ANNIS3 – Multiple Segmentation Corpora Guide

(For the latest documentation see also:
http://korpling.github.io/ANNIS)

title:	ANNIS3 – Multiple Segmentation Corpora Guide
version:	2013-6-15a

author:	Amir Zeldes
organization:	SFB 632 Information Structure / D1 Linguistic Database Humboldt-Universität zu Berlin & Universität Potsdam
e-mail:	annis-admin@ling.uni-potsdam.de
homepage:	http://www.sfb632.uni-potsdam.de/annis/

Table of Contents

ANNIS3 – Multiple Segmentation Corpora Guide

Amir Zeldes / SFB632 D1

Preamble

This guide gives a quick primer on creating and using corpora in ANNIS3, the series 3 family of ANNIS versions, with:

- Subtokenization (units smaller than the basic search unit, e.g. smaller than words)
- Multiple/conflicting token layers (e.g. diplomatic and normalized transcription of a manuscript)
- Overlapping dialogue data (speaker units/words that begin simultaneously/in the middle of another speakers' units/words)

This guide assumes familiarity with normal corpora in ANNIS (i.e. not containing multiple segmentations). For general topics regarding ANNIS, the query language AQL, conversion of corpora etc. please refer to the most recent ANNIS User Guide (available from the download section at http://www.sfb632.uni-potsdam.de/annis/). ANNIS3 is currently being developed further, so that some things that are not possible yet are being planned, and something that should be possible may not work correctly. We appreciate your bug reports, feature requests and questions – please use the contact information in "Section 4: Further information" below.

1. Introduction

Tokens are by definition the smallest units used in the analysis of corpus data. Traditionally tokens coincide with word forms (or punctuation marks, which usually form their own tokens). This guide concerns situations in which annotations need to apply to segments that are smaller than or do not coincide with the borders of the standard unit of reference, in most cases a word. Specifically, the following scenarios will be dealt with:

- **Subtokenization:** Parts of words should be annotated or there are annotations that encompass only parts of words (e.g. in historical corpora, annotation of a letter as an illuminated initial capital, or line annotations where the beginning of an annotation is on one line and the end is on another.
- **Multiple tokenizations:** There is more than one sensible division of the text into basic units that needs to be taken into account. This can happen because data from two annotators or tools differs (e.g. a tagger tags "New" and "York" separately, but a parser treats "New York" as one token), or because there are two levels of analysis (e.g. diplomatic and normalized transcriptions of a manuscript).
- **Overlapping dialogue data:** two or more speakers perform utterances simultaneously, but utterances from all speakers should be treated equally as words in the base text of the conversation. In conversational data, speakers beginning words in the middle of other speakers' words is also commonplace.

The scenarios above create several difficulties for a corpus search system:

- **Search for "basic text":** users seeking to find all occurrences of a certain word do not know how many speakers there are or how annotation borders have caused words to be split up. It must be possible to search for text on either 'all textual levels' or on a specific one (e.g. words from any speaker, or just a specific speaker).
- **Defining "context":** search results in ANNIS are often given with the surrounding context of plus/minus *n* tokens (e.g. 5 tokens to the left and right). If the smallest units are potentially less than a word, context will behave erratically, unpredictably showing parts of words, and more or less context depending on whether the search result happens to include subtokenized positions.
- **Defining "adjacency":** if there are multiple layers of 'basic text' (unannotated, but tokenized raw data), it is difficult to decide when two elements are adjacent. Two consecutive elements on a tokenization layer may be interrupted by elements on another tokenization layer. It is therefore necessary to have separate search facilities for 'absolute' adjacency (nothing in between) and 'relative' adjacency, in terms of some particular segmentation layer.
- **Correct visualization:** many visualizations assume one unambiguous token layer as part of their display. Facilities are required to define the tokenization used and switch between available tokenizations.

The following sections deal with these issues in ANNIS3. Section 2 deals with the definition of segmentation layers in a corpus. Section 3 describes time alignment in ANNIS3. Section 4 describes search and visualization facilities for defining context and adjacency in the ANNIS3 interface. Section 5 gives some further information and contact information for testing and improving ANNIS3.

2. Defining segmentation layers

Any annotation layer not containing internal overlap can serve as a segmentation layer for ANNIS3. Self-overlapping segmentations are not well defined: it is possible to have a layer called "parser_tok" defining the segmentation "New York" and a layer "tagger_tok" defining the segmentation "New" "York", but not all three conflicting units within one and the same segmentation layer. It is also possible for the "tagger_tok" layer to then be less fine-grained than the "parser_tok" layer at another point. Table 1 illustrates these two segmentations and an underlying token layer "tok" which is most granular, i.e. it has as many units as necessary to include all segment borders in the other segmentations.

parser_tok:	New York		would	’ve
tagger_tok:	New	York	would’ve	
tok:	New	York	would	’ve

Table 1. Two segmentations for the text “New York would’ve” and a most granular token layer below.

Not all layers of a corpus need to be segmentation layers: if “New York” has a part-of-speech annotation attached to it, it will behave correctly by virtue of the underlying segmentation defining “New York” as a segment.

In order for segmentations to behave correctly in terms of adjacency and context definition (see next section), segmentation layers must be enriched with a special relationship between consecutive elements of the segmentation: the **ordering relation**. This relation can be added to any non-self-overlapping layer of spans during the conversion process to relANNIS by using the Pepper manipulator module called `OrderRelationAdder` (for general details about converting data into the relANNIS format see the SaltNPepper documentation at http://korpling.german.hu-berlin.de/saltnpepper). Manipulator modules are activated between the import phase and export phase of the conversion job. Additionally, the output format must be specified as the latest version of the relANNIS format, at present 3.2. The resulting job file for Pepper can look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<pepperParams:PepperParams xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:pepperParams="de.hu_berlin.german.korpling.saltnpepper.pepper.pepperParams">
  <pepperJobParams id="1">
    <importerParams formatName="PAULA" formatVersion="1.0" sourcePath="./subtok.demo/"/>
    <moduleParams moduleName="OrderRelationAdder" specialParams="addorder.prop"/>
    <exporterParams formatName="relANNIS" formatVersion="3.2"
          destinationPath="./subtok.demo_r/"/>
  </pepperJobParams>
</pepperParams:PepperParams>
```

The `importerParams` element specifies the source format and version (PAULA, 1.0), as well the source path. The export module specifies relANNIS, version 3.2 and the destination path. Note that ANNIS3 is backwards compatible with previous versions of relANNIS, but multiple segmentations are only supported as of version 3.2. It is also possible to supply absolute paths, for example on a Windows file system `sourcePath="file:/C:/Pepper/corpora/subtok.demo/"`.

Between the importer and exporter, any number of modules can manipulate the data in Salt (for details see the SaltNPepper documentation). The `OrderRelationAdder` module requires a configuration file referred to in the argument `specialParams`. That file, here named `addorder.prop`, must contain a list of the layers for which the ordering relation will be added. The file can look like this for the corpus subtok.demo, which has two tokenization layers: “norm” and “diplomatic”.

```
segmentation-layers={norm, diplomatic}
```

Once the conversion has been carried out, the relANNIS output data will treat these layers as alternative token layers, as described in the following section.

Note that having alternative tokenization layers does not remove the need for a most granular layer, the level of tokens in the strictest sense, marked as "tok" in Table 1 above. In many source formats, this level will have to be present in the data (e.g. PAULA, TreeTagger; see example corpora for ANNIS3 on the ANNIS web site). Some formats, such as EXMaRALDA, have a built in timeline, which is a necessarily most granular layer. If you are using a timeline format, you have the option of generating the most granular token layer automatically from the timeline (if you are using EXMaRALDA XML you will probably want to do this). To do so, use the Pepper module Timeline2Token, which also uses a configuration file for the following parameter:

```
number-artificial-token=true
```

This parameter determines whether the underlying token layer, which is produced based on the time line, will be empty (each underlying token contains the empty string) or filled with a sequence of numbers. By default, the parameter is set to false, that is the artificial tokens are simply left empty.

The module Timeline2Token should apply between import and adding the ordering relation in the Pepper job description (.pepperparams):

```
...
<importerParams moduleName="EXMARaLDAImporter" sourcePath="./mycorpus_exb/"
    specialParams="exmaralda.prop"/>
<moduleParams moduleName="Timeline2Token" specialParams="timeline.prop"/>
<moduleParams moduleName="OrderRelationAdder" specialParams="addorder.prop"/>
<exporterParams moduleName="RelANNISExporter" destinationPath="./relANNIS/"
    formatVersion="3.2" formatName="relANNIS" />
...
```

In this way you do not need to include a granular token layer in your EXMaRALDA data. Alternatively you can create an artificial token layer manually, with one token per time-line event. This has the same effect as using the `Timeline2Token` module.

3. Time alignment

Alignment of annotations to a timeline expressing actual units of time (milliseconds etc.) is important for using streaming A/V data that is triggered by individual annotations. The alignment relies on the existence of two types of annotation. Firstly, an A/V file must be aligned to a span containing the tokens and annotations that should correspond to the file. This is achieved by using a span node with a separate namespace ("audio" and "video" are used as namespaces for audio and video respectively by default, but other namespaces

are possible if the resolver_vis_map.tab is configured to address them). The span must carry an annotation named “audio” or “video” respectively, with the value taking e.g. the following format:

```
[ExtFile]subcorpus/subcorpus/document/file.webm
```

The value must contain the path to the document containing the annotation, including subcorpora (if these are used), and in all cases the name of the document, as well as the name of the file.

Secondly, there must be time alignment annotations to align segments of this file to particular token spans. This is done by assigning a span annotation with the reserved namespace “annis” and the annotation name “time”. The value of the annotation is an absolute range of time within the file in seconds (beginning and end), using ‘double’ number precision or less, e.g. 3.3859730542290265-3.564405882202723 for the time from approximately 3.38 seconds to 3.56 seconds in the file. For some examples, see the demo corpora on the ANNIS3 page.

If you are converting data from EXMaRALDA XML with time alignment information in the original file, the information should automatically be converted by SaltNPepper to relANNIS3.2. If using PAULA XML, you will have to create an appropriate annotation layer containing the values above, with the appropriate namespaces (see the PAULA documentation for more information).

4. Searching with segmentation layers

4.1 Searching for basic text

The first effect of multiple segmentation layers is that the most granular tokens are usually no longer meaningful for searches. It is possible to search explicitly for any segmentation layer by using its name in AQL (for details on AQL see the ANNIS website and User Guide). Compare:

```
norm="Isaac"
```

```
dipl="yſ≠" & dipl="aac" & #1 . #2
```

However in many scenarios, users won’t know which layer they want to use, as in the case of alternative tokenizations of “New York” made by a parser and a tagger. In order to search for “New York” in any segmentation layer, use no annotation name – just quotation marks. The same applies to regular expressions.

```
"New York"
```

```
/New.*/
```

The first query finds only segmentation layer positions with the segment “New York”, as analyzed e.g. by a parser. The second query finds both “New” and “New York”, even at

the same point in the corpus, if two elements exist matching these segmentations. Therefore if the tagger tagged “New” separately and the parser didn’t, the second query returns two matches for the regular expression.

It is still possible to search for the ‘real’ underlying tokens, which may or may not contain text of their own depending on the corpus, by explicitly using the reserved attribute `tok`:

```
tok="New"
```

4.2 Defining context

As soon as a corpus with multiple segmentation layers is selected, ANNIS3 will allow you to choose a segmentation layer for counting context units. By default, the underlying maximally granular tokens are used (plus/minus *n* tokens). You can change this setting by going to “search options” in the search form on the left and using the menu “show context in”, as shown in Figure 1 below.

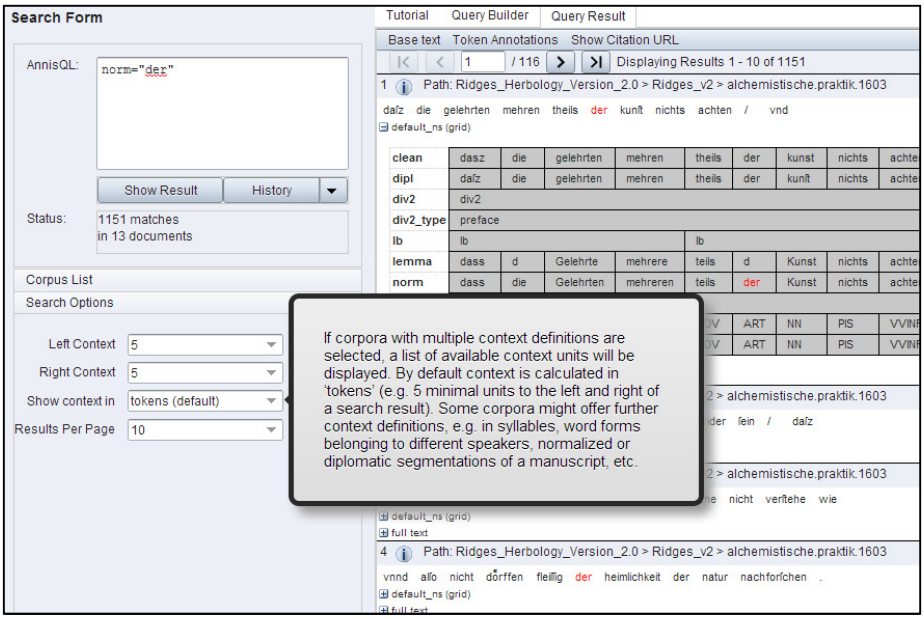

Figure 1. Choosing the context segmentation.

Change this setting to any segmentation, and search results will display a context of plus/minus *n* units (as defined by the Right and Left Context menus) in terms of the segmentation you have chosen. For how to configure default segmentations for each corpus, see the ANNIS User Guide.

Note that you may define any number of higher order segmentations in your corpus, such as chunks or sentences. You will then be able to select a context of *n* sentences, chunks etc.

4.3 Choosing a base text visualization

The basic visualization for a search result in ANNIS is the KWIC (key word in context) view, which shows only tokens and their annotations. If multiple segmentations are present in your search result, you can select the menu 'base text' to choose which segmentation will serve as the basis for the KWIC text, as shown in Figure 2.

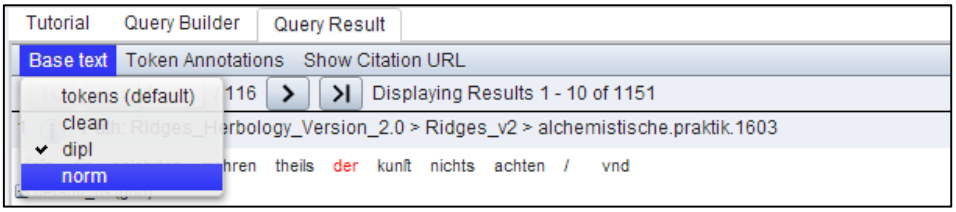

Figure 2. Choosing the context segmentation.

This setting can be changed repeatedly after a search result has been retrieved. Alternatively you may wish to hide the KWIC visualization altogether and display one or more grid views or other visualizations by default for your corpus. This can be achieved using the resolver_vis_map table in your database, or also in the file resolver_vis_map.tab in your relANNIS corpus. To hide the KWIC visualization, use the following line in the file:

```
my_corpus	NULL	NULL	NULL	kwic	kwic	removed 0	NULL
```

The display type 'removed' makes the KWIC visualization not appear under any circumstances. To display a grid instead without having to click it open manually, use the following line with the display type 'permanent' as an example.

```
my_corpus	NULL	my_ns	node	grid	my_name permanent	0	NULL
```

If you wish to hide the granular token layer in the grid (since it is empty/meaningless/ superfluous) add the following option in the final column:

```
my_corpus	NULL	my_ns	node	grid	my_name permanent	0	hide_tok:true
```

For more information on configuring visualizations, see the ANNIS User Guide.

4.4 Searching for adjacent elements

Just as context depends on the segmentation layer used, so does adjacency and distance in terms of tokens between units. If we search for two words "within 3 to 4 tokens of each other", the results can differ depending on the segmentation layer. The precedence operator in AQL "." checks for adjacency on the underlying token layer (or 'timeline'). Similarly, using specific distances checks distance in tokens.

parser_tok:	New York		would	've	been
tagger_tok:	New	York	would've		been
tok:	New	York	would	've	been

Table 2. Data for adjacency queries.

For the data in Table 2, the following queries return one hit:

```
tagger_tok="New" & tagger_tok="York" & #1 . #2
tagger_tok="New" & tagger_tok="been" & #1 .2,4 #2
```

But the following query returns no result, even though "been" is three segments away from "New":

```
tagger_tok="New" & tagger_tok="been" & #1 .3 #2
```

The reason is that the query measures distance on the "tok" layer, not on "tagger_tok". To change this, we can explicitly query distance on the "tagger_tok" layer:

```
tagger_tok="New" & tagger_tok="been" & #1 .tagger_tok,3 #2
```

This query returns the expected match. The "." operator can be extended to any segmentation layer that has been defined in the corpus, allowing the user to work on that layer even if it is split up in the underlying representation.

Similarly, it is possible to look for the next segment in dialogue data, even if another speaker makes intervening utterances. Consider the data in Table 3.

speaker1:	Why?				No!
speaker2:		I	already	told	you
tok:					

Table 3. Dialogue data.

Note that the "tok" is empty, since there is no sensible text to put in it, but it is still required in order to allow, e.g., for the word "I" to begin in the middle of the time span of the other speaker saying "Why?". To search for "Why?" followed by "No!", only the second of the following two queries works:

```
speaker1="Why?" & speaker1="No!" & #1 . #2
speaker1="Why?" & speaker1="No!" & #1 .speaker1 #2
```

This is because "Why?" and "No!" are not strictly adjacent on the time line. However, "No!" is the next thing speaker1 says after "Why?". The ordering relation ensures that the two units are seen as adjacent on the operator ".speaker1".

5. Further information

This guide is mainly meant to be a primer on the new architecture of ANNIS3 for users familiar with ANNIS2 or single-segmentation corpora in ANNIS3. The ANNIS User Guide is more comprehensive and may answer many further questions you might have, so in case of questions, try having a look there first.

ANNIS3 is under intense development and some features may not be stable yet. If you find a bug or wish to file a request for a new feature, please go to the following web page and select “new issue”:

https://github.com/korpling/ANNIS/issues

If you are working on the ANNIS3 developer’s server, you can also use the “report bug” button at the top left of the interface. This will give us some information on the state of the software while your bug occurred. For all further inquiries, please contact us at:

annis-admin@ling.uni-potsdam.de.