

ANNIS

Manual

Amir Zeldes

Thomas Krause

ANNIS: Manual

Amir Zeldes

Thomas Krause

Version \${project.version}

Publication date \${mavenBuildTimestamp}

Table of Contents

| | |
|---|----|
| ANNIS User Guide | 1 |
| 1. Introduction | 1 |
| 2. Installing a Local Version (ANNIS Kickstarter) | 1 |
| 3. Running Queries in ANNIS | 3 |
| 3.1. The ANNIS Interface | 3 |
| 3.2. Using the ANNIS Query Builder | 5 |
| 3.3. Searching for Word Forms | 7 |
| 3.4. Searching for Annotations | 7 |
| 3.5. Searching using Regular Expressions | 8 |
| 3.6. Searching for Trees | 9 |
| 3.7. Searching for Pointing Relations – Coreference and Dependencies | 10 |
| 3.8. Exporting Search Results | 11 |
| 3.9. Complete List of Operators | 13 |
| 4. Converting Corpora for ANNIS using SaltNPepper | 15 |
| Administration Guide | 16 |
| 1. Installing an ANNIS public server | 16 |
| 2. ANNIS back-end service | 17 |
| 2.1. User Configuration | 17 |
| 3. Front-end web application | 18 |
| 3.1. Configuring Tomcat or Jetty as application container | 18 |
| 3.2. General configuration advices | 19 |
| 3.3. Create and configure instances | 19 |
| 3.4. Configuring Visualizations | 20 |

List of Figures

| | |
|--------------------------|---|
| 1. ANNIS interface | 3 |
|--------------------------|---|

List of Tables

| | |
|------------------------------------|----|
| 1. | 13 |
| 1. Folders for configuration | 19 |

ANNIS User Guide

Amir Zeldes <annis-admin@ling.uni-potsdam.de>

Version \${project.version}

\${mavenBuildTimestamp}

1. Introduction

ANNIS is an open source, browser-based search and visualization architecture for multi-layer corpora. It can be used to search for complex graph structures of annotated nodes and edges forming a variety of linguistic structures, such as constituent or dependency syntax trees, coreference and parallel alignment edges, span annotations and associated multi-modal data (audio/video). This guide provides an overview of the current ANNIS system, first steps for installing either a local instance or an ANNIS server with a demo corpus, as well as tutorials for converting data for ANNIS and running queries with AQL (ANNIS Query Language).

2. Installing a Local Version (ANNIS Kickstarter)

Local users who do not wish to make their corpora available online can install ANNIS Kickstarter under most versions of Linux, Windows and Mac OS. To install Kickstarter follow these steps:

1. Download and install PostgreSQL 9.2 for your operating system from <http://www.postgresql.org/download/> and **make a note of the administrator password** you set during the installation.



Note

Under Linux, you might have to set the PostgreSQL password manually. E.g. on Ubuntu you can achieve this with by running the following commands:

```
sudo -u postgres psql
\password
\q
```

After installation, PostgreSQL may automatically launch the PostgreSQL Stack Builder to download additional components – you can safely skip this step and cancel the Stack Builder if you wish. You may need to restart your OS if the Postgres installer tells you to.

2. Download and unzip [annis-kickstarter-\\${project.version}-distribution.zip](https://github.com/downloads/korpling/ANNIS/annis-kickstarter-${project.version}-distribution.zip) [https://github.com/downloads/korpling/ANNIS/annis-kickstarter-\${project.version}-distribution.zip] from the ANNIS website.
3. Start AnnisKickstarter.bat if you're using Windows or run the bash script AnnisKickstarter.sh otherwise (this may take a few seconds the first time you run Kickstarter). At this point your Firewall may try to block Kickstarter and offer you to unblock it – do so and Kickstarter should start up.



Note

For most users it is a good idea to give Java more memory (if this is not already the default). You can do this by editing the script `AnnisKickstarter` and typing the following after the call to start java (before `-splash:splashscreen.gif`):

```
-Xss1024k -Xmx1024m
```

(To accelerate searches it is also possible to give the Postgres database more memory, see the link in the next section below).

4. Once the program has started, if this is the first time you run Kickstarter, press “Init Database” and supply your PostGres administrator password from step 1.
5. Download and unzip the [pcc2 demo corpus](http://korpling.german.huberlin.de/~annis/downloads/sample_corpora/pcc2_relAnnis.zip) [http://korpling.german.huberlin.de/~annis/downloads/sample_corpora/pcc2_relAnnis.zip] from the ANNIS website.
6. Press “Import Corpus” and navigate to the directory containing the directory `pcc2_v2_relAnnis/`. Select this directory (but do not go into it) and press OK.
7. Once import is complete, press “Launch Annis frontend” and login with the username and password “test” to test the corpus (try selecting the pcc2 corpus, typing `pos="NN"` in the AnnisQL box and clicking “Show Result”. See the section “Running Queries in ANNIS” in this guide for some more example queries, or press the Tutorial button at the top left of the interface).

3. Running Queries in ANNIS

3.1. The ANNIS Interface

Figure 1. ANNIS interface


The screenshot displays the ANNIS interface with two main windows. The left window, titled 'Search Form', contains an 'AnnisQL' input field with the query: `tok & tok & #1 ->dep #2 & cat="S" & #3 _i #1 & node & #3 >secedge #4`. Below the input field are 'Show Result' and 'History' buttons. The 'Status' section shows '12 matches in 1 document'. The 'Corpus List' table lists corpora: 'pcc2' (2 texts, 399 tokens), 'Ridges_Herbology_Versik' (13 texts, 60811 tokens), and 'SMULTRON_Banana' (2 texts, 3782 tokens). The right window, titled 'Query Result', shows the 'Query Result' tab. It displays a dependency graph for the sentence 'darüber streiten, was Jugendliche wollen und brauchen, ohne auf die Idee'. Below the graph is a table of dependencies with columns for path, tokens, and grammatical roles. The table shows dependencies between 'was' and 'Jugendliche', 'wollen' and 'brauchen', and 'ohne' and 'auf die Idee'.




















The ANNIS interface (see [Figure 1, "ANNIS interface"](#)) is comprised of several windows, the most important of which are the search form (in the red box above left) and the results window (in the blue box above).

3.1.1. The Search Form

The screenshot shows the 'Search Form' window. It has an 'AnnisQL' input field with the query: `pcc="S,."`. Below the input field are 'Show Result' and 'History' buttons. The 'Status' section shows '399 matches'. The 'Corpus List' table lists corpora: 'pcc-11' (11 texts, 1939 tokens), 'pcc2' (2 texts, 399 tokens), 'pcc2_plus' (2 texts, 399 tokens), and 'zh.dep.test' (1 text, 12 tokens). The 'pcc2' corpus is highlighted. At the bottom are 'Search Options' and 'Export' buttons.

The Search Form on the left of the interface window is available immediately after login. In the middle, the list of currently available corpora is shown. Using the checkboxes on the left of each corpus, it is possible to select which corpora should be searched in (hold down 'shift' to select multiple corpora simultaneously). If you cannot see a corpus that should be available to you, or else if the corpora list is too cluttered, you may click on "more corpora" to open the corpora window. You may then drag and drop the desired or unwanted corpora between the list and the window.

Pressing the  button next to a corpus in the list will open the corpus explorer window (see picture below), which shows metadata for the entire corpus on the left and a list of available annotations and example queries on the right. Clicking on a query will copy it to the "AnnisQL" field at the top of the form. Pressing the link icon will give you a citation link that can be used to access the query from any browser. If the corpus contains hierarchical structures, such as dominance edges or pointing relations, there will be separate segments on the right hand side of the corpus explorer to show the available edge names and annotations together with example queries. Clicking on a query will copy it to the "AnnisQL" field at the top of the form. Pressing the link icon will give you a citation link that can be used to access the query from any browser. If the corpus contains hierarchical structures, such as dominance edges or pointing relations, there will be separate segments on the right hand side of the corpus explorer to show the available edge names and annotations together with example queries.

| Metadata | | Available annotations | | |
|------------------------|--|-----------------------|--------------------------------|---|
| Name | Value | Node annotations | | |
| annotation_description | POS, lemma, morphology, constituent and dependency syntax, information structure, coreference, rhetorical structure | Name | Example (click to use query) | URL |
| annotation_levels | pos;lemma;morph;Inf-Stat;Focus_newInf;PP;NP;Topic;Sent;Foc_cont (for dominance edges);dep:func (for dependency pointing relations);anaphor_antecedent (pointing relations) | ambiguity | ambiguity="not_ambig" |  |
| full_name | Potsdam Commentary Corpus (sample of 2 documents) | anaphor_type | anaphor_type="anaphor_nominal" |  |
| language | German | cat | cat="S" |  |
| source | Project D1, SFB 632 | complex_np | complex_np="no" |  |
| URL | link | dir_speech | dir_speech="text_level" |  |
| version | 2.0 | Focus_newInf | Focus_newInf="nf-unsol" |  |
| | | grammatical_role | grammatical_role="other" |  |
| | | Inf-Stat | Inf-Stat="giv-active" |  |
| | | lemma | lemma="der" |  |
| | | morph | morph="3.PI.Pres.Ind" |  |
| | | NP | NP="NP" |  |
| | | np_form | np_form="defnp" |  |
| | | phrase_type | phrase_type="np" |  |
| | | pos | pos="NN" |  |
| | | PP | PP="PP" |  |
| | | referentiality | referentiality="discourse-new" |  |
| | | rst_relation | rst_relation="evaluation" |  |
| | | rstRel_type | rstRel_type="hypRelation" |  |
| | | Sent | Sent="s" |  |
| | | Edge types | | |
| | | Edge annotations | | |

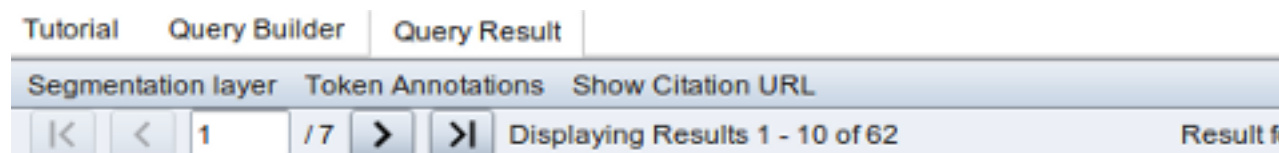
The "AnnisQL" field at the top of the form is used for inputting queries manually (see the tutorials on the ANNIS Query Language). As soon as one or several corpora are selected and a query is entered or modified, the query will be validated automatically and possible errors in the query syntax will be commented on in the "Result" box below. When modifying a query, a delay of two seconds is activated before the query is re-sent to the server for validation.

Once a valid query has been entered, pressing the "Show Result" button (or using the shortcut ctrl+Enter) will retrieve the number of matching positions in the selected corpora in the Result box and open the Result Window to display the first set of matches. Queries from the current session are saved in the query history and can be accessed using the button underneath the result field.

The context surrounding the matching expressions in the result list is determined by the "context left" and "context right" options at the bottom of the

search form, and can be set to up to 10 tokens on each side, though some corpora allow longer spans, such as entire texts, to be viewed using special discourse visualizations.

3.1.2. The Result Window

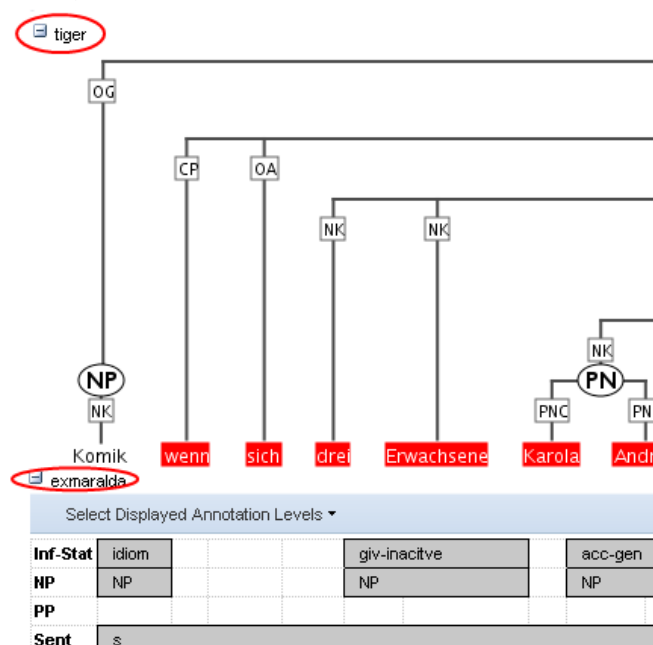


The result window shows search results in pages of 10 hits each by default (this can be changed in the Search Form). The toolbar at the top of the window allows you to navigate between these pages. The "Token Annotations" button on the toolbar allows you to toggle the token based annotations, such as lemmas and parts-of-speech, on or off for your convenience. The "Citation URL" button provides a hyperlink which you can e-mail or cite, allowing others to reproduce your query.

| | | | | | | | |
|---------|----|------|-------------|-------------|-------------|-------------|---------------|
| loblich | . | Mit | dem | Treffen | im | Rathaus | ist |
| loblich | . | mit | der | Treffen | in | Rathaus | sein |
| Pos | -- | -- | Dat.Sg.Neut | Dat.Sg.Neut | Dat.Sg.Neut | Dat.Sg.Neut | 3.Sg.Pres.Ind |
| ADJD | \$ | APPR | ART | NN | ADPART | NN | VAFIN |

token_merged::morph

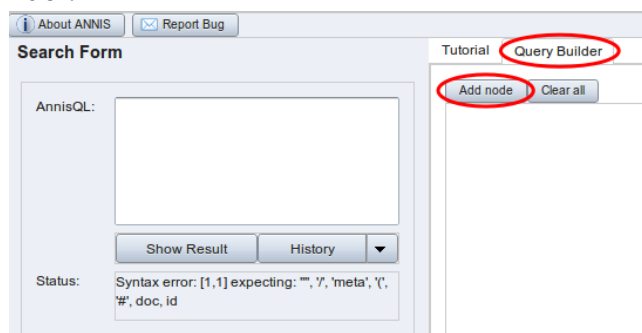
The result list itself initially shows a KWIC (key word in context) concordance of matching positions in the selected corpora, with the matching region marked red and the context in black on either side. Token annotations are displayed in gray under each token, and hovering over them with the mouse will show the annotation name and namespace. More complex annotation levels can be expanded, if available, by clicking on the plus icon next to the level's name, e.g. tiger and exmaralda for the annotations in the tree and grid views in the picture below (circled in red).






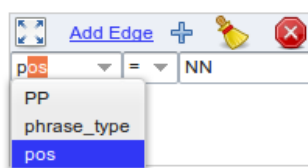
3.2. Using the ANNIS Query Builder

To open the graphical query builder, click on the Query Builder tab. On the left-hand side of the toolbar at the top of the query builder canvas, you will see the Create Node button. Use this button to define nodes to be searched for (tokens,

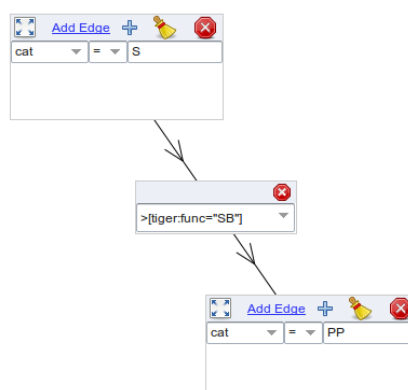
non-terminal nodes or annotations). Creating nodes and modifying them on the canvas will immediately update the AnnisQL field in the Search Form with your query, though updating the query on the Search Form will not create a new graph in the Query Builder.



In each node you create you may click on  to specify an annotation value. The annotation name can be typed or selected from a drop down list. The operator field in the middle allows you to choose between an exact match (the '=' symbol) or wildcard search using Regular Expressions (the '~' symbol). The annotation value is given on the right, and should NOT be surrounded by quotations (see the example below). It is also possible to specify multiple annotations applying to the same position by clicking on  multiple times. Clicking on  will delete the values in the node. To search for word forms, simply leave the field name on the left empty and type directly on the right under "Value". A node with no data entered will match any node, that is an underspecified token or non-terminal node or annotation.



To specify the relationship between nodes, first click on the "Edge" button at the top left of one node, and then click the "Dock" button which becomes available on the other nodes. An edge will connect the nodes with an extra box from which operators may be selected (see below). For operators allowing additional labels (e.g. the dominance operator > allows edge labels to be specified), you may type directly into the edge's operator box, as in the example with a "func" label in the image below. Note that the node clicked on first (where the "Edge" button was clicked) will be the first node in the resulting query, i.e. if this is the first node it will dominate the second node (#1 > #2) and not the other way around, as also represented by the arrows along the edge.



3.3. Searching for Word Forms

To search for word forms in ANNIS, simply select a corpus (in this example the small pcc2 demo corpus) and enter a search string between double quotation marks, e.g.:

```
"statisch"
```

Note that the search is case sensitive, so it will not find cases of capitalized 'Statisch', for example at the beginning of a sentence. In order to find both options, you can either look for one form OR the other using the pipe sign (|):

```
"statisch" | "Statisch"
```

or else you can use regular expressions, which must be surrounded by slashes (/) instead of quotation marks:

```
/[Ss]tatisch/
```

To look for a sequence of multiple word forms, enter your search terms separated by & and then specify that the relation between the elements is one of precedence, as signified by the period (.) operator:

```
"so" & "statisch" & #1 . #2
```

The expression #1 . #2 signifies that the first element ("so") precedes the second element ("statisch"). For indirect precedence (where other tokens may stand between the search terms), use the .* operator:

```
/[Ss]o/ & "statisch" & "wie" & #1 . #2 & #2 .* #3
```

The above query finds sequences beginning with either "So" or "so", followed directly by "statisch", which must be followed either directly or indirectly (.*) by "wie". A range of allowed distances can also be specified numerically as follows:

```
/[Ss]tatisch/ & "wie" & #1 .1,5 #2
```

Meaning the two words may appear at a distance of 1 to 5 tokens. The operator .* allows a distance of up to 50 tokens by default, so searching with .1,50 is the same as using .* instead. Greater distances (e.g. .1,100 for 'within 100 tokens') should always be specified explicitly. Finally, we can add metadata restrictions to the query, which filter out documents not matching our definitions. Metadata attributes must be preceded by the prefix meta:: and may not be bound (i.e. they are not referred to as #1 etc. and the numbering of other elements ignores their existence):

```
/[Ss]tatisch/ & "wie" & #1 .1,5 #2 & meta::Genre="Sport"
```

To view metadata for a search result or for a corpus, press the "i" icon next to it in the result window or in the search form respectively.

3.4. Searching for Annotations

Annotations may be searched for using an annotation name and value. The names of the annotations vary from corpus to corpus, though many corpora contain part-of-speech and lemma annotations with the names pos and lemma respectively (annotation names are case sensitive). For example, to search for all forms of the German verb sein 'to be' in a corpus with lemma annotation such as pcc2, simply select the pcc2 corpus and enter:

```
lemma="sein"
```

Negative searches are also possible using != instead of =. For negated tokens (word forms) use the reserved attribute tok. For example:

```
lemma!="sein"
```

or:

```
tok!="ist"
```

Metadata can also be negated similarly:

```
lemma="sein" & meta::Genre!="Sport"
```

To only find finite forms of this verb in pcc2, use the part-of-speech (pos) annotation concurrently, and specify that both the lemma and pos should apply to the same element:

```
lemma="sein" & pos="VAFIN" & #1 _=_ #2
```

The expression #1 _=_ #2 uses the span identity operator to specify that the first annotation and the second annotation apply to exactly the same position in the corpus. Annotations can also apply to longer spans than a single token: for example, in pcc2, the annotation Inf-Stat signifies the information structure status of a discourse referent. This annotation can also apply to phrases longer than one token. The following query finds spans containing new discourse referents, not previously mentioned in the text:

```
exmaralda:Inf-Stat="new"
```

If the corpus contains no more than one annotation type named Inf-Stat, the optional namespace (in this case exmaralda:) may be dropped; if there are multiple annotations with the same name but different namespaces, dropping the namespace will find all of those annotations. In order to view the span of tokens to which this annotation applies, enter the and click on "Show Result", then open the exmaralda annotation level to view the grid containing the span. Further operators can test the relationships between potentially overlapping annotations in spans. For example, the operator _i_ examines whether one annotation fully contains the span of another annotation (the i stands for 'includes'):

```
Topic="ab" & Inf-Stat="new" & #1 _i_ #2
```

This query finds aboutness topics (Topic="ab") containing information structurally new discourse referents.

3.5. Searching using Regular Expressions

When searching for word forms and annotation values, it is possible to employ wildcards as placeholders for a variety of characters, using Regular Expression syntax (see <http://www.regular-expressions.info/> for detailed information). To search for wildcards use slashes instead of quotation marks to surround your search term. For example, you can use the period (.) to replace any single character:

```
tok=/de./
```

This finds word forms such as "der", "dem", "den" etc. It is also possible to make characters optional by following them with a question mark (?). The following example finds cases of "das" and "dass", since the second "s" is optional:

```
tok=/dass?/
```

It is also possible to specify an arbitrary number of repetitions, with an asterisk (*) signifying zero or more occurrences and a plus (+) signifying at least one occurrence. For example, the first query below finds "da", "das", and "dass" (since the asterisk means zero or more times the preceding "s"), while the second finds "das" and "dass", since at least one "s" must be found:

```
tok=/das*/
```

```
tok=/das+/
```

It is possible to combine these operators with the period operator to mean any number of occurrences of an arbitrary character. For example, the query below searches for pos (part-of-speech) annotations that begin with "VA", corresponding to all forms of auxiliary verbs. The string "VA" means that the result must begin with "VA", the period stands for any character, and the asterisk means that 'any character' can be repeated zero or more time, as above.

```
pos=/VA.*/
```

This finds both finite verbs ("VAFIN") and non-finite ones ("VAINF"). It is also possible to search for explicit alternatives by either specifying characters in square brackets or longer strings in round brackets separated by pipe signs. The first example below finds either "dem" or "der" (i.e. "de" followed by either "m" or "r") while the second example finds lemma annotations that are either "sein" or "werden".

```
tok=/de[mr]/
```

```
lemma=/(sein|werden)/
```

Finally, negative searches can be used as usual with the exclamation point, and regular expressions can generally be used also in edge annotations. For example, if we search for trees (see also Searching for Trees below) where a node dominates another node with edges not containing an object, we can use a wildcard to rule out all edges labels beginning with "O" for object:

```
cat="VP" & cat & #1 >[func!="/O.*/] #2
```

3.6. Searching for Trees

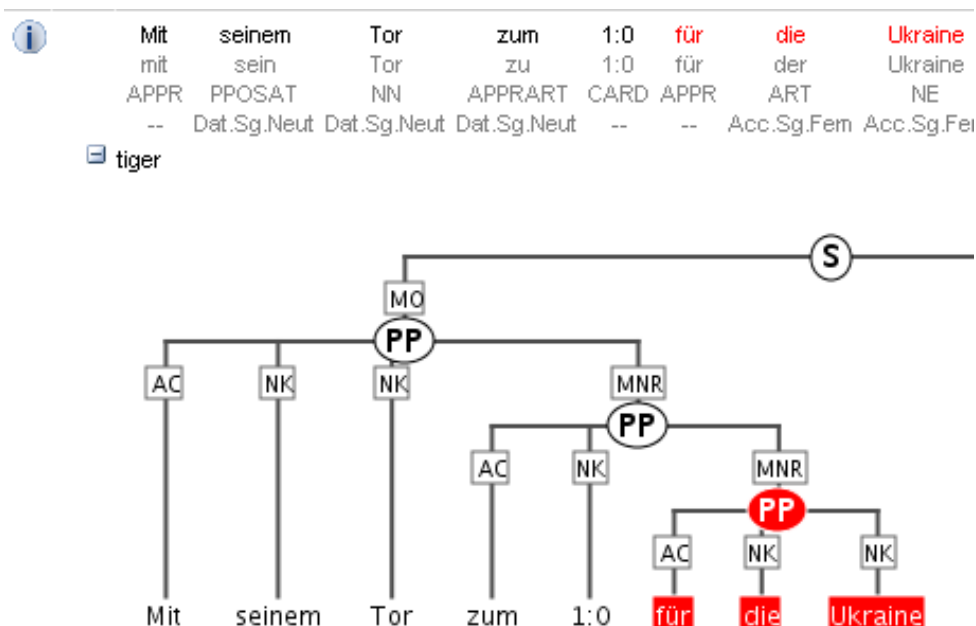
In corpora containing hierarchical structures, annotations such as syntax trees can be searched for by defining terminal or non-terminal node annotations and their values. A simple search for prepositional phrases in the small pcc2 demo corpus looks like this:

```
tiger:cat="PP"
```

If the corpus contains no more than one annotation called cat, the optional namespace, in this case tiger:, may be dropped. This finds all PP nodes in the corpus. To find all PP nodes directly dominating a proper name, a second element can be specified with the appropriate part-of-speech (pos) value:

```
cat="PP" & pos="NE" & #1 > #2
```

The operator > signifies direct dominance, which must hold between the first and the second element. Once the Result Window is shown you may open the "tiger" annotation level to see the corresponding tree.



3.7. Searching for Pointing Relations - Coreference and Dependencies

Pointing relations are used to express an arbitrary directed relationship between two elements (terminals or non-terminals) without implying dominance or coverage inheritance. For instance, in the pcc2 demo corpus, elements in the mmax: namespace may point to each other to express coreference or anaphoric relations. The following query searches for two np_form annotations, which specify for example whether a nominal phrase is pronominal, definite or indefinite.

```
mmax:np_form="pper" &
mmax:np_form="defnp" &
#1 ->anaphor_antecedent #2
```

Using the pointing relation operator -> with the type anaphor_antecedent, the first np_form, which should be a personal pronoun (pper), is said to be the anaphor to its antecedent, the second np_form, which is definite (defnp). To see a visualization of the coreference relations, open the mmax annotation level in the example corpus. In the image below, one of the matches for the above query is highlighted in red (die Seeburger und einige Groß-Glienicker ... sie 'the Seeburgers and some Groß-Glienickers... they'). Other discourse referents in the text (marked with an underline) may be clicked on, causing coreferential chains containing them to be highlighted as well. Note that discourse referents may overlap, leading to multiple underlines: Die Seeburger 'the Seeburgers' is a shorter discourse referent overlapping with the larger one ('the Seeburgers and some Groß-Glienickers'), and each referent has its own underline. Annotations of the coreference edges of each relation can be viewed by hovering of the appropriate underline.

coreference (discourse)
 Steilpass Wunder gibt es immer wieder ! Erst spielen die **Dallgower** Gemeindevertreter so statisch und verzagt wie **die deutsche Abwehrreihe der Fußballkicker** . Und dann kommt aus der Tiefe solch ein fulminanter Steilpass , von dem man hofft , dass **die Seeburger** oder Groß-Glienicker **Mitspieler** ihn aufnehmen können . Ein Befreiungsschlag ist es allerdings nicht , weil es vorerst keine Gefahr fürs **Dallgower** Tor gab . **Die Seeburger und einige Groß-Glienicker** haben den Ball erst zurückgespielt und dann um so drängender wieder gefordert . Nun sollen **sie** zeigen , wie **sie** die Chance verwerten . Eine Diskussion , wo künftig die Trainerkabine stehen soll , wäre in der jetzigen Spielsituation verheerend . Und eine Parallele **zu den deutschen Grotten-Kickern** gibt es immer noch . Auch wenn **die Spieler** aus den verschiedenen Vereinen zusammengewürfelt sind , **sie** müssen sich daran gewöhnen , dass **sie** nun in einer Mannschaft " Döberitzer Heide " spielen . Und das heißt gemeinsam und nicht gegeneinander . Ermahnungen von der Seitenlinie , miteinander fair umzugehen und sich nicht beim kleinsten Schubser gegenseitig zu zerfleischen .

Another way to use pointing relations is found in syntactic dependency trees. The queries in this case can use both pointing relation types and annotation, as in the following query:

```
pos="VVFIN" & tok & #1 ->dep[func="obja"] #2
```

This query searches for a finite verb (with the part-of-speech VVFIN) and a token, with a pointing relation of the type 'dep' (for dependency) between the two, annotated with 'func="obja"' (the function Object, Accusative). The result can be viewed with the dependency arch visualizer, which shows the verb gibt 'gives' and its object Wunder 'miracles'.



3.8. Exporting Search Results

By going to the Export tab at the bottom of the search form on the left, you can select one of several exporters:

The **SimpleTextExporter** simply gives the text for all tokens in each search result, including context, in a one-row-per-hit format. The tokens covered by the match area are marked with square brackets and the results are numbered, as in the following example

1. Tor zum 1:0 für die [Ukraine] stürzte der 1,62 Meter große

2. der 1,62 Meter große Gennadi [Subow] die deutsche Nationalelf
vorübergehend in
3. und Reputation kämpfenden Mannschaft von [Rudi] Völler der Weg
zur Weltmeisterschaft
4. Reputation kämpfenden Mannschaft von Rudi [Völler] der Weg zur
Weltmeisterschaft endgültig
5. die deutschen Nationalkicker einen " [Rudi] Riese " auf der Bank

The **TextExporter** adds all annotations of each token separated by slashes (e.g. dogs/NN/dog for the token dogs annotated with a part-of-speech NN and a lemma dog).

The **GridExporter** adds all annotations available for the span of retrieved tokens, with each annotation layer in a separate line. Annotations are separated by spaces and the hierarchical order of annotations is lost, though the span of tokens covered by each annotation may optionally be given in square brackets (to turn this off use the optional parameter `numbers=false`). The user can specify annotation layers to be exported in the additional 'Parameters' box, using the setting `'keys='` and annotation names separated by commas. If nothing is specified in the parameters box, all available annotations will be exported. Multiple options are separated by a semicolon, e.g. `keys=tok,pos,cat;numbers=false`. An example output with token numbers looks as follows.

```
1. tok ein Dialog zwischen den Generationen angestoßen .
cat NP[1-5] S[1-6] VP[1-6] PP[3-5]
pos ART[1-1] NN[2-2] APPR[3-3] ART[4-4] NN[5-5] VVPP[6-6] $.[7-7]
```

Meaning that the annotation `cat="NP"` applied to tokens 1-5 in the search result, and so on. Note that when specifying annotation layers, if the reserved name **'tok'** is not specified, the tokens themselves will not be exported (annotations only).

The **WekaExporter** outputs the format used by the WEKA machine learning tool (<http://www.cs.waikato.ac.nz/ml/weka/>). Only the attributes of the search elements (#1, #2 etc. in AQL) are outputted, and are separated by commas. The order and name of the attributes is declared in the beginning of the export text, as in this example:

```
@relation name
@attribute #1_id string
@attribute #1_token string
@attribute #1_tiger:cat string
@attribute #2_id string
@attribute #2_token string
@attribute #2_tiger:lemma string
@attribute #2_tiger:morph string
@attribute #2_tiger:pos string
@data
'288662','NULL','NP','288392','ganze','ganz','Pos.Acc.Sg.Fem',
'ADJA'
'289175','NULL','NP','288712','geladenen','geladen',
'Pos.Nom.Pl.*','ADJA'
'289660','NULL','NP','289409','Döberitzer','Döberitzer',
'Pos.*.*.*','ADJA'
'288672','NULL','NP','288302','deutschen','deutsch',
'Pos.Nom.Pl.Masc','ADJA'
'289614','NULL','NP','289291','deutsche','deutsch',
'Pos.Nom.Sg.Fem','ADJA'
```

```

'289625', 'NULL', 'NP', '289245', 'fulminanter', 'fulminant', '
  Pos.Nom.Sg.Masc', 'ADJA'
'288607', 'NULL', 'NP', '288242', 'einstige', 'einstig', '
  Pos.Nom.Sg.Fem', 'ADJA'
'288620', 'NULL', 'NP', '288334', 'ähnliche', 'ähnlich', '
  Pos.Acc.Pl.Neut', 'ADJA'
'289220', 'NULL', 'NP', '288883', 'große', 'groß', 'Pos.Nom.Sg.Fem', '
  ADJA'
'288610', 'NULL', 'NP', '288313', 'deutsche', 'deutsch', '
  Pos.Acc.Sg.Fem', 'ADJA'
'289174', 'NULL', 'NP', '288809', 'böse', 'böse', 'Pos.Nom.Sg.Fem', '
  ADJA'
'289611', 'NULL', 'NP', '289241', 'Dallgower', 'Dallgower', '
  Pos.*.*.*', 'ADJA'
'288624', 'NULL', 'NP', '288330', 'ukrainische', 'ukrainisch', '
  Pos.Nom.Sg.Masc', 'ADJA'

```

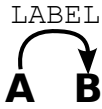

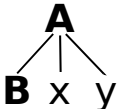
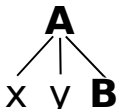


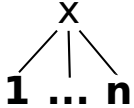
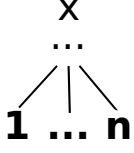
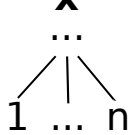
The export shows the properties of an NP node dominating a token with the part-of-speech ADJA. Since the token also has other attributes, such as the lemma, the token text and morphology, these are also retrieved. Note that exporting may be slow in both exporters if the result set is very large.

3.9. Complete List of Operators

The ANNIS Query Language (AQL) currently includes the following operators:

Table 1.

| Operator | Description | Illustration | Notes |
|----------|---------------------|---------------------------------------|--|
| . | direct precedence | AB | For non-terminal nodes, precedence is determined by the right most and left most terminal children |
| .* | indirect precedence | A x y z B | For specific sizes of precedence spans, .n,m can be used, e.g. .3,4 - between 3 and 4 token distance |
| > | direct dominance | A B | A specific edge type may be specified, e.g.: >secedge to find secondary edges. Edges labels are specified in brackets, e.g. >[func="0A"] for an edge with the function 'object, accusative |
| >* | indirect dominance | A ... B | For specific distance of dominance, >n,m can be used, e.g. >3,4 - dominates with 3 to 4 edges distance |
| _=_ | identical coverage | A B | Applies when two annotation cover the exact same span of tokens |
| _i_ | inclusion | AAA B | Applies when one annotation covers a span identical to or larger than another |

| Operator | Description | Illustration | Notes |
|---------------------|---------------------------|---|---|
| _o_ | overlap | AAA BBB | For overlap only on the left or right side, use _ol_ and _or_ respectively |
| _l_ | left aligned | AAA BB | Both elements span an area beginning with the same token |
| _r_ | right aligned | AAA BB | Both elements span an area ending with the same token |
| ->LABEL | labeled pointing relation |  | A labeled, directed relationship between two elements. Annotations can be specified with ->LABEL[annotation="VALUE"] |
| ->LABEL * | labeled pointing relation |  | An indirect labeled relationship between two elements. The length of the chain may be specified with ->LABEL n,m for relation chains of length n to m |
| >@l | left-most child |  | |
| >@r | right-most child |  | |
| >\$ | Common parent node |  | |
| >\$* | Common ancestor node |  | |
| #x:arity=n | Arity |  | Specifies the amount of directly dominated children that the searched node has |
| #x:length=n | Length |  | Specifies the length of the span of tokens covered by the node |
| #x:root | Root |  | node x is the root of a subgraph (i.e. it is not dominated by any node) |

4. Converting Corpora for ANNIS using SaltNPepper

ANNIS uses a relational database format called relANNIS. The Pepper converter framework allows users to convert data from various formats including PAULA XML, EXMARaLDA XML, TigerXML, CoNLL, RSTTool and TreeTagger directly into relAnnis (the Tiger XML conversion is limited to corpora without secondary edges at the moment). Further formats (including Tiger XML with secondary edges, mmax2) can be converted first into PAULA XML and then into relANNIS using the converters found on the ANNIS downloads page.

For complete information on converting corpora with SaltNPepper see: <http://korpling.german.hu-berlin.de/saltnpepper/>

Administration Guide

Amir Zeldes <annis-admin@ling.uni-potsdam.de>

Thomas Krause <thomas.krause@alumni.hu-berlin.de>

Version \${project.version}

\${mavenBuildTimestamp}

1. Installing an ANNIS public server

The ANNIS server version can be installed on UNIX based server, or else under Windows using [Cygwin](http://www.cygwin.com/) [http://www.cygwin.com/], the freely available UNIX emulator. To install the ANNIS server:

1. Install a PostgreSQL 9.2 server for your operating system from <http://www.postgresql.org/download/> and **make a note of the administrator password you set during the installation.**



Note

Under Linux, you might have to set the PostgreSQL password manually. E.g. on Ubuntu you can achieve this with by running the following commands:

```
sudo -u postgres psql
\password
\q
```

2. Install a web server such as [Tomcat](http://tomcat.apache.org/) [http://tomcat.apache.org/] or [Jetty](http://www.mortbay.org/jetty/) [http://www.mortbay.org/jetty/]
3. Make sure you have [JDK 6](http://java.sun.com/javase/downloads/index.jsp) [http://java.sun.com/javase/downloads/index.jsp] and [Maven 3](http://maven.apache.org/) [http://maven.apache.org/] (or install them if you don't)
4. Download [annis-service-\\${project.version}-distribution.tar.gz](https://github.com/downloads/korpling/ANNIS/annis-service-${project.version}-distribution.tar.gz) [https://github.com/downloads/korpling/ANNIS/annis-service-\${project.version}-distribution.tar.gz] and then unzip the downloaded file:

```
tar xzvf annis-service-${project.version}-distribution.tar.gz
-C <installation directory>
```

5. Set the environment variables (each time when starting up)

```
export ANNIS_HOME=<installation directory>
export PATH=$PATH:$ANNIS_HOME/bin
```

6. Next initialize your ANNIS database (only the first time you use the system):

```
annis-admin.sh init -u <username> -d <dbname> -p <password>
-P <postgres password>
```

7. Now you can import some corpora:

```
annis-admin.sh import path/to/corpus1 path/to/corpus2 ...
```



Important

The above import-command calls other PostgreSQL database commands. If you abort the import script with Ctrl+C, these SQL processes will not be automatically terminated; instead they might keep hanging and prevent access to the database. The same might happen if you close your shell before the import script terminates, so you will want to prefix it with the "nohup"-command.

8. Now you can start the ANNIS service:

```
annis-service.sh start
```

9. To get the ANNIS front-end running, first download it: [annis-gui-\\${project.version}.war](https://github.com/downloads/korpling/ANNIS/annis-gui-${project.version}.war) [https://github.com/downloads/korpling/ANNIS/annis-gui-\${project.version}.war] and deploy it to your Java servlet container (this is depending on the servlet container you use).

We also **strongly recommend** reconfiguring the Postgres server's default settings as described here: <https://korpling.german.hu-berlin.de/p/projects/annis/wiki/PostgreSQL>

2. ANNIS back-end service

2.1. User Configuration

ANNIS has an authentication system which allows to handle multiple users which will see different corpora depending on which groups the user is part of. Behind the scenes ANNIS uses the [Apache Shiro](http://shiro.apache.org/) [http://shiro.apache.org/] security framework. Per default ANNIS uses a file based authentication and authorization approach where some configuration files with an ANNIS specific layout are evaluated. This section will discuss how to manage this configuration. Additionally, the administrator can also directly adjust the contents of the conf/shiro.ini configuration file. This allows a much more individual configuration and the usage of external authorization services like LDAP.

2.1.1. Configuration file location

There is a central location where the user configuration files are stored. Configure the path to this location in the conf/shiro.info configuration file of the ANNIS service. The default path is /etc/annis/user_config_trunk/ and must be changed at *two locations* in the configuration file.

```
[main]
annisRealm = annis.security.ANNISUserRealm
annisRealm.resourcePath=/etc/annis/user_config_trunk/
annisRealm.authenticationCachingEnabled = true

globalPermResolver = annis.security.ANNISRolePermissionResolver
globalPermResolver.resourcePath = /etc/annis/user_config_trunk/
```

2.1.2. User and group files

1. Create a file "groups" in the user-configuration directory (e.g. /etc/annis/user_config_trunk/groups):

```
group1=pcc3,falko,tiger2
```

```
group2=pcc3
group3=tiger1
demo=pcc2,falko
```

This example means that a member of group group1 will have access to corpora with the names pcc3,falko, tiger2 (corpus names can be displayed with "**annis-admin.sh list**").

2. Create a subdirectory users
3. You have to create a file for each user inside the users subdirectory where the user's name is *exactly* the file name (no file endings).

```
groups=group1,group3
password=$shiro1$SHA-256$1$Kne+VMg3qnT/zTPK/laSQ==$lML04hAu[...]
given_name=userGivenName
surname=userSurname
```

- A superuser who has access to every corpus can be created with "groups=*"
- given_name and surname can be anything
- The password must be hashed with SHA256 (one iteration and using a Salt) and formatted in the [Shiro1CryptFormat](http://shiro.apache.org/static/current/apidocs/org/apache/shiro/crypto/hash/format/Shiro1CryptFormat.html) [http://shiro.apache.org/static/current/apidocs/org/apache/shiro/crypto/hash/format/Shiro1CryptFormat.html].

The easiest way to generate the password hash is to use the [Apache Shiro command line hasher](http://shiro.apache.org/command-line-hasher.html) [http://shiro.apache.org/command-line-hasher.html] which can be downloaded from <http://shiro.apache.org/download.html#Download-1.2.1.BinaryDistribution>.

1. Execute **java -jar shiro-tools-hasher-1.2.1-cli.jar -i 1 -p** from the command line (the jar-file must be in the working directory)
2. Type the password
3. Retype the password
4. It will produce the following output:

```
$ java -jar shiro-tools-hasher-1.2.1-cli.jar -i 1 -p
Password to hash:
Password to hash (confirm):
$shiro1$SHA-256$1$Kne+VMg3qnT/zTPK/laSQ==$lML04hAu[...]
```

The last line is what you have to insert into the password field.

3. Front-end web application

3.1. Configuring Tomcat or Jetty as application container

We are providing a WAR-file that is deployable by every common Java Servlet Container like Tomcat or Jetty. Please use the documentation of the web application container of your choice how to deploy these war-files.

3.1.1. UTF8 encoding in server.xml

If using Tomcat make sure the UTF-8 encoding is used for URLs. Some installations of Tomcat don't use UTF-8 for the encoding of the URLs and that will cause problems when searching for non-ASCII characters. In order to avoid this the Connector-configuration needs the property "URIEncoding" set to "UTF-8" like in this example (\$CATALINA_HOME/server.xml):

```
<Connector port="8080" protocol="HTTP/1.1"
  connectionTimeout="20000"
  URIEncoding="UTF-8"
  redirectPort="8443"
  executor="tomcatThreadPool" />
```

3.2. General configuration advices

The ANNIS frontend will search in different folders for it's configuration.

Table 1. Folders for configuration

| Folder | Description |
|------------------------------|---|
| <Installation>/WEB-INF/conf/ | Default configuration inside the deployed web application folder. Should not be changed. |
| \$ANNIS_CFG or /etc/annis/ | The global configuration folder defined by the environment variable ANNIS_CFG or a default path if not set. |
| ~/.annis/ | User specific configuration inside the .annis sub-folder inside the user's home folder |

Configuration files can be either in the [Java Properties](http://en.wikipedia.org/w/index.php?title=.properties&oldid=521500688) [http://en.wikipedia.org/w/index.php?title=.properties&oldid=521500688] or [JSON](http://www.json.org/) [http://www.json.org/] format. Configuration files from the user directory can overwrite the global configuration and the global configuration overwrites the default configuration.

3.3. Create and configure instances

When multiple corpora from different sources are hosted on one server it is often still desired to group the corpora by their origin and present them differently. You should not be forced to have an ANNIS frontend and service installation for each of this groups. Instead the administrator can define so called instances.

An instance is defined by a JSON file inside the instances sub-folder in one of the configuration locations. The name of the file also defines the instance name. Thus the file instances/falko.json defines the instance named "falko".

```
{
  "display-name": "Falko",
  "default-querybuilder": "tigersearch",
  "default-corpusset": "falko-essays",
  "corpus-sets": [
    {
      "name": "falko-essays",
```



```
    "corpus": ["falko-essay-l1", "falko-essay-l2"]
  },
  {
    "name": "falko-summaries",
    "corpus": ["falko-summary-l1", "falko-summary-l2"]
  }
]
```

Each instance configuration can have a verbose `display-name` which is displayed in the title of the browser window. `default-querybuilder` defines the short name of the query builder you want to use. Currently only "tigersearch" is available, see the developer guide if you want to add your own query builder.

While any user can group corpora into corpus sets for their own, you can define corpus sets for the whole instance. Each corpus set is an JSON-object with a name and a list of corpora that belong to the corpus set.

Any defined instance is assigned a special URL at which it can be accessed: `http://<server>/annis-gui/app/instance-<name>`. The default instance is additionally accessible by not specifying any instance name in the URL. You can configure your web server (e.g. Apache) to rewrite the URLs if you need a more project specific and less "technical" URL (e.g. `http://<server>/falko`).

3.4. Configuring Visualizations

3.4.1. Triggering Visualizations with the Resolver Table

By default, ANNIS displays all search results in the Key Word in Context (KWIC) view in the search result window. Further visualizations, such as syntax trees or grid views, are displayed by default based on the following namespaces:

| | |
|-------------------------------------|-------------------|
| Nodes with the namespace tiger: | tree visualizer |
| Nodes with the namespace exmaralda: | grid visualizer |
| Edges with the namespace mmax: | discourse view |
| Nodes with the namespace external: | multimedia player |

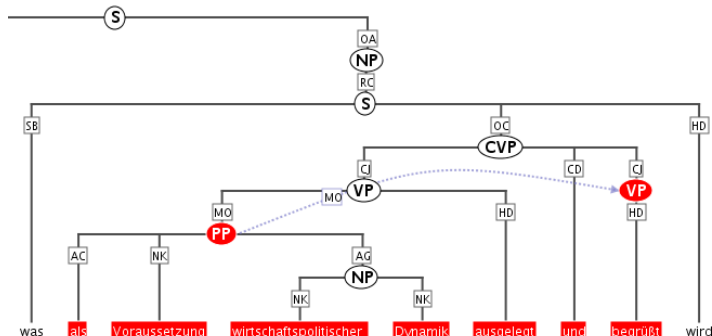
In these cases the namespaces are usually taken from the source format in which the corpus was generated, and carried over into relAnnis during the conversion. It is also possible to use other namespaces, most easily when working with PAULA XML. In PAULA XML, the namespace is determined by the string prefix before the first period in the file name / `paula_id` of each annotation layer. In order to manually determine the visualizer and the display name for each namespace in each corpus, the resolver table in the database must be edited. To do so, open PGAdmin (or if you did not install PGAdmin with ANNIS then via PSQL), and access the table `resolver_vis_map` (it can be found in PGAdmin under *PostgreSQL 9.1 > Databases > anniskickstart > Schemas > public > Tables* (for ANNIS servers replace "anniskickstart" with "annis_db"). You may need to give your PostgreSQL password to gain access. Right click on the table and select *View Data > View All Rows*. The table should look like this:

| | | | | | | | | | |
|---|---|--|------|------|------|------|--------|-----|--|
| 6 | 6 | | urml | node | grid | urml | hidden | 105 | |
|---|---|--|------|------|------|------|--------|-----|--|

Resolver table (resolver_vis_map)

The columns in the table can be filled out as follows:

- *tree* (constituent syntax tree)



- *grid* (annotation grid, with annotations spanning multiple tokens)

g) information structure (grid)

| | | | | | | | | | | | | | | | | | | |
|--------------|--------------|----|--------|----------|-----|-----------|---|-----|------------------------|-----|-----|--|--|------------|--|--|----|--|
| Focus_newInf | | | | nf-unsol | | | | | | | | | | | | | | |
| Inf-Stat | new | | | new | | new | | | glv-active | | | | | glv-active | | | | |
| NP | NP | | | NP | | NP | | | exmaralda:Focus_newInf | | | | | NP | | | | |
| PP | | | PP | | | | | | | | | | | | | | PP | |
| Sent | s | | | | | | | | s | | | | | | | | | |
| Topic | ab | | | | | | | | | | | | | | | | | |
| tok | Jugendlichen | in | Zossen | wollen | ein | Musikcafé | . | Das | forderten | sie | bei | | | | | | | |

- *grid_tree* (a grid visualizing hierarchical tree annotations as ordered grid layers; note that all layers represent the same annotation name at different hierarchical depths, marked level:0,1,2,... etc. on the left)

topo (grid)

| | | | | | | | |
|----------|-----|-----|-----|------------|---------------|------|---|
| level: 0 | TOP | | | | | | |
| level: 1 | VF | | | | | | |
| level: 2 | C | | C | MF | VC | | |
| tok | Daß | und | wie | Demokratie | funktionieren | kann | , |

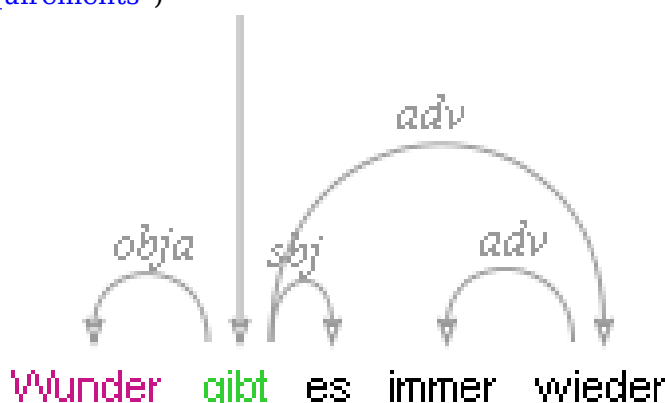
A known Bug is, that equal annotation names which cover the same range or a subset of nodes and have the same hierarchical depths are not rendered correctly by *grid_tree*.

- *discourse* (a view of the entire text of a document, possibly with interactive coreference links. It is possible to use this visualization to view entire texts even if you do not have coreference annotations)

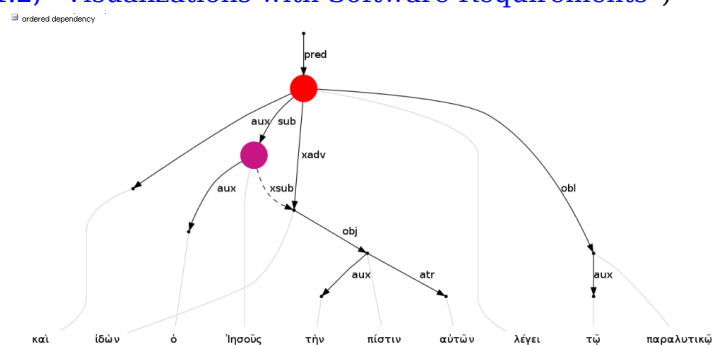
mmax (discourse)

Stellpass Wunder gibt es immer wieder ! Erst spielen die **Dallgower** Gemeindevertreter so statisch und verzagt wie die deutsche Abwehrreihe der Fußballkicker . Und dann kommt aus der Tiefe solch ein fulminanter Stellpass , von dem man hofft , dass **die Seeburger** oder Grotk-Glienicker **Mitspieler** ihn aufnehmen können . Ein Befreiungsschlag ist es allerdings nicht , weil es vorerst keine Gefahr fürs **Dallgower** Tor gab . **Die Seeburger** und **einige Grotk-Glienicker** haben den Ball erst zurückgespielt und dann um so drängender wieder gefordert . Nun sollen **sie** zeigen , wie **sie** die Chance verwerten . Eine Diskussion , wo künftig die Trainerkabine stehen soll , wäre in der jetzigen Spielsituation verheerend . Und eine Parallele zu den deutschen Grotten-Kickern gibt es immer noch . Auch wenn **die Spieler** aus den verschiedenen Vereinen zusammengewürfelt sind , **sie** müssen sich daran gewöhnen , dass **sie** nun in einer Mannschaft " Döbertitzer Heide " spielen . Und das heißt gemeinsam und nicht gegeneinander . Ermahnungen von der


- *arch_dependency* (dependency tree with labeled arches between tokens; requires SVG enabled browser; see [Section 3.4.2, "Visualizations with Software Requirements"](#))

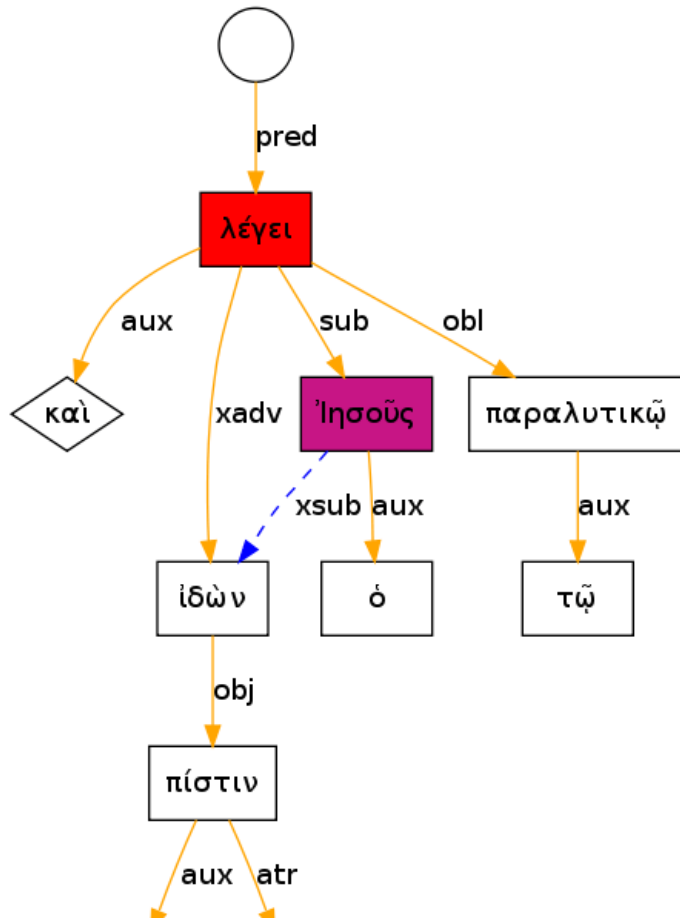


- *ordered_dependency* (arrow based dependency visualization for corpora with dependencies between non terminal nodes; requires GraphViz, see [Section 3.4.2, "Visualizations with Software Requirements"](#))

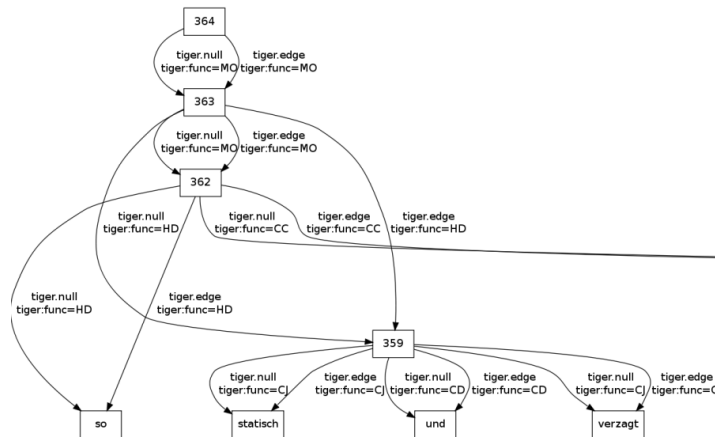


- *hierarchical_dependency* (unordered vertical tree of dependent tokens; requires GraphViz, see [Section 3.4.2, “Visualizations with Software Requirements”](#))

 hierarchical dependency



- *graph* (a debug view of the annotation graph; requires GraphViz, see 5.2)



- *video* (a linked video file)



- *audio* (a linked audio file)

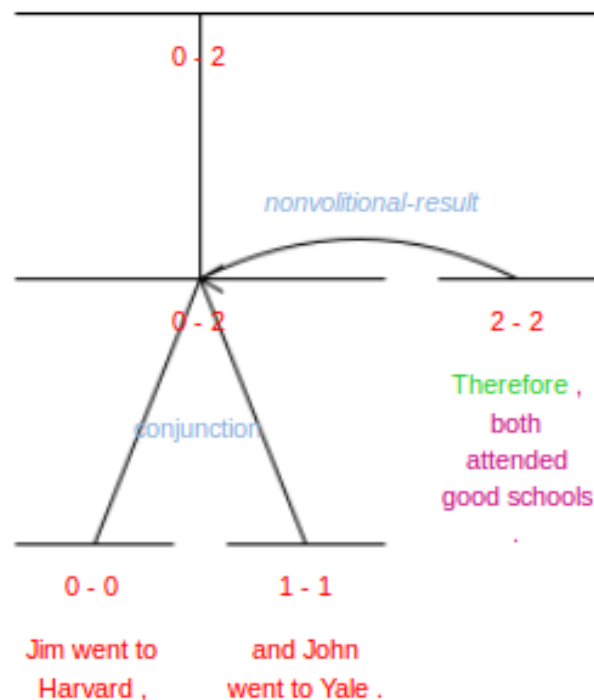


- *rst* - imitates the RST-diagrams from the RST-Tool ([RSTTool](http://www.wagsoft.com/RSTTool/) [http://www.wagsoft.com/RSTTool/])

Note, that the rst-xml-format models edges between nucleus and satellite the other way around, than in the common visualization of rst-graphs. So the dominance relation between nodes must be defined in AQL like that:

```
node & node & "Therefore" & #1 >rst[relname="nonvolitional-result"] #2 & #2
```

The AQL-Query above searches for a satellite which contains the token "Therefore" and dominates a node with the rst edge type "nonvolitional-result". Important is, that the first note dominates the second one, also the visualization shows it the other way around.



Further the `rst` relations are not modeled as pointing relations, so the `->` Operator obviously does not provide a result.

- *display_name* determines the heading that is shown for each visualizer in the interface
- *visibility* determines the visibility state of the visualizer. Valid values are:
 - *permanent* - the visualizer is shown and not closeable
 - *hidden* - the visualizer is hidden, but expandable
 - *visible* - the visualizer is visible and closeable
 - *preload* - behaviour is the same as *hidden*, but the visualizer starts to rendering in the background.
- *order* determines the order in which visualizers are rendered in the interface (low to high)
- *mappings* provides additional parameters for some visualizations:
 - *tree* - the annotation names to be displayed in non terminal nodes can be set e.g. using **node_key:cat** for an annotation called **cat** (the default), and similarly the edge labels using **edge_key:func** for an edge label called **func** (the default). Instructions are separated using semicolons.
 - *graph* - use **ns_all:true** to visualize the entire annotation graph. Specifying e.g. **node_ns:tiger** or **edge_ns:tiger** instead causes only nodes and edges of the namespace **tiger** to be visualized (i.e. only a subgraph of all annotations)
 - *grid* - it is possible to specify the order of annotation layers in each grid. Use **annos: anno_name1, anno_name2, anno_name3** to specify the order of annotation layers. If **anno:** is used, additional annotation layers not present in the list will not be visualized. If mappings is left empty, layers will be ordered alphabetically
 - *grid_tree* - specify the name of the annotation to be visualized in the grid with **node_key:name**. Note that all grid levels visualize the same annotation name at different hierarchical depths.
 - *arch_dependency* - if a annotation key is specified with **node_key:key**, all nodes which carry this annotation are searched for pointing relations and instead token the annotation value is used.

3.4.2. Visualizations with Software Requirements

Some ANNIS visualizers require additional software, depending on whether or not they render graphics as an image directly in Java or not. At present, three visualizations require an installation of the freely available software GraphViz (<http://www.graphviz.org/>): *ordered_dependency*, *hierarchical_dependency* and the general *graph* visualization. To use these, install GraphViz on the server (or your local machine for Kickstarter) and make sure it is available in your system path (check this by calling e.g. the program **dot** on the command line).

Another type of restriction is that some visualizers may use SVG (scalable vector graphics) instead of images, which mean the user's browser must be SVG capable (e.g. FireFox) or a plugin must be used (e.g. for Internet Explorer 8 or below). This is currently the case for the *arch_dependency* visualizer.

3.4.3. Changing maximal context size and possible context steps

The maximal context size of $\pm n$ tokens from each search result (for the KWIC view, but also for other visualization) can be set for the ANNIS service in the file `<service-home>/conf/annis-service.properties` Using the syntax, e.g. for a maximum context of 10 tokens:

```
annis.max-context=10
```