annotation*

Declaration `<statement-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History since v1.3

Element `<statement>`

API Class `Statement` (FoLiApy API Reference)

Layer Element `<statements>`

Span Role Elements `<hd>` (`Headspan`), `<source>` (`Source`), `<rel>` (`StatementRelation`)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `textclass` – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to `current`. See *Text class attribute (advanced)*.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.

- speaker – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data <comment> (*Comment Annotation*), <desc> (*Description Annotation*), <metric> (*Metric Annotation*), <relation> (*Relation Annotation*)

Valid Context <statements> (*Statement Annotation*)

Explanation

The span annotation element <statement> allows to decompose statements into the source of the statement, the content of the statement, and the way these relate, provided these are made explicit in the text. It can be used to annotate attribution (who said what etc). The element is used in a <statements> layer and takes the following span roles:

- <hd> – (required) – The head of the statement is the actual content of the statement; this role spans the words containing the statement.
- <source> – (optional) – The source/holder of the statement, assuming it is explicitly expressed in the text.
- <rel> – (optional) – The relation between the source of the statement and the statement, this usually encompasses verbs like "to say", "to think", or prepositional phrases such as "according to". (not to be confused with *Relation Annotation*)

Example

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
                          <annotator processor="p1" />
                  </token-annotation>
          <text-annotation>
                          <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                          <annotator processor="p1" />
          </sentence-annotation>
          <paragraph-annotation>
                          <annotator processor="p1" />
          </paragraph-annotation>
          <statement-annotation set="attributionset"> <!-- an ad-hoc set -->
                          <annotator processor="p1" />
                  </statement-annotation>
      </annotations>
      <provenance>
```

(continues on next page)

(continued from previous page)

```
        <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <s xml:id="s1">
     <w xml:id="s1.w1"><t>They</t></w>
     <w xml:id="s1.w2"><t>said</t></w>
     <w xml:id="s1.w3"><t>the</t></w>
     <w xml:id="s1.w4"><t>hotel</t></w>
     <w xml:id="s1.w5"><t>was</t></w>
     <w xml:id="s1.w6"><t>a</t></w>
     <w xml:id="s1.w7"><t>nightmare</t></w>
     <w xml:id="s1.w8"><t>.</t></w>
     <statements>
      <statement class="said">
       <source>
        <wref id="s1.w1" />
       </source>
       <hd>
        <wref id="s1.w3" />
        <wref id="s1.w4" />
        <wref id="s1.w5" />
        <wref id="s1.w6" />
        <wref id="s1.w7" />
       </hd>
       <rel>
         <wref id="s1.w2" />
       </rel>
      </statement>
     </statements>
    </s>
  </text>
</FoLiA>
```

4.4.12 Modality Annotation

Modality annotation is used to describe the relationship between cue word(s) and the scope it covers. It is primarily used for the annotation of negation, but also for the annotation of factuality, certainty and truthfulness:.

Note: This annotation type has overlap with sentiment annotation (_sentiment_annotation). Modality annotation is now preferred over sentiment annotation, as it is more generic.

Specification

Annotation Category *Span Annotation*

Declaration `<modality-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since v2.4.0

Element `<modality>`

API Class `Modality` (FoLiApy API Reference)

Layer Element `<modalities>`

Span Role Elements `<cue>` (Cue), `<scope>` (Scope), `<source>` (Source), `<target>` (Target)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `textclass` – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to `current`. See *Text class attribute (advanced)*.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they

use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<metric>` (*Metric Annotation*), `<relation>` (*Relation Annotation*)

Valid Context `<modalities>` (*Modality Annotation*)

Feature subsets (extra attributes)

- `polarity`
- `strength`

Explanation

Note: Please first ensure you are familiar with the general principles of *Span Annotation* to make sense of this annotation type.

Modality analysis marks things such as sentiments, truthfulness, negation, doubt. The `<modality>` span annotation element is used to this end. It is embedded in a `<modalities>` layer.

The `<modalities>` element takes the following span roles:

- `<cue>` – (required) – The cue or trigger of the modality. In case of sentiments, this expresses the actual sentiment and could cover word spans such as "happy", "very satisfied", "highly dissappointed". This may also be nested inside `<scope>`.
- `<scope>` – (optional) – The scope of the modality. In case of negation for example, this covers the text that is negated.
- `<source>` – (optional) – The source/holder of the modality, assuming it is explicitly expressed in the text. This may also be nested inside `<scope>`.
- `<target>` – (optional) – The target/recipient of the modality, assuming it is explicitly expressed in the text. This may also be nested inside `<scope>`.

The following feature subsets are predefined (see *Features*), whether they are actually used depends on the set, their values (classes) are set-dependent as well:

- `polarity` – Expresses the whether the sentiment is positive, neutral or negative.
- `strength` – Expresses the strength or intensity of the sentiment.

Besides these predefined features, FoLiA's feature mechanism can be used to associate other custom properties with any sentiment.

Example

An example of sentiment analysis:

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
                          <annotator processor="p1" />
                  </token-annotation>
          <text-annotation>
                          <annotator processor="p1" />
```

(continues on next page)

(continued from previous page)

```
          </text-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
          </sentence-annotation>
          <modality-annotation set="sentiments"> <!-- an ad-hoc set -->
                         <annotator processor="p1" />
                 </modality-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <s xml:id="s1">
     <w xml:id="s1.w1"><t>He</t></w>
     <w xml:id="s1.w2"><t>is</t></w>
     <w xml:id="s1.w3"><t>happy</t></w>
     <w xml:id="s1.w4"><t>to</t></w>
     <w xml:id="s1.w5"><t>see</t></w>
     <w xml:id="s1.w6"><t>him</t></w>
     <w xml:id="s1.w7"><t>.</t></w>
     <modalities>
      <modality class="emotion.joy" polarity="positive" strength="moderate">
        <source>
          <wref id="s1.w1" t="he" />
        </source>
        <target>
          <wref id="s1.w6" t="him" />
        </target>
        <cue>
          <wref id="s1.w3" t="happy" />
        </cue>
      </modality>
     </modalities>
    </s>
  </text>
</FoLiA>
```

An example of negation annotation:

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
                         <annotator processor="p1" />
                 </token-annotation>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
          </sentence-annotation>
          <modality-annotation set="modalities"> <!-- an ad-hoc set -->
                         <annotator processor="p1" />
```

(continues on next page)

(continued from previous page)

```
                  </modality-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <s xml:id="s1">
     <w xml:id="s1.w1"><t>I</t></w>
     <w xml:id="s1.w2"><t>did</t></w>
     <w xml:id="s1.w3"><t>not</t></w>
     <w xml:id="s1.w4"><t>know</t></w>
     <w xml:id="s1.w5"><t>who</t></w>
     <w xml:id="s1.w6"><t>you</t></w>
     <w xml:id="s1.w7"><t>were</t></w>
     <modalities>
      <modality class="negation">
        <cue>
          <wref id="s1.w3" t="not" />
        </cue>
        <scope>
          <wref id="s1.w1" t="I" />
          <wref id="s1.w2" t="did" />
          <wref id="s1.w4" t="know" />
          <wref id="s1.w5" t="who" />
          <wref id="s1.w6" t="you" />
          <wref id="s1.w7" t="were" />
        </scope>
      </modality>
     </modalities>
    </s>
  </text>
</FoLiA>
```

4.4.13 Group Annotations: Inline Annotations on Span Annotations

It is possible to directly apply inline annotations (see *Inline Annotation*) to span annotations, which allows for example to assign a part-of-speech tag or lemma directly to an entity, rather than to a word (`<w>`) as is customary. This functionality, however, needs to be explicitly enabled by adding the `groupannotations=yes` attribute to the declaration, as it adds extra complexity to a FoLiA document and in this way informs parsers to be aware of this.

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
                         <annotator processor="p1" />
                  </token-annotation>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
```

(continues on next page)

(continued from previous page)

```
          </sentence-annotation>
          <paragraph-annotation>
                         <annotator processor="p1" />
          </paragraph-annotation>
          <entity-annotation groupannotations="yes">
                         <annotator processor="p1" />
                 </entity-annotation>
          <pos-annotation set="brown"> <!-- This is an ad-hoc set declaration as it␣
↪is no URL and therefore not really defined -->
                         <annotator processor="p1" />
          </pos-annotation>
          <lemma-annotation set="english-adhoc"> <!-- This is an ad-hoc set␣
↪declaration as it is no URL and therefore not really defined -->
                         <annotator processor="p1" />
          </lemma-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <p xml:id="example.p.1">
      <s xml:id="example.p.1.s.1">
         <t>The container-ship lost its cargo of bottle openers.</t>
         <w xml:id="example.p.1.s.1.w.1" class="WORD">
            <t>The</t>
            <pos class="AT" />
         </w>
         <w xml:id="example.p.1.s.1.w.2" class="WORD" space="no">
            <t>container</t>
         </w>
         <w xml:id="example.p.1.s.1.w.3" class="WORD" space="no">
            <t>-</t>
         </w>
         <w xml:id="example.p.1.s.1.w.4" class="WORD">
            <t>ship</t>
         </w>
         <w xml:id="example.p.1.s.1.w.5" class="WORD">
            <t>lost</t>
            <pos class="VBD" />
         </w>
         <w xml:id="example.p.1.s.1.w.6" class="WORD">
            <t>its</t>
            <pos class="PP$" />
         </w>
         <w xml:id="example.p.1.s.1.w.7" class="WORD">
            <t>cargo</t>
            <pos class="NN" />
         </w>
         <w xml:id="example.p.1.s.1.w.8" class="WORD">
             <t>of</t>
            <pos class="IN" />
         </w>
         <w xml:id="example.p.1.s.1.w.9" class="WORD">
            <t>bottle</t>
         </w>
```

(continues on next page)

(continued from previous page)

```
          <w xml:id="example.p.1.s.1.w.10" class="WORD" space="no">
            <t>openers</t>
          </w>
          <w xml:id="example.p.1.s.1.w.11" class="PUNCTUATION">
            <t>.</t>
          </w>
          <entities>
              <entity xml:id="example.p.1.s.1.entity.1">
                  <wref id="example.p.1.s.1.w.2" t="container" />
                  <wref id="example.p.1.s.1.w.3" t="-" />
                  <wref id="example.p.1.s.1.w.4" t="ship" />
                  <pos class="NN" />
                  <lemma class="container-ship" />
              </entity>
              <entity xml:id="example.p.1.s.1.entity.2">
                  <wref id="example.p.1.s.1.w.9" t="bottle" />
                  <wref id="example.p.1.s.1.w.10" t="openers" />
                  <pos class="NNS" />
                  <lemma class="bottle opener" />
              </entity>
          </entities>
      </s>
    </p>
  </text>
</FoLiA>
```

4.5 Structure Annotation

This category encompasses annotation types that define the structure of a document, e.g. paragraphs, sentences, words, sections like chapters, lists, tables, etc… These types are not strictly considered linguistic annotation and equivalents are also commonly found in other document formats such as HTML, TEI, MarkDown, LaTeX, and others. For FoLiA it provides the necessary structural basis that linguistic annotation can build on.

FoLiA defines the following types of structure annotation:

- *Structure Annotation* – This category encompasses annotation types that define the structure of a document, e.g. paragraphs, sentences, words, sections like chapters, lists, tables, etc… These types are not strictly considered linguistic annotation and equivalents are also commonly found in other document formats such as HTML, TEI, MarkDown, LaTeX, and others. For FoLiA it provides the necessary structural basis that linguistic annotation can build on.
  - *Token Annotation* – `<w>` – This annotation type introduces a tokenisation layer for the document. The terms **token** and **word** are used interchangeably in FoLiA as FoLiA itself does not commit to a specific tokenisation paradigm. Tokenisation is a prerequisite for the majority of linguistic annotation types offered by FoLiA and it is one of the most fundamental types of Structure Annotation. The words/tokens are typically embedded in other types of structure elements, such as sentences or paragraphs.
  - *Division Annotation* – `<div>` – Structure annotation representing some kind of division, typically used for chapters, sections, subsections (up to the set definition). Divisions may be nested at will, and may include almost all kinds of other structure elements.
  - *Paragraph Annotation* – `<p>` – Represents a paragraph and holds further structure annotation such as sentences.
  - *Head Annotation* – `<head>` – The `head` element is used to provide a header or title for the structure element in which it is embedded, usually a division (`<div>`)

- *List Annotation* – `<list>` – Structure annotation for enumeration/itemisation, e.g. bulleted lists.
- *Figure Annotation* – `<figure>` – Structure annotation for including pictures, optionally captioned, in documents.
- *Vertical Whitespace* – `<whitespace>` – Structure annotation introducing vertical whitespace
- *Linebreak* – `<br>` – Structure annotation representing a single linebreak and with special facilities to denote pagebreaks.
- *Sentence Annotation* – `<s>` – Structure annotation representing a sentence. Sentence detection is a common stage in NLP alongside tokenisation.
- *Event Annotation* – `<event>` – Structural annotation type representing events, often used in new media contexts for things such as tweets, chat messages and forum posts (as defined by a user-defined set definition). Note that a more linguistic kind of event annotation can be accomplished with *Entity Annotation* or even *Time Segmentation* rather than this one.
- *Quote Annotation* – `<quote>` – Structural annotation used to explicitly mark quoted speech, i.e. that what is reported to be said and appears in the text in some form of quotation marks.
- *Note Annotation* – `<note>` – Structural annotation used for notes, such as footnotes or warnings or notice blocks.
- *Reference Annotation* – `<ref>` – Structural annotation for referring to other annotation types. Used e.g. for referring to bibliography entries (citations) and footnotes.
- *Table Annotation* – `<table>` – Structural annotation type for creating a simple tabular environment, i.e. a table with rows, columns and cells and an optional header.
- *Part Annotation* – `<part>` – The structure element `part` is a fairly abstract structure element that should only be used when a more specific structure element is not available. Most notably, the part element should never be used for representation of morphemes or phonemes! Part can be used to divide a larger structure element, such as a division, or a paragraph into arbitrary subparts.
- *Utterance Annotation* – `<utt>` – An utterance is a structure element that may consist of words or sentences, which in turn may contain words. The opposite is also true, a sentence may consist of multiple utterances. Utterances are often used in the absence of sentences in a speech context, where neat grammatical sentences can not always be distinguished.
- *Entry Annotation* – `<entry>` – FoLiA has a set of structure elements that can be used to represent collections such as glossaries, dictionaries, thesauri, and wordnets. *Entry annotation* defines the entries in such collections, *Term annotation* defines the terms, and *Definition Annotation* provides the definitions.
- *Term Annotation* – `<term>` – FoLiA has a set of structure elements that can be used to represent collections such as glossaries, dictionaries, thesauri, and wordnets. *Entry annotation* defines the entries in such collections, *Term annotation* defines the terms, and *Definition Annotation* provides the definitions.
- *Definition Annotation* – `<def>` – FoLiA has a set of structure elements that can be used to represent collections such as glossaries, dictionaries, thesauri, and wordnets. *Entry annotation* defines the entries in such collections, *Term annotation* defines the terms, and *Definition Annotation* provides the definitions.
- *Example Annotation* – `<ex>` – FoLiA has a set of structure elements that can be used to represent collections such as glossaries, dictionaries, thesauri, and wordnets. *Examples annotation* defines examples in such collections.
- *Hidden Token Annotation* – `<hiddenw>` – This annotation type allows for a hidden token layer in the document. Hidden tokens are ignored for most intents and purposes but may serve a purpose when annotations on implicit tokens is required, for example as targets for syntactic movement annotation.

4.5.1 Token Annotation

This annotation type introduces a tokenisation layer for the document. The terms **token** and **word** are used interchangeably in FoLiA as FoLiA itself does not commit to a specific tokenisation paradigm. Tokenisation is a prerequisite for the majority of linguistic annotation types offered by FoLiA and it is one of the most fundamental types of Structure Annotation. The words/tokens are typically embedded in other types of structure elements, such as sentences or paragraphs.

Specification

Annotation Category *Structure Annotation*

Declaration `<token-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since the beginning

Element `<w>`

API Class `Word` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `textclass` – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to `current`. See *Text class attribute (advanced)*.

- space – This attribute indicates whether spacing should be inserted after this element (it's default value is always yes, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to no. This allows for reconstruction of the detokenised original text.
- src – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- begintime – A timestamp in HH:MM:SS.MMM format, indicating the begin time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- endtime – A timestamp in HH:MM:SS.MMM format, indicating the end time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- speaker – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data <alt> (*Alternative Annotation*), <altlayers> (*Alternative Annotation*), <comment> (*Comment Annotation*), <correction> (*Correction Annotation*), <desc> (*Description Annotation*), <external> (*External Annotation*), <metric> (*Metric Annotation*), <part> (*Part Annotation*), <ph> (*Phonetic Annotation/Content*), <ref> (*Reference Annotation*), <relation> (*Relation Annotation*), <str> (*String Annotation*), <t> (*Text Annotation*)

Valid Context <def> (*Definition Annotation*), <div> (*Division Annotation*), <event> (*Event Annotation*), <ex> (*Example Annotation*), <head> (*Head Annotation*), <note> (*Note Annotation*), <p> (*Paragraph Annotation*), <quote> (*Quote Annotation*), <ref> (*Reference Annotation*), <s> (*Sentence Annotation*), <term> (*Term Annotation*), <utt> (*Utterance Annotation*)

Explanation

Tokenisation is a prerequisite for most forms of linguistic annotation. The <w> element is FoLiA's basic token or word (hence the element's name). This element occurs in the scope of wider structural elements such as <s> (*Sentence Annotation*)

Note: This element carries an extra and optional space attribute with value yes (default), or no, indicating whether a space follows between this token and the next one. This attribute is used to reconstruct the untokenised text.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
```

(continues on next page)

(continued from previous page)

```
                              <annotator processor="p1" />
                       </token-annotation>
           <text-annotation>
                              <annotator processor="p1" />
           </text-annotation>
           <sentence-annotation>
                              <annotator processor="p1" />
           </sentence-annotation>
           <paragraph-annotation>
                              <annotator processor="p1" />
           </paragraph-annotation>
       </annotations>
       <provenance>
          <processor xml:id="p1" name="proycon" type="manual" />
       </provenance>
   </metadata>
   <text xml:id="example.text">
     <p xml:id="example.p.1">
       <s xml:id="example.p.1.s.1">
          <w xml:id="example.p.1.s.1.w.1" class="WORD">
             <t>Hello</t>
          </w>
          <w xml:id="example.p.1.s.1.w.2" class="WORD" space="no">
             <t>World</t>
          </w>
          <w xml:id="example.p.1.s.1.w.3" class="PUNCTUATION">
             <t>!</t>
          </w>
       </s>
       <s xml:id="example.p.1.s.2">
          <w xml:id="example.p.1.s.2.w.1" class="WORD">
             <t>This</t>
          </w>
          <w xml:id="example.p.1.s.2.w.2" class="WORD">
             <t>is</t>
          </w>
          <w xml:id="example.p.1.s.2.w.3" class="WORD">
             <t>an</t>
          </w>
          <w xml:id="example.p.1.s.2.w.4" class="WORD" space="no">
             <t>example</t>
          </w>
          <w xml:id="example.p.1.s.2.w.5" class="PUNCTUATION">
             <t>.</t>
          </w>
       </s>
     </p>
   </text>
</FoLiA>
```

4.5.2 Division Annotation

Structure annotation representing some kind of division, typically used for chapters, sections, subsections (up to the set definition). Divisions may be nested at will, and may include almost all kinds of other structure elements.

Specification

Annotation Category *Structure Annotation*

Declaration `<division-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since the beginning

Element `<div>`

API Class `Division` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always `yes`, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to `no`. This allows for reconstruction of the detokenised original text.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.

- speaker – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data <alt> (*Alternative Annotation*), <altlayers> (*Alternative Annotation*), <comment> (*Comment Annotation*), <correction> (*Correction Annotation*), <desc> (*Description Annotation*), <div> (*Division Annotation*), <entry> (*Entry Annotation*), <event> (*Event Annotation*), <ex> (*Example Annotation*), <external> (*External Annotation*), <figure> (*Figure Annotation*), <gap> (*Gap Annotation*), <head> (*Head Annotation*), <br> (*Linebreak*), <list> (*List Annotation*), <metric> (*Metric Annotation*), <note> (*Note Annotation*), <p> (*Paragraph Annotation*), <part> (*Part Annotation*), <ph> (*Phonetic Annotation/Content*), <quote> (*Quote Annotation*), <ref> (*Reference Annotation*), <relation> (*Relation Annotation*), <s> (*Sentence Annotation*), <table> (*Table Annotation*), <t> (*Text Annotation*), <utt> (*Utterance Annotation*), <whitespace> (*Vertical Whitespace*), <w> (*Token Annotation*)

Valid Context <div> (*Division Annotation*), <event> (*Event Annotation*), <quote> (*Quote Annotation*)

Set Definitions You can use any of the following existing set definitions or simply create your own: * **‘https://raw.githubusercontent.com/proycon/folia/master/setdefinitions/divisions.foliaset.xml‘_**

Description & Examples

The structure element <div> is used to create divisions and subdivisions within a text.

Each division *may* be of a particular *class* pertaining to a *set* defining all possible classes, common classes for this annotation type would be *chapter*, *section*, *subsection*. A set, however, is not mandatory for most types of structure, so divisions may be set-less.

Divisions and other structural units are often numbered, think for example of chapters and sections. The number, as it was in the source document, can be encoded in the *n* attribute of the structure annotation element.

Divisions should never be used for marking paragraphs, sentences or other smaller structural entities for which FoLiA provides explicit structural element. Divisions only cover large structures.

The following example shows a FoLiA document with structured divisions with headers and paragraphs. It does not provide any further tokenisation.

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                          <annotator processor="p1" />
          </text-annotation>
          <division-annotation set="https://raw.githubusercontent.com/
↪LanguageMachines/uctodata/master/setdefinitions/divisions.foliaset.xml">
                          <annotator processor="p1" />
                  </division-annotation>
```

(continues on next page)

(continued from previous page)

```
          <head-annotation>
                         <annotator processor="p1" />
                 </head-annotation>
          <paragraph-annotation>
                         <annotator processor="p1" />
                 </paragraph-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
     <div xml:id="example.div.1" class="chapter" n="1">
         <head>
             <t>Chapter 1: In the beginning</t>
         </head>
         <div xml:id="example.div.1.1" class="section" n="1.1">
             <head>
                 <t>Section 1.1: The first steps</t>
             </head>
             <p xml:id="example.div.1.1.p.1">
                 <t>And so the first paragraph commences...</t>
             </p>
         </div>
     </div>
  </text>
</FoLiA>
```

4.5.3 Paragraph Annotation

Represents a paragraph and holds further structure annotation such as sentences.

Specification

Annotation Category *Structure Annotation*

Declaration `<paragraph-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since the beginning

Element `<p>`

API Class `Paragraph` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.

- class – The class of the annotation, i.e. the annotation tag in the vocabulary defined by set.
- processor – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- annotator – This is an older alternative to the processor attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- annotatortype – This is an older alternative to the processor attribute, without support for full provenance. It is used together with annotator and specific the type of the annotator, either manual for human annotators or auto for automated systems.
- confidence – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- datetime – The date and time when this annotation was recorded, the format is YYYY-MM-DDThh:mm:ss (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- n – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- space – This attribute indicates whether spacing should be inserted after this element (it's default value is always yes, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to no. This allows for reconstruction of the detokenised original text.
- src – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- begintime – A timestamp in HH:MM:SS.MMM format, indicating the begin time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- endtime – A timestamp in HH:MM:SS.MMM format, indicating the end time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- speaker – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data <alt> (*Alternative Annotation*), <altlayers> (*Alternative Annotation*), <comment> (*Comment Annotation*), <correction> (*Correction Annotation*), <desc> (*Description Annotation*), <entry> (*Entry Annotation*), <event> (*Event Annotation*), <ex> (*Example Annotation*), <external> (*External Annotation*), <figure> (*Figure Annotation*), <gap> (*Gap Annotation*), <head> (*Head Annotation*), <hiddenw> (*Hidden Token Annotation*), <br> (*Linebreak*), <list> (*List Annotation*), <metric> (*Metric Annotation*), <note> (*Note Annotation*), <part> (*Part Annotation*), <ph> (*Phonetic Annotation/Content*), <quote> (*Quote Annotation*), <ref> (*Reference Annotation*), <relation> (*Relation Annotation*), <s> (*Sentence Annotation*), <str> (*String Annotation*), <t> (*Text Annotation*), <whitespace> (*Vertical Whitespace*), <w> (*Token Annotation*)

Valid Context `<def>` (*Definition Annotation*), `<div>` (*Division Annotation*), `<event>` (*Event Annotation*), `<ex>` (*Example Annotation*), `<head>` (*Head Annotation*), `<note>` (*Note Annotation*), `<quote>` (*Quote Annotation*), `<ref>` (*Reference Annotation*), `<term>` (*Term Annotation*)

Explanation & Examples

Paragraphs are a very common structural element and their use is encouraged. The following example shows a single paragraph within a structure of divisions with headers, it does not provide any deeper tokenisation.

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <division-annotation set="https://raw.githubusercontent.com/
↪LanguageMachines/uctodata/master/setdefinitions/divisions.foliaset.xml">
                         <annotator processor="p1" />
                 </division-annotation>
          <head-annotation>
                         <annotator processor="p1" />
                 </head-annotation>
          <paragraph-annotation>
                         <annotator processor="p1" />
                 </paragraph-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
     <div xml:id="example.div.1" class="chapter" n="1">
         <head>
             <t>Chapter 1: In the beginning</t>
         </head>
         <div xml:id="example.div.1.1" class="section" n="1.1">
             <head>
                 <t>Section 1.1: The first steps</t>
             </head>
             <p xml:id="example.div.1.1.p.1">
                 <t>And so the first paragraph commences...</t>
             </p>
         </div>
     </div>
  </text>
</FoLiA>
```

The next example shows a paragraph with sentences and tokenisation:

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
```

(continues on next page)

(continued from previous page)

```xml
                         <annotator processor="p1" />
                  </token-annotation>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
          </sentence-annotation>
          <paragraph-annotation>
                         <annotator processor="p1" />
          </paragraph-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <p xml:id="example.p.1">
      <s xml:id="example.p.1.s.1">
         <w xml:id="example.p.1.s.1.w.1" class="WORD">
            <t>Hello</t>
         </w>
         <w xml:id="example.p.1.s.1.w.2" class="WORD" space="no">
            <t>World</t>
         </w>
         <w xml:id="example.p.1.s.1.w.3" class="PUNCTUATION">
            <t>!</t>
         </w>
      </s>
      <s xml:id="example.p.1.s.2">
         <w xml:id="example.p.1.s.2.w.1" class="WORD">
            <t>This</t>
         </w>
         <w xml:id="example.p.1.s.2.w.2" class="WORD">
            <t>is</t>
         </w>
         <w xml:id="example.p.1.s.2.w.3" class="WORD">
            <t>an</t>
         </w>
         <w xml:id="example.p.1.s.2.w.4" class="WORD" space="no">
            <t>example</t>
         </w>
         <w xml:id="example.p.1.s.2.w.5" class="PUNCTUATION">
            <t>.</t>
         </w>
      </s>
    </p>
  </text>
</FoLiA>
```

4.5.4 Head Annotation

The `head` element is used to provide a header or title for the structure element in which it is embedded, usually a division (`<div>`)

Specification

Annotation Category *Structure Annotation*

Declaration `<head-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since the beginning

Element `<head>`

API Class `Head` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always `yes`, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to `no`. This allows for reconstruction of the detokenised original text.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.

- speaker – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data `<alt>` (*Alternative Annotation*), `<altlayers>` (*Alternative Annotation*), `<comment>` (*Comment Annotation*), `<correction>` (*Correction Annotation*), `<desc>` (*Description Annotation*), `<event>` (*Event Annotation*), `<external>` (*External Annotation*), `<gap>` (*Gap Annotation*), `<hiddenw>` (*Hidden Token Annotation*), `<br>` (*Linebreak*), `<metric>` (*Metric Annotation*), `<p>` (*Paragraph Annotation*), `<part>` (*Part Annotation*), `<ph>` (*Phonetic Annotation/Content*), `<ref>` (*Reference Annotation*), `<relation>` (*Relation Annotation*), `<s>` (*Sentence Annotation*), `<str>` (*String Annotation*), `<t>` (*Text Annotation*), `<whitespace>` (*Vertical Whitespace*), `<w>` (*Token Annotation*)

Valid Context `<div>` (*Division Annotation*), `<event>` (*Event Annotation*), `<note>` (*Note Annotation*), `<p>` (*Paragraph Annotation*)

Description & Examples

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <division-annotation set="https://raw.githubusercontent.com/
↪LanguageMachines/uctodata/master/setdefinitions/divisions.foliaset.xml">
                         <annotator processor="p1" />
                  </division-annotation>
          <head-annotation>
                         <annotator processor="p1" />
                  </head-annotation>
          <paragraph-annotation>
                         <annotator processor="p1" />
                  </paragraph-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
     <div xml:id="example.div.1" class="chapter" n="1">
        <head>
            <t>Chapter 1: In the beginning</t>
        </head>
        <div xml:id="example.div.1.1" class="section" n="1.1">
            <head>
                <t>Section 1.1: The first steps</t>
            </head>
```

(continues on next page)

(continued from previous page)

```
            <p xml:id="example.div.1.1.p.1">
                <t>And so the first paragraph commences...</t>
            </p>
        </div>
    </div>
  </text>
</FoLiA>
```

4.5.5 List Annotation

Structure annotation for enumeration/itemisation, e.g. bulleted lists.

Specification

List

Annotation Category *Structure Annotation*

Declaration `<list-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since the beginning

Element `<list>`

API Class `List` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.

- n – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- space – This attribute indicates whether spacing should be inserted after this element (it's default value is always yes, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to no. This allows for reconstruction of the detokenised original text.
- src – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- begintime – A timestamp in HH:MM:SS.MMM format, indicating the begin time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- endtime – A timestamp in HH:MM:SS.MMM format, indicating the end time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- speaker – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data <alt> (*Alternative Annotation*), <altlayers> (*Alternative Annotation*), <comment> (*Comment Annotation*), <correction> (*Correction Annotation*), <desc> (*Description Annotation*), <event> (*Event Annotation*), <external> (*External Annotation*), <br> (*Linebreak*), <metric> (*Metric Annotation*), <note> (*Note Annotation*), <part> (*Part Annotation*), <ph> (*Phonetic Annotation/Content*), <ref> (*Reference Annotation*), <relation> (*Relation Annotation*), <str> (*String Annotation*), <t> (*Text Annotation*)

Valid Context <def> (*Definition Annotation*), <div> (*Division Annotation*), <event> (*Event Annotation*), <ex> (*Example Annotation*), <note> (*Note Annotation*), <p> (*Paragraph Annotation*), <term> (*Term Annotation*)

Item

Element <item>

API Class ListItem (FoLiApy API Reference)

Required Attributes

Optional Attributes

- xml:id – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- set – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The set must be referred to also in the *Annotation Declarations* for this annotation type.

- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always `yes`, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to `no`. This allows for reconstruction of the detokenised original text.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<alt>` (*Alternative Annotation*), `<altlayers>` (*Alternative Annotation*), `<comment>` (*Comment Annotation*), `<correction>` (*Correction Annotation*), `<desc>` (*Description Annotation*), `<event>` (*Event Annotation*), `<external>` (*External Annotation*), `<gap>` (*Gap Annotation*), `<hiddenw>` (*Hidden Token Annotation*), `<br>` (*Linebreak*), `<list>` (*List Annotation*), `<metric>` (*Metric Annotation*), `<note>` (*Note Annotation*), `<p>` (*Paragraph Annotation*), `<part>` (*Part Annotation*), `<ph>` (*Phonetic Annotation/Content*), `<quote>` (*Quote Annotation*), `<ref>` (*Reference Annotation*), `<relation>` (*Relation Annotation*), `<s>` (*Sentence Annotation*), `<str>` (*String Annotation*), `<t>` (*Text Annotation*), `<whitespace>` (*Vertical Whitespace*), `<w>` (*Token Annotation*)

Valid Context `<list>` (*List Annotation*)

Explanation

The `<list>` element creates a list, a class and set may be associated with the list to indicate the type of list. As always, this vocabulary is user-defined. Individual list items need to be wrapped in the `<item>` structure element.

Example

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation />
          <token-annotation>
                         <annotator processor="p1" />
                  </token-annotation>
          <list-annotation />
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
      <list xml:id="example.list.1">
          <item n="1">
             <w xml:id="example.list.1.item.1">
                <t>Hello</t>
             </w>
          </item>
          <item n="2">
             <w xml:id="example.list.1.item.2">
                <t>Bonjour</t>
             </w>
          </item>
          <item n="3">
             <w xml:id="example.list.1.item.3">
                <t>Hola</t>
             </w>
          </item>
      </list>
  </text>
</FoLiA>
```

4.5.6 Figure Annotation

Structure annotation for including pictures, optionally captioned, in documents.

Specification

Figure

Annotation Category *Structure Annotation*

Declaration `<figure-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since the beginning

Element `<figure>`

API Class `Figure` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always `yes`, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to `no`. This allows for reconstruction of the detokenised original text.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry

no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data `<alt>` (*Alternative Annotation*), `<altlayers>` (*Alternative Annotation*), `<comment>` (*Comment Annotation*), `<correction>` (*Correction Annotation*), `<desc>` (*Description Annotation*), `<external>` (*External Annotation*), `<br>` (*Linebreak*), `<metric>` (*Metric Annotation*), `<part>` (*Part Annotation*), `<relation>` (*Relation Annotation*), `<str>` (*String Annotation*), `<t>` (*Text Annotation*)

Valid Context `<def>` (*Definition Annotation*), `<div>` (*Division Annotation*), `<event>` (*Event Annotation*), `<ex>` (*Example Annotation*), `<note>` (*Note Annotation*), `<p>` (*Paragraph Annotation*), `<term>` (*Term Annotation*)

Extra Attributes `src` – Points to an image file (URL)

Caption

Element `<caption>`

API Class `Caption` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always yes, so it does not need to be specified in that case), but if

tokens or other structural elements are glued together then the value should be set to no. This allows for reconstruction of the detokenised original text.

- src – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- begintime – A timestamp in HH:MM:SS.MMM format, indicating the begin time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- endtime – A timestamp in HH:MM:SS.MMM format, indicating the end time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- speaker – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data <alt> (*Alternative Annotation*), <altlayers> (*Alternative Annotation*), <comment> (*Comment Annotation*), <correction> (*Correction Annotation*), <desc> (*Description Annotation*), <external> (*External Annotation*), <gap> (*Gap Annotation*), <br> (*Linebreak*), <metric> (*Metric Annotation*), <p> (*Paragraph Annotation*), <part> (*Part Annotation*), <ph> (*Phonetic Annotation/Content*), <quote> (*Quote Annotation*), <ref> (*Reference Annotation*), <relation> (*Relation Annotation*), <s> (*Sentence Annotation*), <str> (*String Annotation*), <t> (*Text Annotation*), <whitespace> (*Vertical Whitespace*)

Valid Context <figure> (*Figure Annotation*), <list> (*List Annotation*)

Explanation

Even figures can be encoded in the FoLiA format, although the actual figure itself can only be included as a mere reference to an external image file, but including such a reference with the src attribute is optional.

Within the context of a figure, a caption element can be used.

Example

The following example shows a figure and caption:

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <division-annotation set="https://raw.githubusercontent.com/
↪LanguageMachines/uctodata/master/setdefinitions/divisions.foliaset.xml">
                         <annotator processor="p1" />
                  </division-annotation>
          <head-annotation>
                         <annotator processor="p1" />
                  </head-annotation>
```

(continues on next page)

(continued from previous page)

```
        <figure-annotation>
                    <annotator processor="p1" />
                </figure-annotation>
    </annotations>
    <provenance>
        <processor xml:id="p1" name="proycon" type="manual" />
    </provenance>
  </metadata>
  <text xml:id="example.text">
    <div xml:id="example.div.1" class="chapter" n="1">
        <head>
            <t>Rosetta Stone</t>
        </head>
        <figure xml:id="example.figure.1" n="1" src="https://upload.wikimedia.org/
↪wikipedia/commons/thumb/2/23/Rosetta_Stone.JPG/1280px-Rosetta_Stone.JPG">
          <caption><t>Wikipedia: The Rosetta Stone is a granodiorite stele, found in␣
↪1799, inscribed with three versions of a decree issued at Memphis, Egypt in 196 BC␣
↪during the Ptolemaic dynasty on behalf of King Ptolemy V </t></caption>
        </figure>
    </div>
  </text>
</FoLiA>
```

4.5.7 Vertical Whitespace

Structure annotation introducing vertical whitespace

Note: Do not confuse this with the `<t-space>` markup element that is used for *horizontal* whitespace, see *Horizontal Whitespace*.

Specification

Annotation Category *Structure Annotation*

Declaration `<whitespace-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since the beginning

Element `<whitespace>`

API Class `Whitespace` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.

- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always `yes`, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to `no`. This allows for reconstruction of the detokenised original text.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<alt>` (*Alternative Annotation*), `<altlayers>` (*Alternative Annotation*), `<comment>` (*Comment Annotation*), `<correction>` (*Correction Annotation*), `<desc>` (*Description Annotation*), `<external>` (*External Annotation*), `<metric>` (*Metric Annotation*), `<part>` (*Part Annotation*), `<relation>` (*Relation Annotation*)

Valid Context `<def>` (*Definition Annotation*), `<div>` (*Division Annotation*), `<event>` (*Event Annotation*), `<ex>` (*Example Annotation*), `<head>` (*Head Annotation*), `<note>` (*Note Annotation*), `<p>` (*Paragraph Annotation*), `<quote>` (*Quote Annotation*), `<ref>` (*Reference Annotation*), `<s>` (*Sentence Annotation*), `<term>` (*Term Annotation*)

Text markup Element

Element `<t-whitespace>`

API Class `TextMarkupWhitespace` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they

use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

- `xlink:href` – Turns this element into a hyperlink to the specified URL
- `xlink:type` – The type of link (you'll want to use `simple` in almost all cases).

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<br>` (*Linebreak*)

Valid Context

Description & Examples

Sometimes you may want to explicitly specify vertical whitespace, rather than repeat multiple linebreaks (*Linebreak*), the *<whitespace>* element accomplishes this. Note that using *<p>* to denote paragraphs is always strongly preferred over using *<whitespace>* to mark their boundaries, this element should be used sparingly!

The difference between `br` and `whitespace` is that the former specifies that only a linebreak was present, not forcing any vertical whitespace between the lines, whilst the latter actually generates an empty space, which would comparable to two successive `br` statements. Moreover, you have the ability to associate your own vocabulary set with `<whitespace>` and assign your own size-interpretation to it. Both elements can be used inside various structural elements, such as divisions, paragraphs, headers, and sentences.

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.5.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <division-annotation set="https://raw.githubusercontent.com/
↪LanguageMachines/uctodata/master/setdefinitions/divisions.foliaset.xml">
                         <annotator processor="p1" />
                  </division-annotation>
          <whitespace-annotation>
                         <annotator processor="p1" />
                  </whitespace-annotation>
          <hspace-annotation>
                         <annotator processor="p1" />
                  </hspace-annotation>
          <linebreak-annotation>
                         <annotator processor="p1" />
          </linebreak-annotation>
          <hyphenation-annotation>
                         <annotator processor="p1" />
         </hyphenation-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <div xml:id="example.div.1" class="section" n="1">
        <t>Blah...</t>
    </div>
    <whitespace />
    <br newpage="yes" pagenr="2" />
```

(continues on next page)

(continued from previous page)

```
      <div xml:id="example.div.2" class="section" n="2">
          <!-- BR has a double role, it can be used a text markup element as well, as␣
↪seen on the next line -->
          <t>To be, <br />or not to be!</t>
      </div>
      <div xml:id="example.div.3" class="section" n="3">
          <t>Don't leave me bro<t-hbr/>ken and alone!</t>
      </div>
      <div xml:id="example.div.4" class="section" n="4">
          <t>Space,<t-hspace/>the<t-hspace/>final<t-hspace/>
            frontier</t>
      </div>
  </text>
</FoLiA>
```

4.5.8 Linebreak

Structure annotation representing a single linebreak and with special facilities to denote pagebreaks.

Specification

Annotation Category *Structure Annotation*

Declaration `<linebreak-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since the beginning

Element `<br>`

API Class `Linebreak` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.

- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always `yes`, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to `no`. This allows for reconstruction of the detokenised original text.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.
- `xlink:href` – Turns this element into a hyperlink to the specified URL
- `xlink:type` – The type of link (you'll want to use `simple` in almost all cases).

Accepted Data `<alt>` (*Alternative Annotation*), `<altlayers>` (*Alternative Annotation*), `<comment>` (*Comment Annotation*), `<correction>` (*Correction Annotation*), `<desc>` (*Description Annotation*), `<external>` (*External Annotation*), `<metric>` (*Metric Annotation*), `<part>` (*Part Annotation*), `<relation>` (*Relation Annotation*)

Valid Context `<def>` (*Definition Annotation*), `<div>` (*Division Annotation*), `<event>` (*Event Annotation*), `<ex>` (*Example Annotation*), `<figure>` (*Figure Annotation*), `<head>` (*Head Annotation*), `<t-hbr>` (*Hyphenation*), `<list>` (*List Annotation*), `<note>` (*Note Annotation*), `<p>` (*Paragraph Annotation*), `<quote>` (*Quote Annotation*), `<ref>` (*Reference Annotation*), `<s>` (*Sentence Annotation*), `<table>` (*Table Annotation*), `<term>` (*Term Annotation*), `<t>` (*Text Annotation*), `<t-correction>` (*Correction Annotation*), `<t-error>` (*Error Detection Annotation (DEPRECATED)*), `<t-gap>` (*Gap Annotation*), `<t-hspace>` (*Horizontal Whitespace*), `<t-lang>` (*Language Annotation*), `<t-ref>` (*Reference Annotation*), `<t-str>` (*String Annotation*), `<t-style>` (*Style Annotation*), `<t-whitespace>` (*Vertical Whitespace*)

Extra Attributes

- `newpage` – Can be set to `yes` to indicate that the break is not just a linebreak, but also a pagebreak (defaults to `no`)
- `pagenr` – The number of the page after the break

- `linenr` – The number of the line after the break

Description & Examples

Linebreaks play a double role, they are a structure element as well as a text markup element, the latter implies that you may also use `<br/>` *within* the scope of text content, so within a `<t>` element.

The difference between `br` and `whitespace` is that the former specifies that only a linebreak was present, not forcing any vertical whitespace between the lines, whilst the latter actually generates an empty space, which would comparable to two successive `br` statements. Both elements can be used inside various structural elements, such as divisions, paragraphs, headers, and sentences. Note that the example below also contains an example of *Hyphenation*, which is a special softer kind of linebreak.

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.5.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <division-annotation set="https://raw.githubusercontent.com/
↪LanguageMachines/uctodata/master/setdefinitions/divisions.foliaset.xml">
                         <annotator processor="p1" />
                  </division-annotation>
          <whitespace-annotation>
                         <annotator processor="p1" />
                  </whitespace-annotation>
          <hspace-annotation>
                         <annotator processor="p1" />
                  </hspace-annotation>
          <linebreak-annotation>
                         <annotator processor="p1" />
          </linebreak-annotation>
          <hyphenation-annotation>
                         <annotator processor="p1" />
         </hyphenation-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
     <div xml:id="example.div.1" class="section" n="1">
         <t>Blah...</t>
     </div>
     <whitespace />
     <br newpage="yes" pagenr="2" />
     <div xml:id="example.div.2" class="section" n="2">
         <!-- BR has a double role, it can be used a text markup element as well, as␣
↪seen on the next line -->
         <t>To be, <br />or not to be!</t>
     </div>
     <div xml:id="example.div.3" class="section" n="3">
         <t>Don't leave me bro<t-hbr/>ken and alone!</t>
     </div>
     <div xml:id="example.div.4" class="section" n="4">
         <t>Space,<t-hspace/>the<t-hspace/>final<t-hspace/>
```

(continues on next page)

(continued from previous page)

```
            frontier</t>
        </div>
    </text>
</FoLiA>
```

You can use `<br/>` also in the context of *Text Markup Annotation*, as in the following example:

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
                  </sentence-annotation>
          <linebreak-annotation>
                         <annotator processor="p1" />
                  </linebreak-annotation>
          <part-annotation>
                         <annotator processor="p1" />
                  </part-annotation>
          <style-annotation set="https://raw.githubusercontent.com/proycon/folia/
↪master/setdefinitions/styles.foliaset.xml">
                         <annotator processor="p1" />
                  </style-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
        <s>
            <t>To <t-style class="italic">be</t-style> or not to be,<br/>that is the
↪<t-style class="bold"><t-style class="red">question</t-style></t-style>.</t>
        </s>
  </text>
</FoLiA>
```

4.5.9 Sentence Annotation

Structure annotation representing a sentence. Sentence detection is a common stage in NLP alongside tokenisation.

Specification

Annotation Category *Structure Annotation*

Declaration `<sentence-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since the beginning

Element `<s>`

API Class `Sentence` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- xml:id – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- set – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The set must be referred to also in the *Annotation Declarations* for this annotation type.
- class – The class of the annotation, i.e. the annotation tag in the vocabulary defined by set.
- processor – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- annotator – This is an older alternative to the processor attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- annotatortype – This is an older alternative to the processor attribute, without support for full provenance. It is used together with annotator and specific the type of the annotator, either manual for human annotators or auto for automated systems.
- confidence – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- datetime – The date and time when this annotation was recorded, the format is YYYY-MM-DDThh:mm:ss (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- n – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- space – This attribute indicates whether spacing should be inserted after this element (it's default value is always yes, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to no. This allows for reconstruction of the detokenised original text.
- src – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- begintime – A timestamp in HH:MM:SS.MMM format, indicating the begin time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- endtime – A timestamp in HH:MM:SS.MMM format, indicating the end time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- speaker – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data <alt> (*Alternative Annotation*), <altlayers> (*Alternative Annotation*), <comment> (*Comment Annotation*), <correction> (*Correction Annotation*), <desc> (*Description Annotation*), <entry> (*Entry Annotation*), <event> (*Event Annotation*), <ex> (*Example Annotation*), <external> (*External Annotation*), <gap> (*Gap Annotation*), <hiddenw> (*Hidden Token Annotation*), <br> (*Linebreak*), <metric> (*Metric Annotation*), <note> (*Note Annotation*), <part> (*Part Annotation*), <ph> (*Phonetic Annotation/Content*), <quote> (*Quote Annotation*), <ref> (*Reference Annotation*), <relation> (*Relation Annotation*), <str> (*String Annotation*), <t> (*Text Annotation*), <whitespace> (*Vertical Whitespace*), <w> (*Token Annotation*)

Valid Context <def> (*Definition Annotation*), <div> (*Division Annotation*), <event> (*Event Annotation*), <ex> (*Example Annotation*), <head> (*Head Annotation*), <note> (*Note Annotation*), <p> (*Paragraph Annotation*), <quote> (*Quote Annotation*), <ref> (*Reference Annotation*), <term> (*Term Annotation*), <utt> (*Utterance Annotation*)

Explanation & Examples

The next example shows a paragraph with sentences and tokenisation:

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
                         <annotator processor="p1" />
                  </token-annotation>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
          </sentence-annotation>
          <paragraph-annotation>
                         <annotator processor="p1" />
          </paragraph-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <p xml:id="example.p.1">
      <s xml:id="example.p.1.s.1">
         <w xml:id="example.p.1.s.1.w.1" class="WORD">
            <t>Hello</t>
         </w>
         <w xml:id="example.p.1.s.1.w.2" class="WORD" space="no">
            <t>World</t>
         </w>
         <w xml:id="example.p.1.s.1.w.3" class="PUNCTUATION">
            <t>!</t>
         </w>
      </s>
      <s xml:id="example.p.1.s.2">
         <w xml:id="example.p.1.s.2.w.1" class="WORD">
            <t>This</t>
```

(continues on next page)

(continued from previous page)

```
          </w>
          <w xml:id="example.p.1.s.2.w.2" class="WORD">
            <t>is</t>
          </w>
          <w xml:id="example.p.1.s.2.w.3" class="WORD">
            <t>an</t>
          </w>
          <w xml:id="example.p.1.s.2.w.4" class="WORD" space="no">
            <t>example</t>
          </w>
          <w xml:id="example.p.1.s.2.w.5" class="PUNCTUATION">
            <t>.</t>
          </w>
        </s>
      </p>
    </text>
</FoLiA>
```

4.5.10 Event Annotation

Structural annotation type representing events, often used in new media contexts for things such as tweets, chat messages and forum posts (as defined by a user-defined set definition). Note that a more linguistic kind of event annotation can be accomplished with *Entity Annotation* or even *Time Segmentation* rather than this one.

Specification

Annotation Category *Structure Annotation*

Declaration `<event-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History since v0.7

Element `<event>`

API Class `Event` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.

- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always `yes`, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to `no`. This allows for reconstruction of the detokenised original text.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<alt>` (*Alternative Annotation*), `<altlayers>` (*Alternative Annotation*), `<comment>` (*Comment Annotation*), `<correction>` (*Correction Annotation*), `<desc>` (*Description Annotation*), `<div>` (*Division Annotation*), `<entry>` (*Entry Annotation*), `<event>` (*Event Annotation*), `<ex>` (*Example Annotation*), `<external>` (*External Annotation*), `<figure>` (*Figure Annotation*), `<gap>` (*Gap Annotation*), `<head>` (*Head Annotation*), `<hiddenw>` (*Hidden Token Annotation*), `<br>` (*Linebreak*), `<list>` (*List Annotation*), `<metric>` (*Metric Annotation*), `<note>` (*Note Annotation*), `<p>` (*Paragraph Annotation*), `<part>` (*Part Annotation*), `<ph>` (*Phonetic Annotation/Content*), `<quote>` (*Quote Annotation*), `<ref>` (*Reference Annotation*), `<relation>` (*Relation Annotation*), `<s>` (*Sentence Annotation*), `<str>` (*String Annotation*), `<table>` (*Table Annotation*), `<t>` (*Text Annotation*), `<utt>` (*Utterance Annotation*), `<whitespace>` (*Vertical Whitespace*), `<w>` (*Token Annotation*)

Valid Context `<div>` (*Division Annotation*), `<event>` (*Event Annotation*), `<head>` (*Head Annotation*), `<list>` (*List Annotation*), `<p>` (*Paragraph Annotation*), `<s>` (*Sentence Annotation*), `<term>` (*Term Annotation*)

Feature subsets (extra attributes)

- `actor`
- `begindatetime`
- `enddatetime`

Explanation

Event structure, though uncommon to regular written text, can be useful in certain documents. Divisions, paragraphs, sentences, or even words can be encapsulated in an event element to indicate they somehow form an event entity of a particular class. This kind of structure annotation is especially useful in dealing with computer-mediated communication such as chat logs, tweets, and internet fora, in which chat turns, forum posts, and tweets can be demarcated as particular events.

The event class predefines some feature subsets you can use (you can use these as XML attributes, see *Features* for more information on features); the subsets `begindatetime` and `enddatetime` can be used express to the exact moment at which an event started or ended. Note that this differs from the common `datetime` attribute, which would describe the time at which the annotation was recorded, rather than when the event took place! The `actor` subset is used to associate the person responsible for the event, i.e. the speaker or poster.

For more fine-grained control over timed events, for example within sentences. It is recommended to use *Time Segmentation*!

Example

The following example shows a chat log composed of message events:

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <event-annotation set="adhoc">
                         <annotator processor="p1" />
                  </event-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
                  </sentence-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <event class="message" begindatetime="2011-12-15T19:01"
     enddatetime="2011-12-15T19:05" actor="Jane Doe">
        <s>
            <t>Hello John.</t>
        </s>
        <s>
            <t>How are you doing?</t>
        </s>
    </event>
    <event class="message" begindatetime="2011-12-15T19:06"
     actor="John Doe">
```

(continues on next page)

(continued from previous page)

```
        <s>
            <t>I am fine Jane, thanks.</t>
        </s>
    </event>
  </text>
</FoLiA>
```

4.5.11 Quote Annotation

Structural annotation used to explicitly mark quoted speech, i.e. that what is reported to be said and appears in the text in some form of quotation marks.

Specification

Annotation Category *Structure Annotation*

Declaration `<quote-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since v0.11

Element `<quote>`

API Class `Quote` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be

an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).

- space – This attribute indicates whether spacing should be inserted after this element (it's default value is always yes, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to no. This allows for reconstruction of the detokenised original text.
- src – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- begintime – A timestamp in HH:MM:SS.MMM format, indicating the begin time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- endtime – A timestamp in HH:MM:SS.MMM format, indicating the end time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- speaker – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data <alt> (*Alternative Annotation*), <altlayers> (*Alternative Annotation*), <comment> (*Comment Annotation*), <correction> (*Correction Annotation*), <desc> (*Description Annotation*), <div> (*Division Annotation*), <external> (*External Annotation*), <gap> (*Gap Annotation*), <hiddenw> (*Hidden Token Annotation*), <br> (*Linebreak*), <metric> (*Metric Annotation*), <p> (*Paragraph Annotation*), <part> (*Part Annotation*), <quote> (*Quote Annotation*), <ref> (*Reference Annotation*), <relation> (*Relation Annotation*), <s> (*Sentence Annotation*), <str> (*String Annotation*), <t> (*Text Annotation*), <utt> (*Utterance Annotation*), <whitespace> (*Vertical Whitespace*), <w> (*Token Annotation*)

Valid Context <div> (*Division Annotation*), <event> (*Event Annotation*), <p> (*Paragraph Annotation*), <quote> (*Quote Annotation*), <ref> (*Reference Annotation*), <s> (*Sentence Annotation*), <utt> (*Utterance Annotation*)

Example

The next example shows a quote annotations:

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation>
             <annotator processor="p1" />
          </token-annotation>
          <text-annotation>
             <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
             <annotator processor="p1" />
          </sentence-annotation>
```

(continues on next page)

(continued from previous page)

```
          <quote-annotation>
             <annotator processor="p1" />
          </quote-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
     <s xml:id="example.p.1.s.1">
      <w xml:id="example.p.1.s.1.w.1"><t>He</t></w>
      <w xml:id="example.p.1.s.1.w.2" space="no"><t>said</t></w>
      <w xml:id="example.p.1.s.1.w.3">
        <t>:</t>
      </w>
      <quote xml:id="example.p.1.s.1.quote.1">
        <w xml:id="example.p.1.s.1.w.4" space="no">
          <t>"</t>
        </w>
        <s xml:id="example.p.1.s.1.quote.1.s.1">
           <w xml:id="example.p.1.s.1.w.5"><t>I</t></w>
           <w xml:id="example.p.1.s.1.w.6"><t>do</t></w>
           <w xml:id="example.p.1.s.1.w.7"><t>not</t></w>
           <w xml:id="example.p.1.s.1.w.8" space="no"><t>know</t></w>
           <w xml:id="example.p.1.s.1.w.9">
            <t>.</t>
           </w>
        </s>
        <s xml:id="example.p.1.s.1.quote.1.s.2">
           <w xml:id="example.p.1.s.1.w.10"><t>I</t></w>
           <w xml:id="example.p.1.s.1.w.11"><t>think</t></w>
           <w xml:id="example.p.1.s.1.w.12"><t>you</t></w>
           <w xml:id="example.p.1.s.1.w.13"><t>are</t></w>
           <w xml:id="example.p.1.s.1.w.14" space="no"><t>right</t></w>
        </s>
        <w xml:id="example.p.1.s.1.w.15">
          <t>"</t>
        </w>
      </quote>
      <w xml:id="example.p.1.s.1.w.16"><t>,</t></w>
      <w xml:id="example.p.1.s.1.w.17"><t>and</t></w>
      <w xml:id="example.p.1.s.1.w.18" space="no"><t>left</t></w>
      <w xml:id="example.p.1.s.1.w.19">
        <t>.</t>
      </w>
     </s>
  </text>
</FoLiA>
```

4.5.12 Note Annotation

Structural annotation used for notes, such as footnotes or warnings or notice blocks.

Specification

Annotation Category *Structure Annotation*

Declaration `<note-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since v0.11

Element `<note>`

API Class `Note` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always `yes`, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to `no`. This allows for reconstruction of the detokenised original text.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.

- speaker – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data <alt> (*Alternative Annotation*), <altlayers> (*Alternative Annotation*), <comment> (*Comment Annotation*), <correction> (*Correction Annotation*), <desc> (*Description Annotation*), <ex> (*Example Annotation*), <external> (*External Annotation*), <figure> (*Figure Annotation*), <head> (*Head Annotation*), <hiddenw> (*Hidden Token Annotation*), <br> (*Linebreak*), <list> (*List Annotation*), <metric> (*Metric Annotation*), <p> (*Paragraph Annotation*), <part> (*Part Annotation*), <ph> (*Phonetic Annotation/Content*), <ref> (*Reference Annotation*), <relation> (*Relation Annotation*), <s> (*Sentence Annotation*), <str> (*String Annotation*), <table> (*Table Annotation*), <t> (*Text Annotation*), <utt> (*Utterance Annotation*), <whitespace> (*Vertical Whitespace*), <w> (*Token Annotation*)

Valid Context <div> (*Division Annotation*), <event> (*Event Annotation*), <list> (*List Annotation*), <p> (*Paragraph Annotation*), <s> (*Sentence Annotation*), <utt> (*Utterance Annotation*)

Set Definitions You can use any of the following existing set definitions or simply create your own: * **‘https://raw.githubusercontent.com/proycon/folia/master/setdefinitions/notes.foliaset.xml‘_**

Explanation

The structure element <note> allows for notes to be included in FoLiA documents. A footnote or a bibliographical reference is an example of a note. The notes form an integral part of the text. For notes that are merely descriptive comments on the text or its annotations, rather than a part of it, use <desc> or <comment> instead. Notes themselves can contain all the usual forms of annotations.

The place of a note in the text is where it will appear. References to the note are made using a specific tag, <ref>, discussed in *Reference Annotation*.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" xmlns:xlink="http://www.w3.org/1999/xlink"␣
↪version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <note-annotation set="https://raw.githubusercontent.com/proycon/folia/
↪master/setdefinitions/notes.foliaset.xml">
                         <annotator processor="p1" />
                     </note-annotation>
          <reference-annotation>
                         <annotator processor="p1" />
```

(continues on next page)

(continued from previous page)

```xml
          </reference-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
                 </sentence-annotation>
          <paragraph-annotation>
                         <annotator processor="p1" />
                 </paragraph-annotation>
          <part-annotation>
                         <annotator processor="p1" />
          </part-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
      <p xml:id="example.p.1">
        <s xml:id="example.p.1.s.1">
          <t>We demonstrated this earlier.</t>
        </s>
        <ref xml:id="example.ref.1" id="example.note.1" />
      </p>
      <note xml:id="example.note.1" class="footnote">
        <part>
            <t>See our website.</t>
        </part>
        <ref xml:id="example.ref.2" xlink:href="https://github.io/folia" xlink:type=
↪"simple" format="text/html" />
      </note>
  </text>
</FoLiA>
```

4.5.13 Reference Annotation

Structural annotation for referring to other annotation types. Used e.g. for referring to bibliography entries (citations) and footnotes.

Not to be confused with *Coreference Annotation*!

Specification

Annotation Category *Structure Annotation*

Declaration `<reference-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since v0.11, external references since v1.2

Element `<ref>`

API Class `Reference` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.

- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always `yes`, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to `no`. This allows for reconstruction of the detokenised original text.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.
- `xlink:href` – Turns this element into a hyperlink to the specified URL
- `xlink:type` – The type of link (you'll want to use `simple` in almost all cases).

Accepted Data `<alt>` (*Alternative Annotation*), `<altlayers>` (*Alternative Annotation*), `<comment>` (*Comment Annotation*), `<correction>` (*Correction Annotation*), `<desc>` (*Description Annotation*), `<external>` (*External Annotation*), `<hiddenw>` (*Hidden Token An-*

notation), `<br>` (*Linebreak*), `<metric>` (*Metric Annotation*), `<p>` (*Paragraph Annotation*), `<part>` (*Part Annotation*), `<ph>` (*Phonetic Annotation/Content*), `<quote>` (*Quote Annotation*), `<relation>` (*Relation Annotation*), `<s>` (*Sentence Annotation*), `<str>` (*String Annotation*), `<t>` (*Text Annotation*), `<utt>` (*Utterance Annotation*), `<whitespace>` (*Vertical Whitespace*), `<w>` (*Token Annotation*)

Valid Context `<def>` (*Definition Annotation*), `<div>` (*Division Annotation*), `<event>` (*Event Annotation*), `<ex>` (*Example Annotation*), `<head>` (*Head Annotation*), `<hiddenw>` (*Hidden Token Annotation*), `<list>` (*List Annotation*), `<note>` (*Note Annotation*), `<p>` (*Paragraph Annotation*), `<quote>` (*Quote Annotation*), `<s>` (*Sentence Annotation*), `<term>` (*Term Annotation*), `<utt>` (*Utterance Annotation*), `<w>` (*Token Annotation*)

Extra Attributes

- `id` – The ID of the element to link to
- `type` (optional) – The type of the element that is being linked to (e.g. `note`)

Element `<t-ref>`

API Class `TextMarkupReference` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.

- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.
- `xlink:href` – Turns this element into a hyperlink to the specified URL
- `xlink:type` – The type of link (you'll want to use `simple` in almost all cases).

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<br>` (*Linebreak*)

Valid Context

Extra Attributes

- `id` – The ID of the element to link to
- `type` (optional) – The type of the element that is being linked to (e.g. `note`)

Explanation & Examples

FoLiA allows for things like footnotes and bibliography entry using *Note Annotation*. In this section we show that you can make references to these notes using the `<ref>` element, this is a structure element, which implies that the references are explicitly present in the text. The `<ref>` element, however, carries an extra higher-order annotation function:

```
<s>
  <t>We demonstrated this earlier.</t>
</s>
<ref id="mynote" />
```

Another example in tokenised data, and now we add the *optional* `type` attribute, which holds the type of the FoLiA element that is referred to:

```
<s>
  <w><t>We</t></w>
  <w><t>demonstrated</t></w>
  <w><t>this</t></w>
  <w><t>earlier</t></w>
  <w><t>.</t></w>
  <ref id="mynote" type="note" />
</s>
```

You can optionally make explicit the symbol used for the reference. When no textual content is provided, whatever program renders the FoLiA document may assign its own numbering or symbol.

```
<s>
  <t>We demonstrated this earlier.</t>
</s>
<ref id="mynote" type="note"><t>1</t></ref>
```

This is often needed for bibliographical references:

```
<s>
  <t>We demonstrated this earlier.</t>
</s>
<ref id="bib.1" type="note"><t>(van Gompel et al, 2014)</t></ref>
```

As a structure element, the `<ref>` element may contain other structure elements such as words (*Token Annotation*) or even sentences (*Sentence Annotation*) or paragraphs (*Paragraph Annotation*), which can in turn contain further linguistic annotations.

Although we framed this section in the context of notes, the `<ref>` element is more general and can be used whereever you need to explicitly refer to other *structure elements*. Common targets are figures, tables, divisions (sections, chapters, etc).

Being a structure element, the note reference itself may carry an ID as well. Note that the ID attribute without the xml namespace always indicates a reference in FoLiA:

```
<s><t>We demonstrated this earlier.</t></s>
<ref xml:id="myreference" id="mynote" />
```

The difference between the reference element and the higher-order relations (*Relation Annotation*) needs to be clearly understood. Relation annotation lays relations between annotations of any kind and thus pertain strongly to linguistic annotation, whereas this reference element is a structural element that is explicitly shown in the text and draws a reference that is explicitly reflected in the text.

External references can also be made with the `<ref>` element, which effectively makes it a valid tool for *hyperlinking*. This is done by setting the `xlink:href` to point to the external resource and by setting the `format` attribute to the format of the external resource. The format is understood to be a MIME type and its value defaults to `text/folia+xml`. When an external reference is made, the `id` attribute is optional and points to an element inside the external resource.

```
<s>
  <w><t>We</t></w>
  <w><t>demonstrated</t></w>
  <w><t>this</t></w>
  <ref xlink:href="http://somewhere" xlink:type="simple"
    format="text/html" id="section2">
    <w><t>here</t></w>
  </ref>
  <w><t>.</t></w>
</s>
```

The `<ref>` element has a text-markup counterpart called `<t-ref>`, which can be used to link from untokenised text, both for internal and external links, as shown in the next two examples:

```
<s>
  <t>We demonstrated this earlier. <t-ref id="mynote" /></t>
</s>
```

```
<s>
  <t>We demonstrated this <t-ref xlink:href="http://somewhere" xlink:type="simple"␣
↪format="text/html" id="section2">here</t-ref>.</t>
</s>
```

The method of hyperlinking described in this section can be contrasted to the more generic one described in *Hyperlinks*. The `<ref>` (and `<t-ref>`) element offers a highly semantic way of hyperlinking, especially suited for explicitly linking to other internal FoLiA elements, whereas the other hyperlinking method is more of a text-markup or stylistic nature and more suited for external hyperlinks.

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" xmlns:xlink="http://www.w3.org/1999/xlink" version="2.0.4" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <note-annotation set="https://raw.githubusercontent.com/proycon/folia/master/setdefinitions/notes.foliaset.xml">
                         <annotator processor="p1" />
                  </note-annotation>
          <reference-annotation>
                         <annotator processor="p1" />
          </reference-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
                  </sentence-annotation>
          <paragraph-annotation>
                         <annotator processor="p1" />
                  </paragraph-annotation>
          <part-annotation>
                         <annotator processor="p1" />
          </part-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
      <p xml:id="example.p.1">
        <s xml:id="example.p.1.s.1">
          <t>We demonstrated this earlier.</t>
        </s>
        <ref xml:id="example.ref.1" id="example.note.1" />
      </p>
      <p xml:id="example.p.2">
        <s xml:id="example.p.1.s.2">
          <t>We demonstrated this earlier.<t-ref id="example.note.1" type="note" /></t>
        </s>
      </p>
      <note xml:id="example.note.1" class="footnote">
        <part>
            <t>See our website.</t>
        </part>
        <ref xml:id="example.ref.2" xlink:href="https://github.io/folia" xlink:type="simple" format="text/html" />
      </note>
      <note xml:id="example.note.2" class="footnote">
          <t>See our website.<t-ref xlink:href="https://github.io/folia" xlink:type="simple" format="text/html" /></t>
      </note>
  </text>
</FoLiA>
```

4.5.14 Table Annotation

Structural annotation type for creating a simple tabular environment, i.e. a table with rows, columns and cells and an optional header.

Specification

Annotation Category *Structure Annotation*

Declaration `<table-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History since v0.9.2

Element `<table>`

API Class `Table` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always `yes`, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to `no`. This allows for reconstruction of the detokenised original text.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.

- begintime – A timestamp in HH:MM:SS.MMM format, indicating the begin time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- endtime – A timestamp in HH:MM:SS.MMM format, indicating the end time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- speaker – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data <alt> (*Alternative Annotation*), <altlayers> (*Alternative Annotation*), <comment> (*Comment Annotation*), <correction> (*Correction Annotation*), <desc> (*Description Annotation*), <external> (*External Annotation*), <br> (*Linebreak*), <metric> (*Metric Annotation*), <part> (*Part Annotation*), <relation> (*Relation Annotation*)

Valid Context <def> (*Definition Annotation*), <div> (*Division Annotation*), <event> (*Event Annotation*), <ex> (*Example Annotation*), <note> (*Note Annotation*), <term> (*Term Annotation*)

Element <tablehead>

API Class TableHead (FoLiApy API Reference)

Required Attributes

Optional Attributes

- xml:id – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- set – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The set must be referred to also in the *Annotation Declarations* for this annotation type.
- class – The class of the annotation, i.e. the annotation tag in the vocabulary defined by set.
- processor – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- annotator – This is an older alternative to the processor attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- annotatortype – This is an older alternative to the processor attribute, without support for full provenance. It is used together with annotator and specific the type of the annotator, either manual for human annotators or auto for automated systems.
- confidence – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- datetime – The date and time when this annotation was recorded, the format is YYYY-MM-DDThh:mm:ss (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.

- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always `yes`, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to `no`. This allows for reconstruction of the detokenised original text.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<alt>` (*Alternative Annotation*), `<altlayers>` (*Alternative Annotation*), `<comment>` (*Comment Annotation*), `<correction>` (*Correction Annotation*), `<desc>` (*Description Annotation*), `<external>` (*External Annotation*), `<metric>` (*Metric Annotation*), `<part>` (*Part Annotation*), `<relation>` (*Relation Annotation*)

Valid Context `<table>` (*Table Annotation*)

XXX .. foliaspec:specification_element(Row) :**Element**: `<row>` :API Class: `Row` (FoLiApy API Reference) :Required Attributes: :Optional Attributes: * `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.

- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.

- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always `yes`, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to `no`. This allows for reconstruction of the detokenised original text.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<alt>` (*Alternative Annotation*), `<altlayers>` (*Alternative Annotation*), `<comment>` (*Comment Annotation*), `<correction>` (*Correction Annotation*), `<desc>` (*Description Annotation*), `<external>` (*External Annotation*), `<metric>` (*Metric Annotation*), `<part>` (*Part Annotation*), `<relation>` (*Relation Annotation*)

Valid Context `<table>` (*Table Annotation*)

Element `<cell>`

API Class `Cell` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.

- annotatortype – This is an older alternative to the processor attribute, without support for full provenance. It is used together with annotator and specific the type of the annotator, either manual for human annotators or auto for automated systems.
- confidence – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- datetime – The date and time when this annotation was recorded, the format is YYYY-MM-DDThh:mm:ss (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- n – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- space – This attribute indicates whether spacing should be inserted after this element (it's default value is always yes, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to no. This allows for reconstruction of the detokenised original text.
- src – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- begintime – A timestamp in HH:MM:SS.MMM format, indicating the begin time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- endtime – A timestamp in HH:MM:SS.MMM format, indicating the end time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- speaker – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data <alt> (*Alternative Annotation*), <altlayers> (*Alternative Annotation*), <comment> (*Comment Annotation*), <correction> (*Correction Annotation*), <desc> (*Description Annotation*), <entry> (*Entry Annotation*), <event> (*Event Annotation*), <ex> (*Example Annotation*), <external> (*External Annotation*), <figure> (*Figure Annotation*), <gap> (*Gap Annotation*), <head> (*Head Annotation*), <hiddenw> (*Hidden Token Annotation*), <br> (*Linebreak*), <list> (*List Annotation*), <metric> (*Metric Annotation*), <note> (*Note Annotation*), <p> (*Paragraph Annotation*), <part> (*Part Annotation*), <quote> (*Quote Annotation*), <ref> (*Reference Annotation*), <relation> (*Relation Annotation*), <s> (*Sentence Annotation*), <str> (*String Annotation*), <t> (*Text Annotation*), <whitespace> (*Vertical Whitespace*), <w> (*Token Annotation*)

Valid Context

Explanation

Support for simple tables is provided in a fashion similar to HTML and TEI. The element <table> introduces a table, within its scope row elements mark the various rows, <tablehead> marks the header of the table and contains one or more rows. The rows themselves consist of <cell> elements, which in turn may contain other structural elements such as words, sentences or even entire paragraphs.

Tables, rows and cells can all be assigned classes (in the same set).

Example

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <table-annotation>
                         <annotator processor="p1" />
                  </table-annotation>
          <token-annotation>
                         <annotator processor="p1" />
                  </token-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <table xml:id="example.table.1">
      <tablehead>
        <row>
          <cell>
            <w xml:id="example.table.1.w.1"><t>Name</t></w>
          </cell>
          <cell>
            <w xml:id="example.table.1.w.2"><t>Affiliation</t></w>
          </cell>
        </row>
      </tablehead>
      <row>
        <cell>
          <w xml:id="example.table.1.w.3"><t>Maarten van Gompel</t></w>
        </cell>
        <cell>
          <w xml:id="example.table.1.w.4">
            <t>Radboud University Nijmegen</t>
          </w>
        </cell>
      </row>
      <row>
        <cell>
          <w xml:id="example.table.1.w.5"><t>Ko van der Sloot</t></w>
        </cell>
        <cell>
          <w xml:id="example.table.1.w.6"><t>Radboud University Nijmegen</t></w>
        </cell>
      </row>
    </table>
  </text>
</FoLiA>
```

4.5.15 Part Annotation

The structure element `part` is a fairly abstract structure element that should only be used when a more specific structure element is not available. Most notably, the part element should never be used for representation of morphemes or phonemes! Part can be used to divide a larger structure element, such as a division, or a paragraph into arbitrary subparts.

Specification

Annotation Category *Structure Annotation*

Declaration `<part-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History since v0.11.2

Element `<part>`

API Class `Part` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always `yes`, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to `no`. This allows for reconstruction of the detokenised original text.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.

- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<alt>` (*Alternative Annotation*), `<altlayers>` (*Alternative Annotation*), `<comment>` (*Comment Annotation*), `<correction>` (*Correction Annotation*), `<desc>` (*Description Annotation*), `<external>` (*External Annotation*), `<metric>` (*Metric Annotation*), `<part>` (*Part Annotation*), `<ph>` (*Phonetic Annotation/Content*), `<relation>` (*Relation Annotation*), `<t>` (*Text Annotation*)

Valid Context `<def>` (*Definition Annotation*), `<div>` (*Division Annotation*), `<entry>` (*Entry Annotation*), `<event>` (*Event Annotation*), `<ex>` (*Example Annotation*), `<figure>` (*Figure Annotation*), `<gap>` (*Gap Annotation*), `<head>` (*Head Annotation*), `<hiddenw>` (*Hidden Token Annotation*), `<br>` (*Linebreak*), `<list>` (*List Annotation*), `<morpheme>` (*Morphological Annotation*), `<note>` (*Note Annotation*), `<p>` (*Paragraph Annotation*), `<part>` (*Part Annotation*), `<phoneme>` (*Phonological Annotation*), `<quote>` (*Quote Annotation*), `<ref>` (*Reference Annotation*), `<s>` (*Sentence Annotation*), `<table>` (*Table Annotation*), `<term>` (*Term Annotation*), `<utt>` (*Utterance Annotation*), `<whitespace>` (*Vertical Whitespace*), `<w>` (*Token Annotation*)

Explanation

Part can be used to divide a larger structure element, such as a division, or a paragraph into arbitrary subparts.

```xml
<p>
  <part xml:id="p.1.part.1">
    <t>First part of the paragraph.</t>
  </part>
  <part xml:id="p.2.part.2">
    <t>Last part of the paragraph.</t>
  </part>
</p>
```

The part element may seem alike to the division element, but divisions are typically used for text blocks larger than a paragraph, typically correspondings to chapters, sections or subsections and often carrying a `<head>` element. Do not use parts for these structures!

The part element, on the other hand, is more abstract and plays a role on a deeper level. It can be embedded within paragraphs, sentences, and most other structure elements, even words, though we have to again emphasize **it should not be used for morphology**, always use *Morphological Annotation* for that!

Contact the FoLiA authors if you find yourself using part and you feel a more specific FoLiA element is missing.

4.5.16 Utterance Annotation

An utterance is a structure element that may consist of words or sentences, which in turn may contain words. The opposite is also true, a sentence may consist of multiple utterances. Utterances are often used in the absence of sentences in a speech context, where neat grammatical sentences can not always be distinguished.

Specification

Annotation Category *Structure Annotation*

Declaration `<utterance-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History since v0.12

Element `<utt>`

API Class `Utterance` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always yes, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to `no`. This allows for reconstruction of the detokenised original text.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.

- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<alt>` (*Alternative Annotation*), `<altlayers>` (*Alternative Annotation*), `<comment>` (*Comment Annotation*), `<correction>` (*Correction Annotation*), `<desc>` (*Description Annotation*), `<external>` (*External Annotation*), `<gap>` (*Gap Annotation*), `<hiddenw>` (*Hidden Token Annotation*), `<metric>` (*Metric Annotation*), `<note>` (*Note Annotation*), `<part>` (*Part Annotation*), `<ph>` (*Phonetic Annotation/Content*), `<quote>` (*Quote Annotation*), `<ref>` (*Reference Annotation*), `<relation>` (*Relation Annotation*), `<s>` (*Sentence Annotation*), `<str>` (*String Annotation*), `<t>` (*Text Annotation*), `<w>` (*Token Annotation*)

Valid Context `<def>` (*Definition Annotation*), `<div>` (*Division Annotation*), `<event>` (*Event Annotation*), `<ex>` (*Example Annotation*), `<note>` (*Note Annotation*), `<quote>` (*Quote Annotation*), `<ref>` (*Reference Annotation*), `<term>` (*Term Annotation*)

Example

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <phon-annotation>
                         <annotator processor="p1" />
          </phon-annotation>
          <utterance-annotation>
                         <annotator processor="p1" />
          </utterance-annotation>
          <token-annotation>
                         <annotator processor="p1" />
          </token-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <speech xml:id="example.speech">
    <utt xml:id="example.utt.1" src="helloworld.mp3"  begintime="00:00:01.000" endtime="00:00:02.000">
       <ph>hel o  w ld</ph>
       <w xml:id="example.utt.1.w.1" begintime="00:00:00.000" endtime="00:00:01.000">
           <ph>hel o </ph>
```

(continues on next page)

(continued from previous page)

```
        </w>
        <w xml:id="example.utt.1.w.2" begintime="00:00:01.000" endtime="00:00:02.000">
            <ph>w ld</ph>
        </w>
    </utt>
  </speech>
</FoLiA>
```

4.5.17 Entry Annotation

FoLiA has a set of structure elements that can be used to represent collections such as glossaries, dictionaries, thesauri, and wordnets. *Entry annotation* defines the entries in such collections, *Term annotation* defines the terms, and *Definition Annotation* provides the definitions.

In this documentation we cover all four annotation types, as they are intimately connected.

Specification

Annotation Category *Structure Annotation*

Declaration `<entry-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History since v0.12

Element `<entry>`

API Class `Entry` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.

- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always `yes`, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to `no`. This allows for reconstruction of the detokenised original text.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<alt>` (*Alternative Annotation*), `<altlayers>` (*Alternative Annotation*), `<comment>` (*Comment Annotation*), `<correction>` (*Correction Annotation*), `<def>` (*Definition Annotation*), `<desc>` (*Description Annotation*), `<ex>` (*Example Annotation*), `<external>` (*External Annotation*), `<metric>` (*Metric Annotation*), `<part>` (*Part Annotation*), `<relation>` (*Relation Annotation*), `<str>` (*String Annotation*), `<term>` (*Term Annotation*), `<t>` (*Text Annotation*)

Valid Context `<div>` (*Division Annotation*), `<event>` (*Event Annotation*), `<p>` (*Paragraph Annotation*), `<s>` (*Sentence Annotation*)

Annotation Category *Structure Annotation*

Declaration `<term-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History since v0.12

Element `<term>`

API Class `Term` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.

- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always `yes`, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to `no`. This allows for reconstruction of the detokenised original text.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<alt>` (*Alternative Annotation*), `<altlayers>` (*Alternative Annotation*), `<comment>` (*Comment Annotation*), `<correction>` (*Correction Annotation*), `<desc>` (*Description Annotation*), `<event>` (*Event Annotation*), `<external>` (*External Annotation*), `<figure>` (*Figure Annotation*), `<gap>` (*Gap Annotation*), `<hiddenw>` (*Hidden Token Annotation*), `<br>` (*Linebreak*), `<list>` (*List Annotation*), `<metric>` (*Metric Annotation*), `<p>` (*Paragraph Annotation*), `<part>` (*Part Annotation*), `<ph>` (*Phonetic Annotation/Content*), `<ref>` (*Reference Annotation*), `<relation>` (*Relation Annotation*), `<s>` (*Sentence Annotation*), `<str>` (*String Annotation*), `<table>` (*Table Annotation*), `<t>` (*Text Annotation*),

<utt> (*Utterance Annotation*), <whitespace> (*Vertical Whitespace*), <w> (*Token Annotation*)

Valid Context <entry> (*Entry Annotation*)

Annotation Category *Structure Annotation*

Declaration `<definition-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History since v0.12

Element `<def>`

API Class `Definition` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always yes, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to `no`. This allows for reconstruction of the detokenised original text.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.

- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<alt>` (*Alternative Annotation*), `<altlayers>` (*Alternative Annotation*), `<comment>` (*Comment Annotation*), `<correction>` (*Correction Annotation*), `<desc>` (*Description Annotation*), `<external>` (*External Annotation*), `<figure>` (*Figure Annotation*), `<hiddenw>` (*Hidden Token Annotation*), `<br>` (*Linebreak*), `<list>` (*List Annotation*), `<metric>` (*Metric Annotation*), `<p>` (*Paragraph Annotation*), `<part>` (*Part Annotation*), `<ph>` (*Phonetic Annotation/Content*), `<ref>` (*Reference Annotation*), `<relation>` (*Relation Annotation*), `<s>` (*Sentence Annotation*), `<str>` (*String Annotation*), `<table>` (*Table Annotation*), `<t>` (*Text Annotation*), `<utt>` (*Utterance Annotation*), `<whitespace>` (*Vertical Whitespace*), `<w>` (*Token Annotation*)

Valid Context `<entry>` (*Entry Annotation*)

Annotation Category *Structure Annotation*

Declaration `<example-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History since v0.12

Element `<ex>`

API Class `Example` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.

- datetime – The date and time when this annotation was recorded, the format is YYYY-MM-DDThh:mm:ss (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- n – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- space – This attribute indicates whether spacing should be inserted after this element (it's default value is always yes, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to no. This allows for reconstruction of the detokenised original text.
- src – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- begintime – A timestamp in HH:MM:SS.MMM format, indicating the begin time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- endtime – A timestamp in HH:MM:SS.MMM format, indicating the end time of the speech. If a sound clip is specified (src); the timestamp refers to a location in the soundclip.
- speaker – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- tag – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use class and optionally features instead.

Accepted Data <alt> (*Alternative Annotation*), <altlayers> (*Alternative Annotation*), <comment> (*Comment Annotation*), <correction> (*Correction Annotation*), <desc> (*Description Annotation*), <external> (*External Annotation*), <figure> (*Figure Annotation*), <hiddenw> (*Hidden Token Annotation*), <br> (*Linebreak*), <list> (*List Annotation*), <metric> (*Metric Annotation*), <p> (*Paragraph Annotation*), <part> (*Part Annotation*), <ph> (*Phonetic Annotation/Content*), <ref> (*Reference Annotation*), <relation> (*Relation Annotation*), <s> (*Sentence Annotation*), <str> (*String Annotation*), <table> (*Table Annotation*), <t> (*Text Annotation*), <utt> (*Utterance Annotation*), <whitespace> (*Vertical Whitespace*), <w> (*Token Annotation*)

Valid Context <div> (*Division Annotation*), <entry> (*Entry Annotation*), <event> (*Event Annotation*), <note> (*Note Annotation*), <p> (*Paragraph Annotation*), <s> (*Sentence Annotation*)

Explanation

Collections such as glossaries, dictionaries, thesauri and wordnets have in common that they consist of a set of entries, which is represented in FoLiA by the <entry> element, and each entry is identified by one or more terms, represented by the <term> element within an entry.

Terms need not be words, but a wide variety of structural elements can be used as the term. Within the entry, these terms can subsequently be associated with one or more definitions, using the <def> element, or with examples, using the <ex> element.

The <term>, <def> and <ex> elements can all take sets and classes, and thus need to be declared. The <entry> elements themselves are simple containers and can contain multiple terms if they are deemed dependent or related, such as in case of morphological variants such as verb conjugations and declensions. The elements <term> and <def> can only be used within an <entry>. The <ex> element, however, can also be used standalone in different contexts.

In FoLiA, linguistic annotations are associated with the structure element within the term itself. This is where a glossary can for instance obtain part-of-speech or lexical semantic sense information, to name just a few examples.

Below you see an example of a glossary entry, the sense set used comes from WordNet. The other sets are fictitious.

```
<entry xml:id="entry.1">
 <term xml:id="entry.1.term.1">
  <w xml:id="entry.1.term.1.w.1">
    <t>house</t>
    <pos class="n">
      <feat subset="number" class="sing" />
    </pos>
    <lemma class="house" />
    <sense class="house\%1:06:00::">
  </w>
 </term>
 <term xml:id="entry.1.term.2">
  <w xml:id="entry.1.term.2.w.1">
    <t>houses/t>
    <pos class="n">
      <feat subset="number" class="plural" />
    </pos>
    <lemma class="house" />
    <sense class="house\%1:06:00::">
  </w>
 </term>
 <def xml:id="entry.1.def.1" class="sensedescription">
  <p xml:id="entry.1.def.1.p.1">
    <t>A dwelling, place of residence</t>
  </p>
 </def>
 <ex>
  <s xml:id="entry.1.ex.1.s.1>
    <t>My house was constructed ten years ago.</t>
  </s>
 </ex>
</entry>
```

Other semantic senses would be represented as separate entries.

The definitions (<def>) are a generic element that can be used for multiple types of definition. As always, the set is not predefined and purely fictitious in our examples, giving the user flexibility. Definitions are for instance suited for dictionaries:

```
<entry xml:id="entry.1">
 <term xml:id="entry.1.term.1">
  <w xml:id="entry.1.term.1.w.1">
    <t>house</t>
    <pos set="englishpos" class="n">
      <feat subset="number" class="sing" />
    </pos>
    <lemma set="englishlemma" class="house" />
    <sense set="englishsense" class="house\%1:06:00::">
  </w>
```

(continues on next page)

(continued from previous page)

```
 </term>
 <def xml:id="entry.1.def.1" class="translation-es">
  <w xml:id="entry.1.def.1.w.1">
    <t>casa</t>
    <pos set="spanishpos"  class="n">
      <feat subset="number" class="sing" />
    </pos>
    <lemma set="spanishlemma" class="casa" />
  </w>
 </def>
</entry>
```

Or for etymological definitions:

```
<def xml:id="entry.1.def.2" class="etymology">
 <p xml:id="entry.1.def.2.p.1">
  <t>Old English hus "dwelling, shelter, house," from Proto-Germanic *husan
(cognates: Old Norse, Old Frisian hus, Dutch huis, German Haus), of unknown
origin, perhaps connected to the root of hide (v.) [OED]. In Gothic only in
gudhus "temple," literally "god-house;" the usual word for "house" in Gothic
being razn.  </t>
 </p>
</def>
```

The following two samples illustrate a dictionary distributed over multiple FoLiA files, using *Relation Annotation* to link the two:

English part, `doc-english.xml` (excerpt):

```
<entry xml:id="en-entry.1">
 <term xml:id="en-entry.1.term.1">
  <w xml:id="en-entry.1.term.1.w.1">
    <t>house</t>
    <pos set="englishpos" class="n">
      <feat subset="number" class="sing" />
    </pos>
    <lemma set="englishlemma" class="house" />
    <sense set="englishsense" class="house\%1:06:00::">
  </w>
 </term>
 <relation class="translation-es" xlink:href="doc-spanish.xml"
    xlink:type="simple">
      <xref id="es-entry.1" type="entry" />
 </relation>
</entry>
```

Spanish part, `doc-spanish.xml` (excerpt):

```
<entry xml:id="es-entry.1">
 <term xml:id="es-entry.1.def.1" class="translation-es">
  <w xml:id="entry.1.def.1.w.1">
    <t>casa</t>
    <pos set="spanishpos"  class="n">
      <feat subset="number" class="sing" />
    </pos>
    <lemma set="spanishlemma" class="casa" />
  </w>
```

(continues on next page)

(continued from previous page)

```
 </term>
 <relation class="translation-en" xlink:href="doc-english.xml"
    xlink:type="simple">
      <xref id="en-entry.1" type="entry" />
 </relation>
</entry>
```

For simple multilingual documents, explicit relations may be too much hassle, For situations where this seems overkill, a simple multi-document mechanism is available. This mechanism is based purely on convention: It assumes that structural elements that are translations simply share the same ID. This approach is quite feasible when used on higher-level structural elements, such as divisions, paragraphs, events or entries.

4.5.18 Term Annotation

See also:

This annotatino type is covered by the documentation on *Entry Annotation*

4.5.19 Definition Annotation

See also:

This annotation type is covered by the documentation on *Entry Annotation*

4.5.20 Example Annotation

See also:

This annotation type is covered by the documentation on *Entry Annotation*

4.5.21 Hidden Token Annotation

This annotation type allows for a hidden token layer in the document. Hidden tokens are ignored for most intents and purposes but may serve a purpose when annotations on implicit tokens is required, for example as targets for syntactic movement annotation.

Specification

Annotation Category *Structure Annotation*

Declaration `<hiddentoken-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since v2.0

Element `<hiddenw>`

API Class `Hiddenword` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.

- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `textclass` – Refers to the text class this annotation is based on. This is an advanced attribute, if not specified, it defaults to `current`. See *Text class attribute (advanced)*.
- `space` – This attribute indicates whether spacing should be inserted after this element (it's default value is always `yes`, so it does not need to be specified in that case), but if tokens or other structural elements are glued together then the value should be set to `no`. This allows for reconstruction of the detokenised original text.
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<alt>` (*Alternative Annotation*), `<altlayers>` (*Alternative Annotation*), `<comment>` (*Comment Annotation*), `<correction>` (*Correction Annotation*), `<desc>` (*Description Annotation*), `<external>` (*External Annotation*), `<metric>` (*Metric Annotation*), `<part>` (*Part Annotation*), `<ph>` (*Phonetic Annotation/Content*), `<ref>` (*Reference Annotation*), `<relation>` (*Relation Annotation*), `<str>` (*String Annotation*), `<t>` (*Text Annotation*)

Valid Context `<def>` (*Definition Annotation*), `<event>` (*Event Annotation*), `<ex>` (*Example Annotation*), `<head>` (*Head Annotation*), `<note>` (*Note Annotation*), `<p>` (*Paragraph Annotation*), `<quote>` (*Quote Annotation*), `<ref>` (*Reference Annotation*), `<s>` (*Sentence Annotation*), `<term>` (*Term Annotation*), `<utt>` (*Utterance Annotation*)

Explanation

Hidden tokens are tokens that are explicitly not part of the original text. They are either implied or tokens that act as a dummy for further linguistic annotation. Hidden tokens are are valid target for any form of span annotation through the `<wref>` element (see **:ref:‘span_annotation_category‘_**). They are structure elements so may appear interleaved with the normal tokenisation layer, for which the order is significant.

Example

The following example shows syntactic movement annotation which makes use of a hidden token for an implicit subject.

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <token-annotation set="https://raw.githubusercontent.com/LanguageMachines/
↪uctodata/master/setdefinitions/tokconfig-eng.foliaset.ttl">
                         <annotator processor="p1" />
                  </token-annotation>
          <hiddentoken-annotation>
                         <annotator processor="p1" />
                  </hiddentoken-annotation>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
          </sentence-annotation>
          <pos-annotation set="adhoc">
                         <annotator processor="p1" />
          </pos-annotation>
          <syntax-annotation set="adhoc">
                         <annotator processor="p1" />
          </syntax-annotation>
          <description-annotation>
                         <annotator processor="p1" />
          </description-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
    <s xml:id="example.s.1">
      <t>Isn't a whole lot left.</t>
        <hiddenw xml:id="example.s.1.w.0">
            <t>*exp*</t>
            <desc>empty expletive subject</desc>
            <pos class="EX" />
        </hiddenw>
```

(continues on next page)

(continued from previous page)

```
        <w xml:id="example.s.1.w.1" space="no">
            <t>Is</t>
            <pos class="BEP" />
        </w>
        <w xml:id="example.s.1.w.2">
            <t>n't</t>
            <pos class="NEG" />
        </w>
        <w xml:id="example.s.1.w.3">
            <t>a</t>
            <pos class="D" />
        </w>
        <w xml:id="example.s.1.w.4">
            <t>whole</t>
            <pos class="ADJ" />
        </w>
        <w xml:id="example.s.1.w.5">
            <t>lot</t>
            <pos class="N" />
        </w>
        <w xml:id="example.s.1.w.6" space="no">
            <t>left</t>
            <pos class="VAN" />
        </w>
        <w xml:id="example.s.1.w.7">
            <t>.</t>
            <pos class="PUNC" />
        </w>
        <syntax>
            <su xml:id="example.s.1.su.1" class="IP-MAT">
                <su xml:id="example.s.1.su.2" class="NP-SBJ">
                    <wref id="example.s.1.w.0" />
                </su>
                <su xml:id="example.s.1.su.3" class="VP">
                    <su xml:id="example.s.1.su.4" class="BEP">
                        <wref id="example.s.1.w.1" />
                    </su>
                    <su xml:id="example.s.1.su.5" class="NEG">
                        <wref id="example.s.1.w.2" />
                    </su>
                    <su xml:id="example.s.1.su.6" class="VP">
                        <su xml:id="example.s.1.su.7" class="NP-LGS">
                            <wref id="example.s.1.w.3" />
                            <su xml:id="example.s.1.su.8" class="ADJP">
                                <wref id="example.s.1.w.4" />
                            </su>
                            <wref id="example.s.1.w.5" />
                        </su>
                        <wref id="example.s.1.w.6" />
                    </su>
                </su>
                <su class="PUNC">
                    <wref id="example.s.1.w.7" />
                </su>
            </su>
        </syntax>
```

(continues on next page)

(continued from previous page)

```
    </s>
  </text>
</FoLiA>
```

4.6 Subtoken Annotation

This category contains morphological annotation and phonological annotation, i.e. the segmentation of a word into morphemes and phonemes, and recursively so if desired. It is a special category that mixes characteristics from structure annotation (the `morpheme` and `phoneme` elements are very structure-like) and also from span annotation, as morphemes and phonemes are embedded in an annotation layer and refer back to the text/phonetic content they apply to. Like words/tokens, these elements may also be referenced from `wref` elements.

FoLiA defines the following types of subtoken annotation:

- *Subtoken Annotation* – This category contains morphological annotation and phonological annotation, i.e. the segmentation of a word into morphemes and phonemes, and recursively so if desired. It is a special category that mixes characteristics from structure annotation (the `morpheme` and `phoneme` elements are very structure-like) and also from span annotation, as morphemes and phonemes are embedded in an annotation layer and refer back to the text/phonetic content they apply to. Like words/tokens, these elements may also be referenced from `wref` elements.
  - *Morphological Annotation* – `<morpheme>` – Morphological Annotation allows splitting a word/token into morphemes, morphemes itself may be nested. It is embedded within a layer `morphology` which can be embedded within word/tokens.
  - *Phonological Annotation* – `<phoneme>` – The smallest unit of annotatable speech in FoLiA is the phoneme level. The phoneme element is a form of structure annotation used for phonemes. Alike to morphology, it is embedded within a layer `phonology` which can be embedded within word/tokens.

4.6.1 Morphological Annotation

Morphological Annotation allows splitting a word/token into morphemes, morphemes itself may be nested. It is embedded within a layer `morphology` which can be embedded within word/tokens.

Specification

Annotation Category *Subtoken Annotation*

Declaration `<morphological-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Heavily revised since v0.9

Element `<morpheme>`

API Class `Morpheme` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.

- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<alt>` (*Alternative Annotation*), `<altlayers>` (*Alternative Annotation*), `<comment>` (*Comment Annotation*), `<correction>` (*Correction Annotation*), `<desc>` (*Description Annotation*), `<metric>` (*Metric Annotation*), `<morpheme>` (*Morphological Annotation*), `<part>` (*Part Annotation*), `<ph>` (*Phonetic Annotation/Content*), `<relation>` (*Relation Annotation*), `<str>` (*String Annotation*), `<t>` (*Text Annotation*)

Valid Context `<morpheme>` (*Morphological Annotation*), `<morphology>` (*Morphological Annotation*)

Feature subsets (extra attributes)

- `function`

Explanation

Tokens can be further segmented into morphemes, a form of structure annotation. Morphemes behave much like `<w>` elements (tokens). Moreover, morphemes can be referred to from within in span annotation using `<wref>`, allowing spans to be defined not only over whole words/tokens but also parts thereof. The element for morphemes is `<morpheme>`, and can only occur within `<w>` elements. Recall that `<t>` elements can contain references to higher-level `<t>` elements. In such cases, the `offset` attribute is used to designate the offset index in the word's associated text element (`<t>`)' (zero being right at the start of the text). Morphemes may do this.

Furthermore, a morpheme may take a class in a user-defined set, referring to its type.

Morphemes are grouped in a `morphology` layer, in turn embedded in a word, this is analogous to *Span Annotation*.

Consider the following example:

```
<w xml:id="example.p.4.s.2.w.4">
    <t>leest</t>
    <lemma class="lezen" />
    <morphology>
        <morpheme class="stem" function="lexical">
            <t offset="0">lees</t>
        </morpheme>
        <morpheme class="suffix" function="inflexional">
            <t offset="4">t</t>
        </morpheme>
    </morphology>
</w>
```

There is a predefined *feature subset* (see *Features*) which you can use with morphemes, it is called `function` and denotes the function of the morpheme, the class it takes is defined by the particular set used.

Morphemes allow the same kinds of inline annotation just as words do. We can for instance bind lemma annotation to the morpheme representing the word's stem rather than only to the entire word:

```
<w xml:id="example.p.4.s.2.w.4">
    <t>leest</t>
    <lemma class="lezen" />
    <morphology>
        <morpheme xml:id="example.p.4.s.2.w.4.m.1" class="stem"
         function="lexical">
            <lemma class="lezen" />
            <t offset="0">lees</t>
        </morpheme>
        <morpheme xml:id="example.p.4.s.2.w.4.m.2" class="suffix"
         function="inflexional">
            <t offset="4">t</t>
        </morpheme>
    </morphology>
</w>
```

Similarly, consider the Spanish word or phrase "Dámelo" (give it to me), written as one entity. If this has not been split during tokenisation, but left as a single token, you can annotate its morphemes, as all morphemes allow token annotation to be placed within their scope:

```
<w xml:id="example.p.1.s.1.w.1">
    <t>dámelo</t>
    <morphology>
        <morpheme class="stem">
```

(continues on next page)

(continued from previous page)

```
            <t offset="0">dá</t>
            <lemma class="dar" />
            <pos class="v" />
        </morpheme>
        <morpheme class="suffix">
            <t offset="2">me</t>
            <lemma class="me" />
            <pos class="pron" />
        </morpheme>
        <morpheme class="suffix">
            <t offset="4">lo</t>
            <lemma class="lo" />
            <pos class="pron" />
        </morpheme>
    </morphology>
</w>
```

Unlike words, but similar to *Syntactic Annotation*, morphemes may also be nested, as they can be expressed on multiple levels:

```
<w xml:id="example.p.1.s.1.w.1">
    <t>comfortable</t>
    <morphology>
        <morpheme class="base">
            <t offset="0">comfort</t>
            <morpheme class="prefix">
                <t offset="0">com</t>
            </morpheme>
            <morpheme class="morph">
                <t offset="3">fort</t>
            </morpheme>
        </morpheme>
        <morpheme class="suffix">
            <t offset="7">able</t>
        </morpheme>
    </morphology>
</w>
```

The next example will illustrate how morphemes can be referred to in span annotation. Here we have a morpheme, and not the entire word, which forms a named entity:

```
<w xml:id="example.p.4.s.2.w.4">
    <t>CDA-voorzitter</t>
    <morphemes>
        <morpheme xml:id="example.p.4.s.2.w.1.m.1">
            <t offset="0">CDA</t>
        </morpheme>
    </morphemes>
    <entities>
        <entity xml:id="entity.1" class="organisation">
            <wref id="example.p.4.s.2.w.1.m.1" t="CDA" />
        </entity>
    </entities>
</w>
```

The same approach can be followed for other kinds of span annotation. Note that the span annotation layer (`<entities>` in the example) may be embedded on various levels. Most commonly on sentence level, but also on word level, paragraph level or the global text level.

4.6.2 Phonological Annotation

The smallest unit of annotatable speech in FoLiA is the phoneme level. The phoneme element is a form of structure annotation used for phonemes. Alike to morphology, it is embedded within a layer `phonology` which can be embedded within word/tokens.

Specification

Annotation Category *Subtoken Annotation*

Declaration `<phonological-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History since v0.12

Element `<phoneme>`

API Class `Phoneme` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.

- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.

Accepted Data `<alt>` (*Alternative Annotation*), `<altlayers>` (*Alternative Annotation*), `<comment>` (*Comment Annotation*), `<correction>` (*Correction Annotation*), `<desc>` (*Description Annotation*), `<metric>` (*Metric Annotation*), `<part>` (*Part Annotation*), `<ph>` (*Phonetic Annotation/Content*), `<phoneme>` (*Phonological Annotation*), `<relation>` (*Relation Annotation*), `<str>` (*String Annotation*), `<t>` (*Text Annotation*)

Valid Context `<phoneme>` (*Phonological Annotation*), `<phonology>` (*Phonological Annotation*)

Feature subsets (extra attributes)

- `function`

Explanation & Example

The smallest unit of annotatable speech in FoLiA is the phoneme level. The `<phoneme>` element is a form of subtoken annotation used for phonemes.

Very much alike to morphology, it is embedded within a layer `<phonology>` which can be used within word/token elements (`<w>`) or directly within higher structure such as utterances (`<utt>`) if no words are distinguished:

```xml
<utt>
  <w xml:id="word" src="book.wav">
    <t>book</t>
    <ph>b k</ph>
    <phonology>
      <phoneme begintime="..."  endtime="...">
          <ph>b</ph>
      </phoneme>
      <phoneme begintime="..." endtime="...">
          <ph> </ph>
      </phoneme>
      <phoneme begintime="..." endtime="...">
          <ph>k</ph>
      </phoneme>
    </phonology>
  </w>
</utt>
```

4.7 Text Markup Annotation

The text content element (`<t>`) allows within its scope elements of a this category; these are **Text Markup** elements, they always contain textual content and apply a certain markup to certain spans of the text. One of it's common uses is for styling (emphasis, underlines, etc.). Text markup elements may be nested.

Text markup elements may take sets and classes as most other elements, and any of the remaining common FoLiA attributes may be freely associated with any of the text-markup elements.

As text markup operates in the scope of the text content element, it is ideally suited for untokenised text. You should, however, limit your usage of text markup elements and only use them when other existing annotation elements do not suffice, or for extra verbosity in addition to existing elements.

Each text-markup element, save for one exception, starts with `<t-` and demands a declaration. The following subsections will discuss the various text markup elements available.

Text markup elements may carry an optional identifier. However, it may happen that textual content is repeated on multiple levels (see *Text Annotation*), which implies the textual markup elements may also be repeated, but this is not a strict requirement. However, if that is the case, only one of them may carry the customary `xml:id` attribute; duplicates may carry the `id` *reference* attribute (in the FoLiA namespace instead of the XML namespace!) which is interpreted as a reference. Such an element should be identical to the one it refers to, and explicitly include the value (if applicable) to facilitate the job of XML parsers. Certain elements may also use this `id` reference attribute to refer to structural elements that cover the very same data. A markup element may thus take either `xml:id` or `id` (a reference to another element); they may never occur together.

FoLiA defines the following types of text markup annotation:

- *Text Markup Annotation* – The text content element (`<t>`) allows within its scope elements of a this category; these are **Text Markup** elements, they always contain textual content and apply a certain markup to certain spans of the text. One of it's common uses is for styling (emphasis, underlines, etc.). Text markup elements may be nested.
  - *Style Annotation* – `<t-style>` – This is a text markup annotation type for applying styling to text. The actual styling is defined by the user-defined set definition and can for example included classes such as italics, bold, underline
  - *Hyphenation* – `<t-hbr>` – This is a text-markup annotation form that indicates where in the original text a linebreak was inserted and a word was hyphenised.
  - *Horizontal Whitespace* – `<t-hspace>` – Markup annotation introducing horizontal whitespace

4.7.1 Style Annotation

This is a text markup annotation type for applying styling to text. The actual styling is defined by the user-defined set definition and can for example included classes such as italics, bold, underline

Specification

Annotation Category *Text Markup Annotation*

Declaration `<style-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History since v0.10

Element `<t-style>`

API Class `TextMarkupStyle` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.
- `xlink:href` – Turns this element into a hyperlink to the specified URL
- `xlink:type` – The type of link (you'll want to use `simple` in almost all cases).

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<br>` (*Linebreak*)

Valid Context

Feature subsets (extra attributes)

- font
- size

Explanation

The text markup element `<t-style>` marks a specific portion of textual concent to be rendered in a specific style. Styles in turn are simply classes in your set.

Text-markup elements may always be nested.

Example

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <sentence-annotation>
                         <annotator processor="p1" />
                  </sentence-annotation>
          <linebreak-annotation>
                         <annotator processor="p1" />
                  </linebreak-annotation>
          <part-annotation>
                         <annotator processor="p1" />
                  </part-annotation>
          <style-annotation set="https://raw.githubusercontent.com/proycon/folia/
↪master/setdefinitions/styles.foliaset.xml">
                         <annotator processor="p1" />
                  </style-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
        <s>
            <t>To <t-style class="italic">be</t-style> or not to be,<br/>that is the
↪<t-style class="bold"><t-style class="red">question</t-style></t-style>.</t>
        </s>
  </text>
</FoLiA>
```

In the next example, features are used rather than nested styles:

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.4.2" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
```

(continues on next page)

(continued from previous page)

```
        <sentence-annotation>
                        <annotator processor="p1" />
                </sentence-annotation>
        <linebreak-annotation>
                        <annotator processor="p1" />
                </linebreak-annotation>
        <part-annotation>
                        <annotator processor="p1" />
                </part-annotation>
        <style-annotation set="https://raw.githubusercontent.com/proycon/folia/
↪master/setdefinitions/styles.foliaset.xml">
                        <annotator processor="p1" />
                </style-annotation>
    </annotations>
    <provenance>
       <processor xml:id="p1" name="proycon" type="manual" />
    </provenance>
  </metadata>
  <text xml:id="example.text">
        <s>
            <t>To <t-style class="italic">be</t-style> or not to be,<br/>that is the
↪<t-style class="bold"><feat subset="color" class="red"/>question</t-style>.</t>
        </s>
  </text>
</FoLiA>
```

4.7.2 Hyphenation

This is a text-markup annotation form that indicates where in the original text a linebreak was inserted and a word was hyphenised.

Specification

Annotation Category *Text Markup Annotation*

Declaration `<hyphenation-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since v2.0

Element `<t-hbr>`

API Class `Hyphbreak` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.

- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.
- `xlink:href` – Turns this element into a hyperlink to the specified URL
- `xlink:type` – The type of link (you'll want to use `simple` in almost all cases).

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<br>` (*Linebreak*)

Valid Context

Extra Attributes

- `newpage` – Can be set to yes to indicate that the break is not just a linebreak, but also a pagebreak (defaults to `no`)
- `pagenr` – The number of the page after the break
- `linenr` – The number of the line after the break

Description & Examples

Hyphenation breaks are a text markup element that indicate where in the original text a word was broken up for line-wrapping purposes.

The difference between `t-hbr` and `br` (*Linebreak*) is that the hyphenised break is a softer break, only there for page formatting purposes. The hyphen symbol is by definition implied in its usage, so should never be explicitly incorporated in the text content. For most intents and purposes, a word with a hyphenised break can be considered semantically identical to a word without one. The following example demonstrates hyphenation in the last division, alongside the more classical linebreak:

```xml
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <division-annotation set="https://raw.githubusercontent.com/
↪LanguageMachines/uctodata/master/setdefinitions/divisions.foliaset.xml">
                         <annotator processor="p1" />
                  </division-annotation>
          <whitespace-annotation>
                         <annotator processor="p1" />
                  </whitespace-annotation>
          <linebreak-annotation>
                         <annotator processor="p1" />
          </linebreak-annotation>
          <hyphenation-annotation>
                         <annotator processor="p1" />
         </hyphenation-annotation>
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
     <div xml:id="example.div.1" class="section" n="1">
         <t>Blah...</t>
     </div>
     <whitespace />
     <br newpage="yes" pagenr="2" />
     <div xml:id="example.div.2" class="section" n="2">
         <!-- BR has a double role, it can be used a text markup element as well, as␣
↪seen on the next line -->
         <t>To be, <br />or not to be!</t>
     </div>
     <div xml:id="example.div.3" class="section" n="3">
         <t>Don't leave me bro<t-hbr/>ken and alone!</t>
     </div>
  </text>
</FoLiA>
```

4.7.3 Horizontal Whitespace

Markup annotation introducing horizontal whitespace

Note: Do not confuse this with the `<whitespace>` structure element and `<t-whitespace>` markup element that are used for *vertical* whitespace, see *Vertical Whitespace*.

Specification

Annotation Category *Text Markup Annotation*

Declaration `<hspace-annotation set="...">` *(note: set is optional for this annotation type; if you declare this annotation type to be setless you can not assign classes)*

Version History Since the v2.5.0

Element `<t-hspace>`

API Class `TextMarkupHSpace` (FoLiApy API Reference)

Required Attributes

Optional Attributes

- `xml:id` – The ID of the element; this has to be a unique in the entire document or collection of documents (corpus). All identifiers in FoLiA are of the XML NCName datatype, which roughly means it is a unique string that has to start with a letter (not a number or symbol), may contain numbers, but may never contain colons or spaces. FoLiA does not define any naming convention for IDs.
- `set` – The set of the element, ideally a URI linking to a set definition (see *Set Definitions (Vocabulary)*) or otherwise a uniquely identifying string. The `set` must be referred to also in the *Annotation Declarations* for this annotation type.
- `class` – The class of the annotation, i.e. the annotation tag in the vocabulary defined by `set`.
- `processor` – This refers to the ID of a processor in the provenance_data. The processor in turn defines exactly who or what was the annotator of the annotation.
- `annotator` – This is an older alternative to the `processor` attribute, without support for full provenance. The annotator attribute simply refers to the name o ID of the system or human annotator that made the annotation.
- `annotatortype` – This is an older alternative to the `processor` attribute, without support for full provenance. It is used together with `annotator` and specific the type of the annotator, either `manual` for human annotators or `auto` for automated systems.
- `confidence` – A floating point value between zero and one; expresses the confidence the annotator places in his annotation.
- `datetime` – The date and time when this annotation was recorded, the format is `YYYY-MM-DDThh:mm:ss` (note the literal T in the middle to separate date from time), as per the XSD Datetime data type.
- `n` – A number in a sequence, corresponding to a number in the original document, for example chapter numbers, section numbers, list item numbers. This this not have to be an actual number but other sequence identifiers are also possible (think alphanumeric characters or roman numerals).
- `src` – Points to a file or full URL of a sound or video file. This attribute is inheritable.
- `begintime` – A timestamp in `HH:MM:SS.MMM` format, indicating the begin time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.
- `endtime` – A timestamp in `HH:MM:SS.MMM` format, indicating the end time of the speech. If a sound clip is specified (`src`); the timestamp refers to a location in the soundclip.

- `speaker` – A string identifying the speaker. This attribute is inheritable. Multiple speakers are not allowed, simply do not specify a speaker on a certain level if you are unable to link the speech to a specific (single) speaker.
- `tag` – Contains a space separated list of processing tags associated with the element. A processing tag carries arbitrary user-defined information that may aid in processing a document. It may carry cues on how a specific tool should treat a specific element. The tag vocabulary is specific to the tool that processes the document. Tags carry no instrinsic meaning for the data representation and should not be used except to inform/aid processors in their task. Processors are encouraged to clean up the tags they use. Ideally, published FoLiA documents at the end of a processing pipeline carry no further tags. For encoding actual data, use `class` and optionally features instead.
- `xlink:href` – Turns this element into a hyperlink to the specified URL
- `xlink:type` – The type of link (you'll want to use `simple` in almost all cases).

Accepted Data `<comment>` (*Comment Annotation*), `<desc>` (*Description Annotation*), `<br>` (*Linebreak*)

Valid Context

Description & Examples

If normal spacing is not enough and you need to express **horizontal** whitespace explicitly, then you can use the `<t-hspace>` element.

```
<t>To be<t-hspace class="long" />or not to be</t>
```

The vocabulary is defined by your set definition and you can assign your own size-interpretation. Tools that are not aware of your vocabulary should simply render a single space.

An alternative to `t-hspace` is to use the `xml:space="preserve"` attribute as described in *Preserving whitespace (advanced)*, but the use of `<t-hspace>` is preferred.

The last section in this example shows horizontal whitespace:

```
<?xml version="1.0" encoding="utf-8"?>
<FoLiA xmlns="http://ilk.uvt.nl/folia" version="2.5.0" xml:id="example">
  <metadata>
      <annotations>
          <text-annotation>
                         <annotator processor="p1" />
          </text-annotation>
          <division-annotation set="https://raw.githubusercontent.com/
↪LanguageMachines/uctodata/master/setdefinitions/divisions.foliaset.xml">
                         <annotator processor="p1" />
                 </division-annotation>
          <whitespace-annotation>
                         <annotator processor="p1" />
                 </whitespace-annotation>
          <hspace-annotation>
                         <annotator processor="p1" />
                 </hspace-annotation>
          <linebreak-annotation>
                         <annotator processor="p1" />
          </linebreak-annotation>
          <hyphenation-annotation>
                         <annotator processor="p1" />
         </hyphenation-annotation>
```

(continues on next page)

(continued from previous page)

```xml
      </annotations>
      <provenance>
         <processor xml:id="p1" name="proycon" type="manual" />
      </provenance>
  </metadata>
  <text xml:id="example.text">
     <div xml:id="example.div.1" class="section" n="1">
         <t>Blah...</t>
     </div>
     <whitespace />
     <br newpage="yes" pagenr="2" />
     <div xml:id="example.div.2" class="section" n="2">
         <!-- BR has a double role, it can be used a text markup element as well, as seen on the next line -->
         <t>To be, <br />or not to be!</t>
     </div>
     <div xml:id="example.div.3" class="section" n="3">
         <t>Don't leave me bro<t-hbr/>ken and alone!</t>
     </div>
     <div xml:id="example.div.4" class="section" n="4">
         <t>Space,<t-hspace/>the<t-hspace/>final<t-hspace/>
            frontier</t>
     </div>
  </text>
</FoLiA>
```

CHAPTER 5

Foreign Annotation

It may happen that you want to include annotations inside your FoLiA that are not actually in FoLiA, but in some other XML format. Though **this is very much discouraged**, even more so if FoLiA has proper facilities for your annotation needs, it may in rare cases be needed; for example if FoLiA has no support yet for a particular advanced type of annotation or if another scheme has already been in use and conversion is not an option. It is most suitable for attaching further data to arbitrary elements, though for metadata *Submetadata* should always be considered first!

The higher-order annotation element `<foreign-data>` can be used to accomplish foreign annotations. It acts as a container in which annotation must be in a different XML namespace, rather than the FoLiA namespace. The element is allowed almost anywhere: inside structure annotation, inside inline/span annotation, inside other higher annotation elements, but **not** inside text content (`<t>`), phonetic content (`<ph>`) or text-markup (`<t-*>`).

In the following example we attach an annotation in a custom fictitious XML format and namespace to a FoLiA word:

```
<w xml:id="w.1">
   <t>Hello</t>
   <foreign-data xmlns:myformat="http://my.com/custom/format">
      <myformat:myannotation myattribute="myvalue" />
   </foreign-data>
</w>
```

Foreign annotation does not need to be declared and, as can not be emphasised enough, should really only be used when no proper FoLiA solution exists, and even in such cases it is preferable to contact the FoLiA developers and see if FoLiA can be extended for your needs.. Be aware that generic FoLiA tools and libraries will usually not process the contents of foreign-data, as it can contain anything by definition, and special-purpose tools need to be written for your specific use case if you use foreign-data.

CHAPTER 6

Querying

6.1 XPath

Considering the fact that FoLiA is an XML-based format, XPath and its derivatives are available as tools for searching in a FoLiA document.

Some common XPath queries are listed below, note that for the sake of brevity and readability the namespace prefix is omitted. In actual situations you will have to specify the FoLiA namespace with each element, as XPath unfortunately has no notion of a default namespace.

- XPath query for all paragraphs: `//p`
- XPath query for all sentences: `//s`
- XPath query for all words/tokens: `//w`
- XPath query for all words with lemma X: `//w[.//lemma[@class="X"]`

This seems simple, but there are important caveats. When formulating XPath queries, however, one needs to be well aware of how FoLiA works, as XPath is a generic tool that can not take care of specific FoLiA ideosyncracies for you, unlike *FoLiA Query Language (FQL)* or the FoLiA programming libraries. These simple queries will be insufficient when dealing with a document containing *Correction Annotation*, *Alternative Annotation* or even *Quote Annotation*. You can rely on the *Annotation Declarations* to know whether this is the case. To formulate queries that work in all cases, you need to be aware of the exceptions.

For example, if we query all words as `//w` in a document that contains structural corrections, we also get the original words prior to correction. If we query for lemmas as `//lemma[@class="X"]` and our document has alternative annotations, we may end up also getting lemmas that were specified as an alternative. This is of course fine if this is what you want, but you need to be aware of it. A construct you will often see in FoLiA XPath Queries is `not(ancestor-or-self::*/X)`, where X is a particular FoLiA element.

Consider the following more thought-out and more generic queries:

- XPath query for the text of all words/tokens: *//w//t[not(ancestor-or-self::*/original) and not(ancestor-or-self::*/suggestion) and not(ancestor-or-self::*/alt) and not(ancestor-or-self::*/morpheme) and not(ancestor-or-self::*/str) and not(@class)]//text()‘*
  - Explanation: The `not(@class)` predicate is important here

  and makes sure to select only the current text content element in case there are multiple text content elements in different classes. (See *Text Annotation*). The `not(ancestor-or-self::*/morpheme` makes sure morphemes are excluded, `not(ancestor-or-self::*/str)` makes sure strings are excluded, `not(ancestor-or-self::*/alt)` makes sure alternatives are excluded.

- **XPath query for all words with PoS-tag A in set S:** `//w[.//pos[@set="S" and @class="A" and not(ancesto`
  - Note: This query assumes the set attribute was set explicitly, i.e. there are multiple possible sets in the document for this annotation type. If there is only one set for this annotation type, it would be the default and only appear in the header's annotation declarations.
- XPath query to select all alternative PoS tags for all words: `//w/alt/pos`
- **XPath query for to obtain sentences except those in quotes:** `//s[not(ancestor::quote]`
  - Explanation: When selecting sentences, you often do not want sub-sentences that are part of a quote, since they may overlap with the larger sentence they form a part of.

When selecting text elements `t`, you generally want to add `not(@class)` to the constraint, to select only the text content elements that have not been assigned an alternative class. Recall that multiple text content may be present, bearing another class. Omitting this constraint will prevent you from properly retrieving the current text of a document, as it will also retrieve all this differently typed text content.

Before you release XPath queries on FoLiA documents, make sure to first parse the declarations present in the *Annotation Declarations*. Verify that the annotation type with the desired set you are looking for is actually present, otherwise you need not bother running a query at all. Note that the XPath expression differs based on whether there is only one set defined for the sought annotation type, or if there are multiple. In the former case, you cannot use the `@set` attribute to select, and in the latter case, you must.

As crafting good XPath queries is not trivial and requires knowledge of FoLiA, higher level abstractions are available in the form *FoLiA Query Language (FQL)*, or the use of dedicated FoLiA libraries.

6.2 FoLiA Query Language (FQL)

Whereas XPath is a very generic query language, the FoLiA Query Language (FQL) is a very specific language, designed purely for FoLiA. It allows advanced querying and document editing.

FQL statements are separated by newlines and encoded in UTF-8. The expressions are case sensitive, all keywords are in upper case, all element names and attributes in lower case.

FQL is also strict about parentheses, they are generally either required or forbidden for an expression. Parentheses usually indicate a sub-expression, and it is also used in boolean logic.

As a general rule, it is more efficient to do a single big query than multiple standalone queries.

Note that for readability, queries may have been split on multiple lines in the presentation here, whereas in reality they should be on one.

6.2.1 Global variables

- `SET variable$=$value` - Sets global variables that apply to all statements that follow. String values need to be in double quotes. Available variables are: * `processor` - The ID of the processor * `annotator` - The name of the annotator (the use of `processor` is preferred!) * `annotatortype` - The type of the annotator, can be *auto* or *manual* (the use of `processor` is preferred!)

Usually your queries on a particular annotation type are limited to one specific set. To prevent having to enter the set explicitly in your queries, you can set defaults. The annotation type corresponds to a FoLiA element:

```
DEFAULTSET entity https://raw.githubusercontent.com/proycon/folia/master/
↪setdefinitions/namedentitycorrection.foliaset.xml
```

If the FoLiA document only has one set of that type anyway, then this is not even necessary and the default will be automatically set.

6.2.2 Provenance

To make use of FoLiA's ability to register provenance_data, you need to define a processor as follows:

```
PROCESSOR id "p0" name "mytool" type "auto"
```

The processor will be created and appended to the provenance chain. If you do not specify an ID, one will be generated for you automatically, unless the **last** processor in the chain matches, then that one will be reused.

If you do specify an ID and it already exists, the existing processor will be selected, any subsequent assignments you make will overwrite the original values.

Processors can be nested using the `IN PROCESSOR` keyword, the matching rules when omitting an ID apply here as well, but for subprocessors all candidates are considered rather than only the last processor:

```
PROCESSOR id "p0.1" name "john doe" type "manual" IN PROCESSOR id "p0" name
↪"annotationtool" type "auto"
```

As you see, the `PROCESSOR` statement allows you to create and select processors. The selected processor will be used for all subsequent queries, meaning that all annotations make will be associated with that processor. You can select any existing processor again using `PROCESSOR id "id"`. To deselect processors entirely, use `PROCESSOR NONE`.

6.2.3 Declarations

All annotation types in FoLiA need to be declared. FQL does this for you automatically. If you make an edit of a previously undeclared set, it will be declared for you. These default declarations will never assign default annotators or annotator types, and if a *PROCESSOR* statement was issued earlier, it will use that processor.

Explicit declarations are possible using the *DECLARE* keyword followed by the annotation type you want to declare, this represented the tag of the respective FoLiA annotation element:

```
DECLARE entity OF "https://github.com/proycon/folia/blob/master/setdefinitions/
↪namedentities.foliaset.xml"
```

The *WITH* clause is optional, the set following the *OF* keyword is mandatory for annotation types where you use a set.

Declarations may be chained, i.e. multiple *DECLARE* statements may be issued on one line, as well as prepended to action statements (see next section).

You can declare a processor and use it in a declaration in one line:

```
PROCESSOR id "p0" name "named-entity-recogniser" version "1.0" DECLARE entity OF
↪"https://github.com/proycon/folia/blob/master/setdefinitions/namedentities.foliaset.
↪xml"
```

Note: If you don't intend to use FoLiA v2's provenance mechanism, i.e. processors, then you can declare default annotators and annotator types the old way:

```
DECLARE entity OF "https://github.com/proycon/folia/blob/master/setdefinitions/
↪namedentities.foliaset.xml"
WITH annotator = "me" annotatortype = "manual"
```

Note that the statement must be on one single line, it is split here only for ease of presentation.

6.2.4 Actions

The core part of an FQL statement consists of an action verb, the following are available:

- `SELECT <focus expression> [<target expression>]` - Selects an annotation
- `DELETE <focus expression> [<target expression>]` - Deletes an annotation
- `EDIT <focus expression> [<assignment expression>] [<target expression>]` - Edits an existing annotation
- `ADD <focus expression> <assignment expression> <target expression>` - Adds an annotation (to the target expression)
- `APPEND <focus expression> <assignment expression> <target expression>` - Inserts an annotation after the target expression
- `PREPEND <focus expression> <assignment expression> <target expression>` - Inserts an annotation before the target expression

Following the action verb is the focus expression, this starts with an annotation type, which is equal to the FoLiA XML element tag. The set is specified using `OF <set>` and/or the ID with `ID <id>`. An example:

```
pos OF "http://some.domain/some.folia.set.xml"
```

If an annotation type is already declared and there is only one in document, or if the *DEFAULTSET* statement was used earlier, then the *OF* statement can be omitted and will be implied and detected automatically. If it is ambiguous, an error will be raised (rather than applying the query regardless of set).

To further filter a the focus, the expression may consist of a *WHERE* clause that filters on one or more FoLiA attributes:

- `class`
- `annotator`
- `annotatortype`
- `n`
- `confidence`
- `src`
- `speaker`
- `begintime`
- `endtime`

The following keywords are also available on when the elements contains text and/or phonetic/phonological content:

- `text`
- `phon`

The *WHERE* statement requires an operator (=,“!=“,“>“,“<, ``<=,“>=“,“CONTAINS“,“MATCHES“), the *AND*, *OR* and *NOT* operators are available (along with parentheses) for grouping and boolean logic. The operators must never be glued to the attribute name or the value, but have spaces left and right.

We can now show some examples of full FQL queries with some operators:

- `SELECT pos OF "http://some.domain/some.folia.set.xml"`
- `SELECT pos WHERE class = "n" AND annotator = "johndoe"`
- `DELETE pos WHERE class = "n" AND annotator != "johndoe"`
- `DELETE pos WHERE class = "n" AND annotator CONTAINS "john"`
- `DELETE pos WHERE class = "n" AND annotator MATCHES "^john$"`

The *ADD* and *EDIT* change actual attributes, this is done in the *assignment expression* that starts with the *WITH* keyword. It applies to all the common FoLiA attributes like the *WHERE* keyword, but has no operator or boolean logic, as it is a pure assignment function.

SELECT and DELETE only support WHERE, EDIT supports both WHERE and WITH, if both are use they than WHERE is always before WITH. the ADD action supports only WITH. If an EDIT is done on an annotation that can not be found, and there is no WHERE clause, then it will fall back to ADD.

Here is an *EDIT* query that changes all nouns in the document to verbs (assuming a particular set):

- `EDIT pos WHERE class = "n" WITH class "v" AND annotator = "johndoe"`

The query is fairly crude as it still lacks a *target expression*: A target expression determines what elements the focus is applied to, rather than to the document as a whole, it starts with the keyword *FOR* and is followed by either an annotation type (i.e. a FoLiA XML element tag) *or* the ID of an element. The target expression also determines what elements will be returned. More on this in a later section.

The following FQL query shows how to get the part of speech tag for a word:

```
SELECT pos FOR ID mydocument.word.3
```

Or for all words:

```
SELECT pos FOR w
```

The *ADD* action almost always requires a target expression:

```
ADD pos WITH class "n" FOR ID mydocument.word.3
```

Multiple targets may be specified, comma delimited:

```
ADD pos WITH class "n" FOR ID mydocument.word.3  , ID myword.document.word.25
```

The target expression can again contain a *WHERE* filter:

```
SELECT pos FOR w WHERE class != "PUNCT"
```

Target expressions, starting with the *FOR* keyword, can be nested:

```
SELECT pos FOR w WHERE class != "PUNCT" FOR event WHERE class = "tweet"
```

You may also use the SELECT keyword without focus expression, but only with a target expression. This is particularly useful when you want to return multiple distinct elements, for instance by ID:

```
SELECT FOR ID mydocument.word.3 , ID myword.document.word.25
```

The *SELECT* keyword can also be used with the special *ALL* selector that selects all elemens in the scope, the following two statement are identical and will return all elements in the document:

```
SELECT ALL
SELECT FOR ALL
```

It can be used at deeper levels too, the following will return everything under all words:

```
SELECT ALL FOR w
```

Target expressions are vital for span annotation, the keyword *SPAN* indicates that the target is a span (to do multiple spans at once, repeat the SPAN keyword again), the operator `&` is used for consecutive spans, whereas `,` is used for disjoint spans:

```
ADD entity WITH class "person" FOR SPAN ID mydocument.word.3 & ID myword.document.
↪word.25
```

This works with filters too, the & operator enforced a single consecutive span:

```
ADD entity WITH class "person" FOR SPAN w WHERE text = "John" & w WHERE text = "Doe"
```

Remember we can do multiple at once:

```
ADD entity WITH class "person" FOR SPAN w WHERE text = "John" & w WHERE text = "Doe"
SPAN w WHERE text = "Jane" & w WHERE text = "Doe"
```

The *HAS* keyword enables you to descend down in the document tree to siblings. Consider the following example that changes the part of speech tag to "verb", for all occurrences of words that have lemma "fly". The parentheses are mandatory for a *HAS* statement:

```
EDIT pos OF "someposset" WITH class = "v" FOR w WHERE (lemma OF "somelemmaset" HAS␣
↪class = "fly")
```

Target expressions can be former with either *FOR* or with *IN*, the difference is that *IN* is much stricter, the element has to be a direct child of the element in the *IN* statement, whereas *FOR* may skip intermediate elements. In analogy with XPath, *FOR* corresponds to // and *IN* corresponds to /. *FOR* and *IN* may be nested and mixed at will. The following query would most likely not yield any results because there are likely to be paragraphs and/or sentences between the wod and event structures:

```
SELECT pos FOR w WHERE class != "PUNCT" IN event WHERE class = "tweet"
```

Multiple actions can be combined, all share the same target expressions:

```
ADD pos WITH class "n" ADD lemma WITH class "house" FOR w WHERE text = "house" OR␣
↪text = "houses"
```

It is also possible to nest actions, use parentheses for this, the nesting occurs after any WHERE and WITH statements:

```
ADD w ID mydoc.sentence.1.word.1 (ADD t WITH text "house" ADD pos WITH class "n") FOR␣
↪ID mydoc.sentence.1
```

Though explicitly specified here, IDs will be automatically generated when necessary and not specified.

The *ADD* action has two cousins: *APPEND* and *PREPEND*. Instead of adding something in the scope of the target expression, they either append or prepend an element, so the inserted element will be a sibling:

```
APPEND w (ADD t WITH text "house") FOR w WHERE text = "the"
```

This above query appends/inserts the word "house" after every definite article.

6.2.5 Text

Our previous examples mostly focussed on part-of-speech annotation. In this section we look at text content, which in FoLiA is an annotation element too (t).

Here we change the text of a word:

```
EDIT t WITH text = "house" FOR ID mydoc.word.45
```

Here we edit or add (recall that EDIT falls back to ADD when not found and there is no further selector) a lemma and check on text content:

```
EDIT lemma WITH class "house" FOR w WHERE text = "house" OR text = "houses"
```

You can use WHERE text on all elements, it will cover both explicit text content as well as implicit text content, i.e. inferred from child elements. If you want to be really explicit you can do:

```
EDIT lemma WITH class "house" FOR w WHERE (t HAS text = "house")
```

Advanced:

Such syntax is required when covering texts with custom classes, such as OCRed or otherwise pre-normalised text. Consider the following OCR correction:

```
ADD t WITH text = "spell" FOR w WHERE (t HAS text = "5pe11" AND class = "OCR" )
```

6.2.6 Query Response

We have shown how to do queries but not yet said anything on how the response is returned. This is regulated using the *RETURN* keyword:

- `RETURN focus` (default)
- `RETURN parent` - Returns the parent of the focus
- `RETURN target` or `RETURN inner-target`
- `RETURN outer-target`
- `RETURN ancestor-target`
- `RETURN alternative`

The default focus mode just returns the focus. Sometimes, however, you may want more context and may want to return the target expression instead. In the following example returning only the pos-tag would not be so interesting, you are most likely interested in the word to which it applies:

```
SELECT pos WHERE class = "n" FOR w RETURN target
```

When there are nested FOR/IN loops, you can specify whether you want to return the inner one (highest granularity, default) or the outer one (widest scope). You can also decide to return the first common structural ancestor of the (outer) targets, which may be specially useful in combination with the *SPAN} keyword.

The return type can be set using the *FORMAT* statement:

- `FORMAT xml` - Returns FoLiA XML, the response is contained in a simple `<results><result/></results>` structure.
- `FORMAT single-xml` - Like above, but returns pure unwrapped FoLiA XML and therefore only works if the response only contains one element. An error will be raised otherwise.
- `FORMAT json` - Returns JSON list
- `FORMAT single-json` - Like above, but returns a single element rather than a list. An error will be raised if the response contains multiple.
- `FORMAT python` - Returns a Python object, can only be used when directly querying the FQL library without the document server
- `FORMAT flat` - Returns a parsed format optimised for FLAT. This is a JSON reply containing an HTML skeleton of structure elements (key html), parsed annotations (key annotations). If the query returns a full FoLiA document, then the JSON object will include parsed set definitions, (key setdefinitions), and declarations.

The *RETURN* statement may be used standalone or appended to a query, in which case it applies to all subsequent queries. The same applies to the *FORMAT* statement, though an error will be raised if distinct formats are requested in the same HTTP request.

When context is returned in *target* mode, this can get quite big, you may constrain the type of elements returned by using the *REQUEST* keyword, it takes the names of FoLiA XML elements. It can be used standalone so it applies to all subsequent queries:

```
REQUEST w,t,pos,lemma
```

..or after a query:

```
SELECT pos FOR w WHERE class!="PUNCT" FOR event WHERE class="tweet" REQUEST w,pos,
↪lemma
```

Two special uses of request are `REQUEST ALL` (default) and `REQUEST NOTHING`, the latter may be useful in combination with *ADD*, *EDIT* and *DELETE*, by default it will return the updated state of the document.

Note that if you set REQUEST wrong you may quickly end up with empty results.

6.2.7 Span Annotation

Selecting span annotations is identical to token annotation. You may be aware that in FoLiA span annotation elements are technically stored in a separate stand-off layers, but you can forget this fact when composing FQL queries and can access them right from the elements they apply to.

The following query selects all named entities (of an actual rather than a fictitious set for a change) of people that have the name John:

```
SELECT entity OF "https://github.com/proycon/folia/blob/master/setdefinitions/
↪namedentities.foliaset.xml"
WHERE class = "person" FOR w WHERE text = "John"
```

Or consider the selection of noun-phrase syntactic units (su) that contain the word house:

```
SELECT su WHERE class = "np" FOR w WHERE text CONTAINS "house"
```

Note that if the *SPAN} keyword were used here, the selection would be exclusively constrained to single words "John":

```
SELECT entity WHERE class = "person" FOR SPAN w WHERE text = "John"
```

We can use that construct to select all people named John Doe for instance:

```
SELECT entity WHERE class = "person" FOR SPAN w WHERE text = "John" & w WHERE text =
↪"Doe"
```

Span annotations like syntactic units are typically nested trees, a tree query such as "//pp/np/adj" can be represented as follows. Recall that the *IN* statement starts a target expression like *FOR*, but is stricter on the hierarchy, which is what we would want here:

```
SELECT su WHERE class = "adj" IN su WHERE class = "np" IN su WHERE class = "pp"
```

In such instances we may be most interested in obtaining the full PP:

```
SELECT su WHERE class = "adj" IN su WHERE class = "np" IN su WHERE class = "pp"␣
↪RETURN outer-target
```

The *EDIT* action is not limited to editing attributes, sometimes you want to alter the element of a span. A separate *RESPAN* keyword (without FOR/IN/WITH) accomplishes this. It takes the keyword *RESPAN* which behaves the same as a *FOR SPAN* target expression and represents the new scope of the span, the normal target expression represents the old scope:

```
EDIT entity WHERE class= "person" RESPAN ID word.1 & ID word.2 FOR SPAN ID word.1 &␣
↪ID word.2 & ID word.3
```

WITH statements can be used still too, they always preceed *RESPAN*:

```
EDIT entity WHERE class= "person" WITH class="location" RESPAN ID word.1 & ID word.2␣
↪FOR SPAN ID word.1 & ID word.2 & ID word.3
```

6.2.8 Corrections and Alternatives

Both FoLiA and FQL have explicit support for corrections and alternatives on annotations. A correction is not a blunt substitute of an annotation of any type, but the original is preserved as well. Similarly, an alternative annotation is one that exists alongside the actual annotation of the same type and set, and is not authoritative.

The following example is a correction but not in the FoLiA sense, it bluntly changes part-of-speech annotation of all occurrences of the word *fly* from *n* to *v*, for example to correct erroneous tagger output:

```
EDIT pos WITH class "v" WHERE class = "n" FOR w WHERE text = "fly"
```

Now we do the same but as an explicit correction:

```
EDIT pos WITH class "v" WHERE class = "n" (AS CORRECTION OF "some/correctionset" WITH␣
↪class "wrongpos")
FOR w WHERE text = "fly"
```

Another example in a spelling correction context, we correct the misspelling *concous* to *conscious*:

```
EDIT t WITH text "conscious" (AS CORRECTION OF "some/correctionset" WITH class
↪"spellingerror")
FOR w WHERE text = "concous"
```

The *AS CORRECTION* keyword (always in a separate block within parentheses) is used to initiate a correction. The correction is itself part of a set with a class that indicates the type of correction.

Alternatives are simpler, but follow the same principle:

```
EDIT pos WITH class "v" WHERE class = "n" (AS ALTERNATIVE) FOR w WHERE text = "fly"
```

Confidence scores are often associationed with alternatives:

```
EDIT pos WITH class "v" WHERE class = "n" (AS ALTERNATIVE WITH confidence 0.6)
FOR w WHERE text = "fly"
```

The *AS* clause is also used to select alternatives rather than the authoritative form, this will get all alternative pos tags for words with the text "fly":

```
SELECT pos (AS ALTERNATIVE) FOR w WHERE text = "fly"
```

If you want the authoritative tag as well, you can chain the actions. The same target expression (FOR..) always applies to all chained actions, but the AS clause applies only to the action in the scope of which it appears:

```
SELECT pos SELECT pos (AS ALTERNATIVE) FOR w WHERE text = "fly"
```

Filters on the alternative themselves may be applied as expected using the WHERE clause:

```
SELECT pos (AS ALTERNATIVE WHERE confidence > 0.6) FOR w WHERE text = "fly"
```

Note that filtering on the attributes of the annotation itself is outside of the scope of the AS clause:

```
SELECT pos WHERE class = "n" (AS ALTERNATIVE WHERE confidence > 0.6) FOR w WHERE text␣
↪= "fly"
```

When you use *AS ALTERNATIVE**, you can combine with with *RETURN alternative* to return the entire alternative block in which the alternatives reside, rather than the alternative annotations themselves:

```
SELECT pos (AS ALTERNATIVE) FOR w WHERE text = "fly" RETURN alternative
```

Corrections by definition are authoritative, so no special syntax is needed to obtain them. Assuming the part of speech tag is corrected, this will correctly obtain it, no AS clause is necessary:

```
SELECT pos FOR w WHERE text = "fly"
```

Adding *AS CORRECTION* will only enforce to return those that were actually corrected:

```
SELECT pos (AS CORRECTION) FOR w WHERE text = "fly"
```

However, if you want to obtain the original prior to correction, you can do so using *AS CORRECTION ORIGINAL*:

```
SELECT pos (AS CORRECTION ORIGINAL) FOR w WHERE text = "fly"
```

FoLiA does not just distinguish corrections, but also supports suggestions for correction. Envision a spelling checker suggesting output for misspelled words, but leaving it up to the user which of the suggestions to accept. Suggestions are not authoritative and can be obtained in a similar fashion by using the *SUGGESTION* keyword:

```
SELECT pos (AS CORRECTION SUGGESTION) FOR w WHERE text = "fly"
```

Note that *AS CORRECTION* may take the *OF* keyword to specify the correction set, they may also take a *WHERE* clause to filter:

```
SELECT t (AS CORRECTION OF "some/correctionset" WHERE class = "confusible") FOR w
```

The *SUGGESTION* keyword can take a WHERE filter too:

```
SELECT t (AS CORRECTION OF "some/correctionset" WHERE class = "confusible" SUGGESTION␣
↪WHERE confidence > 0.5) FOR w
```

To add a suggestion for correction rather than an actual authoritative correction, you can do:

```
EDIT pos (AS CORRECTION OF "some/correctionset" WITH class "poscorrection" SUGGESTION␣
↪class "n") FOR w ID some.word.1
```

The absence of a WITH statement in the action clause indicates that this is purely a suggestion. The actual suggestion follows the *SUGGESTION* keyword.

Any attributes associated with the suggestion can be set with a *WITH* statement after the suggestion:

```
EDIT pos (AS CORRECTION OF "some/correctionset" WITH class "poscorrection" SUGGESTION␣
↪class "n" WITH confidence 0.8) FOR w ID some.word.1
```

Even if a *WITH* statement is present for the action, making it an actual correction, you can still add suggestions:

```
EDIT pos WITH class "v" (AS CORRECTION OF "some/correctionset" WITH class
↪"poscorrection" SUGGESTION class "n" WITH confidence 0.8) FOR w ID some.word.1
```

The *SUGGESTION* keyword can be chaineed to add multiple suggestions at once:

```
EDIT pos (AS CORRECTION OF "some/correctionset" WITH class "poscorrection"
SUGGESTION class "n" WITH confidence 0.8
SUGGESTION class "v" wITH confidence 0.2) FOR w ID some.word.1
```

Another example in a spelling correction context:

```
EDIT t (AS CORRECTION OF "some/correctionset" WITH class "spellingerror"
SUGGESTION text "conscious" WITH confidence 0.8 SUGGESTION text "couscous" WITH␣
↪confidence 0.2)
FOR w WHERE text = "concous"
```

When a correction is made on an element, all annotations below it (recursively) are left intact, i.e. they are copied from the original element to the new correct element. The same applies to suggestions. Moreover, all references to the original element, from for instance span annotation elements, will be made into references to the new corrected elements.

This is not always what you want, if you want the correction not to have any annotations inherited from the original, simply use *AS BARE CORRECTION* instead of *AS CORRECTION*.

You can also use *AS CORRECTION* with *ADD* and *DELETE*.

The most complex kind of corrections are splits and merges. A split separates a structure element such as a word into multiple, a merge unifies multiple structure elements into one.

In FQL, this is achieved through substitution, using the action *SUBSTITUTE*:

```
SUBSTITUTE w WITH text "together" FOR SPAN w WHERE text="to" & w WHERE text="gether"
```

Sub-queries are common with SUBSTITUTE, the following is equivalent to the above:

```
SUBSTITUTE w (ADD t WITH text "together") FOR SPAN w WHERE text="to" & w WHERE text=
↪"gether"
```

To perform a split into multiple substitutes, simply chain the SUBSTITUTE clause:

```
SUBSTITUTE w WITH text "each" SUBSTITUTE w WITH TEXT "other" FOR w WHERE text=
↪"eachother"
```

Like *ADD*, both *SUBSTITUTE* may take assignments (*WITH*), but no filters (*WHERE*).

You may have noticed that the merge and split examples were not corrections in the FoLiA-sense; the originals are removed and not preserved. Let's make it into proper corrections:

```
SUBSTITUTE w WITH text "together"
(AS CORRECTION OF "some/correctionset" WITH class "spliterror")
FOR SPAN w WHERE text="to" & w WHERE text="gether"
```

And a split:

```
SUBSTITUTE w WITH text "each" SUBSTITUTE w WITH text "other"
(AS CORRECTION OF "some/correctionset WITH class "runonerror")
FOR w WHERE text="eachother"
```

To make this into a suggestion for correction instead, use the *SUGGESTION} folloed by *SUBSTITUTE*, inside the *AS* clause, where the chain of substitute statements has to be enclosed in parentheses:

```
SUBSTITUTE (AS CORRECTION OF "some/correctionset" WITH class "runonerror" SUGGESTION␣
↪(SUBTITUTE w WITH text "each" SUBSTITUTE w WITH text "other") )
FOR w WHERE text="eachother"
```

6.2.9 Dealing with context

We have seen that with the *FOR* keyword we can move to bigger elements in the FoLiA document, and with the *HAS* keyword we can move to siblings. There are several *context keywords* that give us all the tools we need to peek at the context. Like *HAS* expressions, these need always be enclosed in parentheses.

For instance, consider a part-of-speech tagging scenario. If we have a word where the left neighbour is a determiner, and the right neighbour a noun, we can be pretty sure the word under our consideration (our target expression) is an adjective. Let's add the pos tag:

```
EDIT pos WITH class = "adj" FOR w WHERE (PREVIOUS w WHERE (pos HAS class == "det"))␣
↪AND (NEXT w WHERE (pos HAS class == "n"))
```

You may append a number directly to the *PREVIOUS/NEXT* modifier if you're interested in further context, or you may use *LEFTCONTEXT/RIGHTCONTEXT/CONTEXT* if you don't care at what position something occurs:

```
EDIT pos WITH class = "adj" FOR w WHERE (PREVIOUS2 w WHERE (pos HAS class == "det"))␣
↪AND (PREVIOUS w WHERE (pos HAS class == "adj")) AND (RIGHTCONTEXT w WHERE (pos HAS␣
↪class == "n"))
```

Instead of the *NEXT* and *PREVIOUS* keywords, a target expression can be used with the *SPAN* keyword and the *&* operator:

```
SELECT FOR SPAN w WHERE text = "the" & w WHERE (pos HAS class == "adj") & w WHERE␣
↪text = "house"
```

Within a *SPAN* keyword, an *expansion expression* can be used to select any number, or a certain number, of elements. You can do this by appending curly braces after the element name (but not attached to it) and specifying the minimum and maximum number of elements. The following expression selects from zero up to three adjectives between the words *the* and *house*:

```
SELECT FOR SPAN w WHERE text = "the" & w {0,3} WHERE (pos HAS class == "adj") & w␣
↪WHERE text = "house"
```

If you specify only a single number in the curly braces, it will require that exact number of elements. To match at least one word up to an unlimited number, use an expansion expression such as `{1,}`

If you are now perhaps tempted to use the FoLiA document server and FQL for searching through large corpora in real-time, then be advised that this is not a good idea. It will be prohibitively slow on large datasets as this requires smart indexing, which this document server does not provide. You can therefore not do this real-time, but perhaps only as a first step to build an actual search index.

Other modifiers are PARENT and and ANCESTOR. PARENT will at most go one element up, whereas ANCESTOR will go on to the largest element:

```
SELECT lemma FOR w WHERE (PARENT s WHERE  text CONTAINS "wine")
```

Instead of *PARENT*, the use of a nested *FOR* is preferred and more efficient:

```
SELECT lemma FOR w FOR s WHERE text CONTAINS "wine"
```

Let's revisit syntax trees for a bit now we know how to obtain context. Imagine we want an NP to the left of a PP:

```
SELECT su WHERE class = "np" AND (NEXT su WHERE class = "pp")
```

... and where the whole thing is part of a VP:

```
SELECT su WHERE class = "np" AND (NEXT su WHERE class = "pp") IN su WHERE class = "vp"
```

... and return that whole tree rather than just the NP we were looking for:

```
SELECT su WHERE class = "np" AND (NEXT su WHERE class = "pp") IN su WHERE class = "vp
↪" RETURN target
```

6.2.10 Relations

FoLiA's *Relation Annotation* is a higher-order annotation type that allows linking between arbitrary annotations, and even between documents or to external resources.

Internal Relations

In FQL, there is a special construction using a sub-query to actively select the items you want to link to (within the same document), the sub-query start with the TO keyword, which behaves identical to SELECT:

```
ADD relation WITH class "wh-movement" (TO su ID "su.moved") FOR su ID "su.1"
```

Multiple targets may be chained chained by simply adding extra sub-queries (each in its own set of parentheses).

External Relations

For external relations, you usually don't need the individual references and the TO statement won't work. For those types you just set the href attribute, possibly on the relation, combined also with the format attribute. Here we add a external relationship to the relevant Wikipedia page on a named entity about the Dalai Lama:

```
ADD relation WITH class "wikipedia" href "https://en.wikipedia.org/wiki/Dalai_Lama"␣
↪format "text/html" FOR entity ID "example.p.1.s.1.entity.1"
```

6.2.11 Shortcuts

Classes are prevalent all throughout FoLiA, it is very common to want to select on classes. To select words with pos tag n for example you can do:

```
SELECT w WHERE (pos HAS class = "n")
```

Because this is so common, there is a shortcut. Specify the annotation type directly preceeded by a colon, and a HAS statement that matches on class will automatically be constructed:

```
SELECT w WHERE :pos = "n"
```

The two statements are completely equivalent.

Another third alternative to obtain the same result set is to use a target expression:

```
SELECT pos WHERE class = "n" FOR w RETURN target
```

This illustrates that there are often multiple ways of obtaining the same result set. Due to lazy evaluation in the FQL library, there is not much difference performance-wise.

Another kind of shortcut exists for setting text on structural elements. The explicit procedure to add a word goes as follows:

```
ADD w (ADD t WITH text "hello") IN ID some.sentence
```

The shortcut is:

```
ADD w WITH text "hello" IN ID some.sentence
```

CHAPTER 7

Form

In addition to the normal form of FoLiA XML, there is an additional *explicit* form. This form of XML serialisation is functionally equivalent to the normal form, but any defaults that are implicit in the normal form are expressed explicitly instead. Documents in either form can always be converted to eachother without any gain or loss of information, it is just the accessibility of the certain information that is facilitated in explicit mode, at the cost of redundancy, bigger filesize and higher memory footprint.

The reason for the existance of this explicit form is to help parsers, especially those not implementing the full FoLiA logic. Parsers that can not deal with a document in normal form should themselves invoke `foliavalidator --explicit` to do the conversion to explicit form prior to parsing it themselves.

The explicit form is declared by the attribute `form="explicit"` on the FoLiA root tag. When this form attribute is not set to explicit (or absent) altogether, behaviour is unchanged and normal form is used.

In explicit form, all defaults are made explicit:

- All annotations that carry a set have a set attribute, sets never refer to aliases.
- All annotations associated with a processor have an explicit processor attribute.
- Layers themsleves carry a set attribute if the span elements within carry a set.
- All text-content elements explicitly declare their class (so `<t>` will become `<t class="current">`)
- Predefined features/subsets are serialised explicitly using `<feat>` elements rather than as XML attributes.

Certain FoLiA internals are made explicit:

- All annotation elements get a `typegroup` attribute that makes explicit what kind of annotation category we are dealing with. Values are: *structure, inline, span, higherorder, textmarkup, content, layer*. So `<w>` becomes `<w typegroup="structure">`, `<pos>` becomes `<pos typegroup="inline">`. This allows for example xpath expressions like: give me the deepest structural ancestor.

CHAPTER 8

Implementations

8.1 Libraries

Currently, the following FoLiA library implementations exist. Both follow a highly object-oriented model in which FoLiA XML elements correspond with classes.

- FoLiApy - A FoLiA library in Python.
- Library documentation and API reference
- libfolia - A FoLiA library in C++.

Both libraries are shipped as part of our LaMachine software distribution.

Information regarding implementation of certain elements for these two libraries is present in the status boxes throughout this documentation. The following table shows the level of FoLiA support in these libraries:

The following table lists FoLiA library implementations, the last column lists the predecessor of FoLiApy, which was part of PyNLPl.

Table 1: FoLiA Library Implementations

Category	Name	FoLiApy	libfolia	folia-rust
General	Language	Python 3.5+	C++	Rust
General	Uses official specification	yes	yes	yes
General	Maintenance Status	active	active	active
General	Maturity	stable	stable	alpha
General	Lead developer	Maarten van Gompel	Ko van der Sloot	Maarten
General	Supports latest version	2.3	2.3	2.3
General	Supports earliest version	0.1	0.1	2.0
Implementation	Full in-memory representation?	default	default	default
Implementation	Streaming parser?	optional	optional	not yet
Implementation	Element representation	class hierarchy (OOP)	class hierarchy (OOP)	other
Implementation	Memory-saving encoding	no	no	yes

Table 1 – continued from previous page

Category	Name	FoLiApy	libfolia	folia-rust
Supported Features	Structure annotation	yes	yes	yes
Supported Features	Inline annotation	yes	yes	yes
Supported Features	Span annotation	yes	yes	yes
Supported Features	Text mark-up	yes	yes	yes
Supported Features	Subtoken annotation (e.g. morphology)	yes	yes	yes
Supported Features	Alternatives	yes	yes	limited
Supported Features	Corrections	yes	yes	limited
Supported Features	Substrings	yes	yes	yes
Supported Features	Relations	yes	yes	limited
Supported Features	Native Metadata	yes	yes	yes
Supported Features	Submetadata	yes	yes	no
Supported Features	Foreign Metadata	yes	yes	no
Supported Features	Provenance data	yes	yes	yes
Supported Features	Foreign Annotations	yes	yes	no
Supported Features	Features	yes	yes	yes
Validation	Shallow Validation	yes	yes	no
Validation	Text Consistency Validation	yes	yes	no
Validation	Deep Validation	yes	no	no
Validation	… with RDF+XML sets	yes	no	no
Validation	… with RDF+turtle sets	yes	no	no
Validation	… with legacy XML sets	yes	no	no
Validation	RelaxNG schema generation	yes	no	no
Serialisation	XML (normal form)	yes	yes	yes
Serialisation	JSON (not standarised)	yes	no	no
Serialisation	RDF	no	no	no
Querying	select() mechanism	yes	yes	yes
Querying	FoLiA Query Language (FQL)	yes	no	no
Querying	… CQL support	through pynlpl	no	no
Querying	Document findwords() method	yes	yes	no
Querying	Introspection into FoLiA specification	yes	yes	yes
Quality Control	Integration tests	yes	yes	yes (limit
Documentation	API reference	yes	no	yes
Documentation	Tutorial	yes	no	no
Legacy	D-Coi read compatibility	no	no	no
Parsing Details	Predefined subsets as attributes	yes	yes	yes
Parsing Details	Compression?	gz+bz2	gz+bz2	none
Parsing Details	Default set	yes	yes	yes
Parsing Details	Set aliases	yes	yes	?
Parsing Details	Default processor	yes	yes	yes
Parsing Details	Default annotator (old-style)	yes	yes	no
Serialisation Details	Predefined subsets as attributes	yes	yes	no
Serialisation Details	Default set	yes	yes	yes
Serialisation Details	Set aliases	yes	yes	?
Serialisation Details	Default processor	yes	yes	yes
Serialisation Details	Default annotator (old-style)	yes	yes	no

Table 1 – continued from previous page

Category	Name	FoLiApy	libfolia	folia-rust
Serialisation Details	Write explicit form	yes	no	no

8.2 Tools

The following tool collections are available:

- FoLiA Tools - A set of Python-based command-line tools for FoLiA processing. Contains a validator, convertors, and more.
- Tool overview and documentation
- FoLiAutils - A set of command-line utilities for working with FoLiA, powered by libfolia.

CHAPTER 9

Guidelines

This section collects guidelines, tips, do's and don'ts and conventions in dealing with FoLiA documents.

9.1 For data creators/publishers

1. **Always validate all FoLiA documents you create and intend to publish!**. Use one of the official validation tools (`foliavalidator` and `folialint`). See *Validation*. This will already catch most of the issues that could arise out of not following these guidelines.
2. Never invent custom XML elements and attributes. If you really must, make sure they are in a different XML namespace. See *Foreign Annotation*.
3. If you want to encode something and FoLiA does not seem to offer a good solution yet, or if you are simply unsure whether the solution you want to use is appropriate, contact the FoLiA developers on our Issue tracker. FoLiA can be extended in collaboration. Do not simply add your own elements/attributes.
4. Mind the sets you use. Creating and publishing set definitions is recommended but not strictly mandatory for most uses. See *Set Definitions (Vocabulary)*
5. Identifiers should never change: Once you assign an identifier to something and publish your data: do not change any identifier that is in use.
6. All annotation types you use must be declared, see *Annotation Declarations*. Take care not to declare annotation types that you don't actually use in your document unless you have good reason to believe the annotation type will be added soon.

9.2 For developers

1. Using a high-level FoLiA programming library, if available for your programming language, is strongly recommended over parsing/writing/querying the XML yourself, as it will make things a lot easier and save a lot of work!
2. Always use the latest version of FoLiA and its libraries.
3. Mind the sets you use. Actively check whether the sets uses in a document are in fact the ones your software handles, i.e. check the declarations (see *Annotation Declarations*). For example, do not blindly assume any part-of-speech tag will be in your intended vocabulary. See *Set Definitions (Vocabulary)*

4. Considering that FoLiA is vast, it is fine to only support a subset of a certain annotation types in your software, or not to support certain complexities such as *Correction Annotation*. Just make sure to check the declarations based on which you can reject processing a document.

5. The structure of a text as represented in FoLiA documents can differ greatly between documents, as different types of documents (books,articles,papers,poetry,etc..) are structured differently. The annotation declaration in the metadata tell you what structural types you can encounter, but they don't convey precisely how these structures are nested. Unless you have very good reason to do so, do NOT assume your documents are neatly subdivided into e.g. only paragraphs and sentences. There may be lists, figures, divisions. Generally spoken, you'll often want to descend into the deepest structural nodes that have text. The FoLiA libraries provide a high-level API for you to do this.

6. If you don't use a FoLiA library, you may want to consider accepting only FoLiA documents in so-called *explicit form* (see *Form*). Explicit form does not use any implicit defaults but makes everything explicit in the XML. This means the logic in your parser can be kept less complicated. You can turn any explicit form document into a normal form one and vice versa (without loss). If you get a normal form document (which is the norm), run an external tool like `foliavalidator --explicit` to turn it into explicit form before parsing it. It's strongly recommended not to shift this burden to the user as he/she may be confused by it.

9.3 Conventions

Conventions are good practices that you will encounter and are encouraged to follow, but they remain just conventions rather than strict guidelines.

1. Most FoLiA software assigns verbose identifiers for all elements. We use the the ID of the FoLiA document as the base identifier and then append the element type and sequence number, all delimited by dots. The IDs are cumulative in nature, so we get for instance `example.p.1.s.2.w.3` for the third word in the second sentence in the first paragraph of the document with ID `example`. See *Identifiers*

2. Adding metadata to your document is always encouraged.

- genindex
- search

Bibliography

[vanGompel2014] Maarten van Gompel & Martin Reynaert (2014). FoLiA: A practical XML format for linguistic annotation - a descriptive and comparative study; Computational Linguistics in the Netherlands Journal; 3:63-81; 2013. (PDF) (BibTeX)

[Burnard2007] Lou Burnard & Syd Bauman (2007). TEI P5: Guidelines for Electronic Text Encoding and Interchange. (HTML)

[Ide2004] Nancy Ide & Laurent Romary (2005). International standard for a linguistic annotation framework. In: Natural Language Engineering. Volume 10. pp 211-225.

[Heid2010] Ulrich Heid and Helmut Schmid and Kerstin Eckart and Erhard Hinrichs (2010). A Corpus Representation Format for Linguistic Web Services: The D-SPIN Text Corpus Format and its Relationship with ISO Standards. In: Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10).

[Fokkens2014] Antske Fokkens and Aitor Soroa and Zuhaitz Beloki and German Rigan and Willem Robert van Hage and Piek Vossen. NAF: The NLP Annotation Format. (PDF)

[RDF] Richard Cyganiak, David Wood and Markus Lanthaler (2014). RDF 1.1 Concepts and Abstract Syntax (website)

[SKOS] Alistair Miles & Sean Bechhofer (2009). SKOS: Simple Knowledge Organization System Reference (website)