



User Guide – Version 2.2.1

(For the latest documentation see also:

<https://korpling.german.hu-berlin.de/p/projects/annis/>)

title: ANNIS User Guide
version: 2.2.1
date: 18 April 2012

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/d1/annis/>

Contents

1	Introduction.....	2
2	New Features in Version 2.2.1.....	2
3	Installing ANNIS2	3
3.1	Installing a Local Version (ANNIS Kickstarter)	3
3.2	Building and Installing an ANNIS Server	3
4	Running Queries in ANNIS2.....	6
4.1	The ANNIS2 Interface.....	6
4.2	Using the ANNIS2 Query Builder.....	8
4.3	Searching for Word Forms.....	9
4.4	Searching for Annotations	11
4.5	Searching using Regular Expressions.....	12
4.6	Searching for Trees	13
4.7	Searching for Pointing Relations – Coreference and Dependencies	14
4.8	Exporting Search Results.....	15
4.9	Complete List of Operators.....	18
5	Configuring Visualizations	20
5.1	Triggering Visualizations with the Resolver Table	20
5.2	Visualizations with Software Requirements.....	24
5.3	Changing maximal context size and possible context steps	24
6	Converting Corpora for ANNIS using SaltNPepper.....	25

1 Introduction

ANNIS2 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 ANNIS2 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 New Features in Version 2.2.1

Features:

- Subcorpus and document path for each hit can be shown in search result list
- Metadata coming from documents, subcorpora and top level corpora is sorted separately in the metadata display
- New grid-tree visualizer for a span visualization of hierarchical structures (e.g. topological fields in German; see Section 5.1)
- More intuitive search button position in GUI and other small GUI improvements
- Updated tutorial
- Faster import when large corpora are already in the system

Bugfixes:

- Support for JDK 7
- Fixed import bug which could disrupt queries of the sort >secedge m,n (see <https://bugs.launchpad.net/bugs/870108>)
- Fixed bugs in some forms of query disjunction with ‘|’
- Fixed resizing of visualizers in Chrome browsers
- Fixed escaping of characters in WEKA export

(For change logs of previous version see their respective distributions or user guides)

3 Installing ANNIS2

3.1 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 8.4 for your operating system from <http://www.postgresql.org/download/> and **make a note of the administrator password** you set during the installation. After installation, Postgres may automatically launch the Postgres 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-2.2.1-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](#) 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 ANNIS2” in this guide for some more example queries, or press the Tutorial button at the top left of the interface).

3.2 Building and Installing an ANNIS Server

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

1. Install a PostgreSQL server for your operating system from <http://www.postgresql.org/download/>
2. Install a web server such as [Tomcat](#) or [Jetty](#)
3. Make sure you have [JDK 6](#) and [Maven 2](#) (or install them if you don't)
4. If you're using Cygwin and Windows you will also need to install the "patch" program via the Cygwin package manager
5. Download and unzip [Annis-2.2.1.zip](#), then run the following commands (replacing the appropriate directories):

```
cd <unzipped source>/Annis-Service
mvn -DskipTests=true install
mvn -DskipTests=true assembly:assembly

tar xzvf target/annis-service-<version>-distribution.tar.gz -C <installation
directory>
```

6. Next initialize your ANNIS database (only the first time you use the system):
7. Set the environment variables (each time when starting up)

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

8. 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.

9. Now you can start the ANNIS service:

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

10. To get the Annis front-end running, first compile it:

```
cd <unzipped source>
mvn -DskipTests=true install
```

If no error occurs the war-file will be available under

<unzipped source>/Annis-web/target/Annis-web.war.

11. And configure your web server as described here:
<https://korpling.german.hu-berlin.de/p/projects/annis/wiki/Tomcat>

The latest instructions for compiling and installing the ANNIS Server can also be found at: <https://korpling.german.hu-berlin.de/p/projects/annis/wiki/Documentation>

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>

4 Running Queries in ANNIS2


4.1 The ANNIS2 Interface

The screenshot displays the ANNIS2 interface. On the left, the search form is highlighted with a red box. It contains an AnnisQL query: `tok & tok & #1 ->dep[func="OA"] #2 & cat="S" & #3 | #1 & node & #3 >secedge #4 | correction="correcting" | cat="S"`. Below the query is a table of corpora with columns for Name, Texts, and Tokens. The 'tiger1.dep' corpus is selected. On the right, the results window is highlighted with a blue box. It shows a video player at the top, followed by the search results for the query. The results include a list of dependencies and constituents for the sentence 'während 78 Prozent sich für Bush und vier Prozent für Clinton aussprechen'. Below this, there is a table of annotations for the phrase 'Die Vase auf dem Tisch ist größer als die Vase'.

The ANNIS2 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).

The Search Form

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.

The screenshot shows the Search Form window. It contains an AnnisQL query: `cat="S" & #1 >secedge[func="SB"] #2`. Below the query is a table of corpora with columns for Name, Texts, and Tokens. The 'DDB.AHD' corpus is selected. Below the table are search options: Context Left: 5, Context Right: 5, Results Per Page: 10.

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.

meta data		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;I (for dominance edges);dep.func (for dependency pointing relations);anaphor_antecedent (pointing relations)	Focus_newInf	Focus_newInf="nf-unsol" ↔
full_name	Potsdam Commentary Corpus (sample of 2 documents)	Inf-Stat	Inf-Stat="giv-active" ↔
language	German	NP	NP="NP" ↔
source	Project D1, SFB 632	PP	PP="PP" ↔
URL	link	Sent	Sent="s" ↔
version	2.0	Topic	Topic="ab" ↔
		ambiguity	ambiguity="not_ambig" ↔
		anaphor_type	anaphor_type="anaphor_nominal" ↔
		complex_np	complex_np="no" ↔
		dir_speech	dir_speech="text_level" ↔
		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.

The Result Window

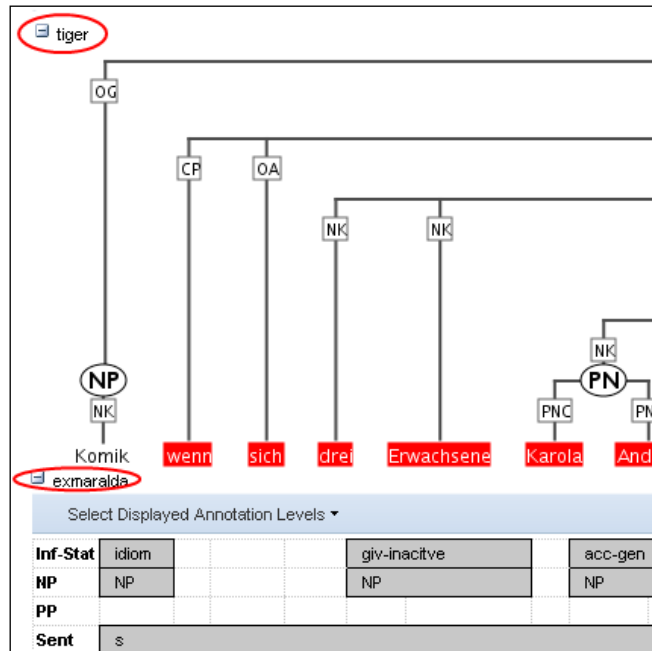
The result window shows search results in pages of 10 hits each by default (this can be changed in the Search Form). The toolbar at the top of the window allows you to navigate between these pages. The "Token Annotations" button on the toolbar allows you to toggle

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.

der	wie	eine	Mumie	auf	der	Bank
der	wie	ein	Mumie	auf	der	Bank
ART	KOKOM	ART	NN	APPR	ART	NN
n.Sg.Masc	--	Nom.Sg.Fem	Nom.Sg.Fem	--	Dat.Sg.Fem	Dat.Sg.Fem

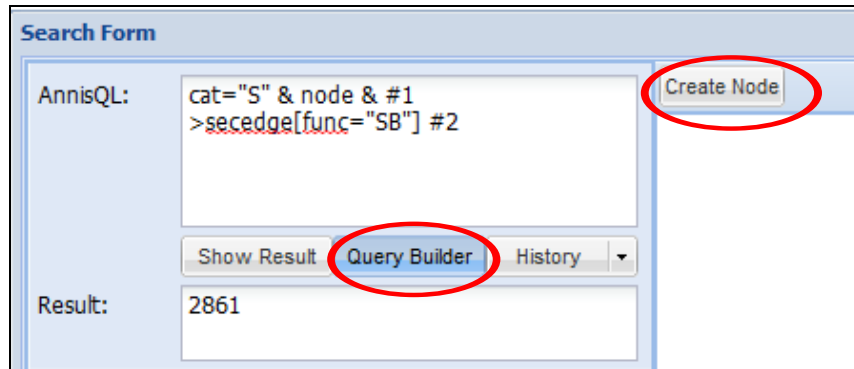
tiger:morph = Nom.Sg.Fem

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 to the right (circled in red).

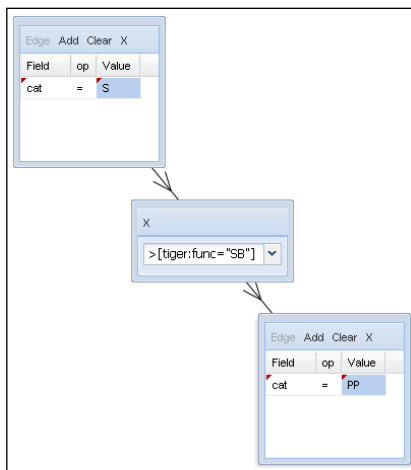
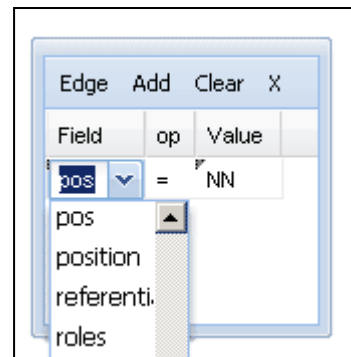


4.2 Using the ANNIS2 Query Builder

To open the graphical query builder, click on the Query Builder button on the Search Form (clicking the button again will close the Query Builder). 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 "Add" to specify an annotation value. The annotation name can be typed in or selected from a drop down list. The "Op[erator]" 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 "Add" multiple times. Clicking on "Clear" 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.

4.3 Searching for Word Forms

To search for word forms in ANNIS2, 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.

4.4 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.

4.5 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
```

4.6 Searching for Trees

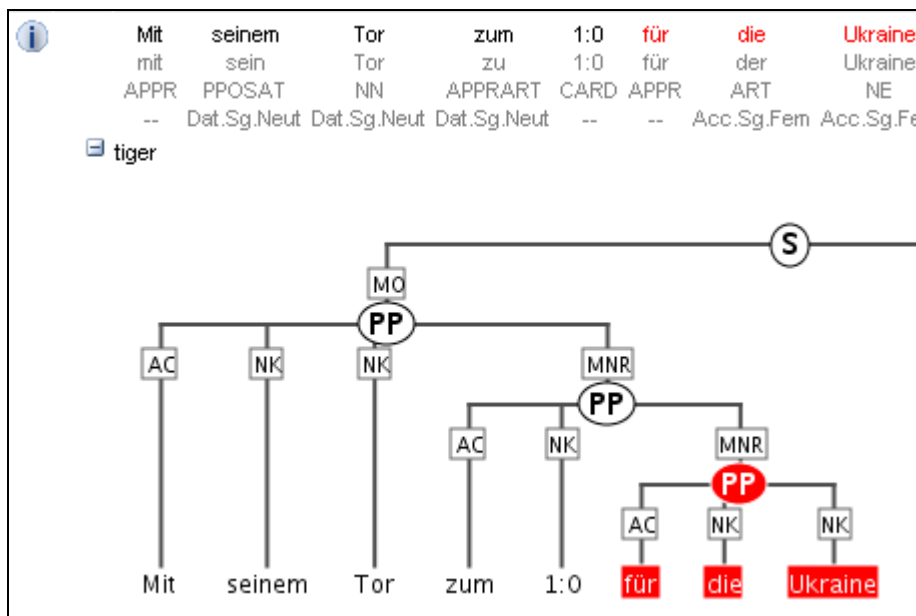
In corpora containing hierarchical structures, annotations such as syntax trees can be searched for by defining terminal or non-terminal node annotations and their values. A simple search for prepositional phrases in the small pcc2 demo corpus looks like this:

```
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.



Note that since the context is set to a number of tokens left and right of the search term, the tree for the whole sentence may not be retrieved. To do this, you may want to specifically search for the sentence dominating the PP. To do so, specify the sentence in another element and use the indirect dominance ($>^*$) operator:

```
cat="S" & cat="PP" & pos="NE" & #1 >^* #2 & #2 > #3
```

If the annotations in the corpus support it, you may also look for edge labels. Using the following query will find all adjunct modifiers of a VP, dominated by the VP node through an edge labeled MO. Since we do not know anything about the modifying node, whether it is a non-terminal node or a token, we simply use the node element as a placeholder. This element can match any node or annotation in the graph:

```
cat="VP" & node & #1 >[tiger:func="MO"] #2
```

It is also possible to negate the label of the dominance edge as in the following query:

```
cat="VP" & node & #1 >[tiger:func!="MO"] #2
```

which finds all VPs dominating a node with a label other than MO.

4.7 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
```

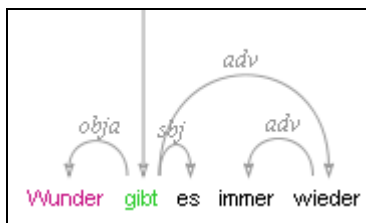
Using the pointing relation operator $->$ with the type `anaphor_antecedent`, the first `np_form`, which should be a personal pronoun (`pper`), is said to be the anaphor to its antecedent, the second `np_form`, which is definite (`defnp`). To see a visualization of the coreference relations, open the 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.

mmax (discourse)
 Steilpass Wunder gibt es immer wieder ! Erst spielen die Dalgower 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 Dalgower 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

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'.



4.8 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] Völlner der Weg zur Weltmeisterschaft
4. Reputation kämpfenden Mannschaft von Rudi [Völlner] 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.

4.9 Complete List of Operators

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

Operator	Description	Illustration	Notes
.	direct precedence	A B	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, <code>.n,m</code> can be used, e.g. <code>.3,4</code> - between 3 and 4 token distance
>	direct dominance	A B	A specific edge type may be specified, e.g.: <code>>secedge</code> to find secondary edges. Edges labels are specified in brackets, e.g. <code>>[func="OA"]</code> for an edge with the function 'object, accusative'
>*	indirect dominance	A ... B	For specific distance of dominance, <code>>n,m</code> can be used, e.g. <code>>3,4</code> - dominates with 3 to 4 edges distance
<code>_=_</code>	identical coverage	A B	Applies when two annotation cover the exact same span of tokens
<code>_i_</code>	inclusion	AAA B	Applies when one annotation covers a span identical to or larger than another
<code>_o_</code>	overlap	AAA BBB	For overlap only on the left or right side, use <code>_ol_</code> and <code>_or_</code> respectively
<code>_l_</code>	left aligned	AAA BB	Both elements span an area beginning with the same token
<code>_r_</code>	right aligned	AA BBB	Both elements span an area ending with the same token
<code>->LABEL</code>	labeled pointing relation	LABEL 	A labeled, directed relationship between two elements. Annotations can be specified with <code>->LABEL[annotation="VALUE"]</code>
<code>->LABEL *</code>	indirect pointing relation	LABEL LABEL  A ... B	An indirect labeled relationship between two elements. The length of the chain may be specified with <code>->LABEL n,m</code> for relation chains of length n to m

>@l	left-most child	$ \begin{array}{c} \mathbf{A} \\ / \quad \quad \backslash \\ \mathbf{B} \quad x \quad y \end{array} $	
>@r	right-most child	$ \begin{array}{c} \mathbf{A} \\ / \quad \quad \backslash \\ x \quad y \quad \mathbf{B} \end{array} $	
\$	Common parent node	$ \begin{array}{c} x \\ / \quad \backslash \\ \mathbf{A} \quad \mathbf{B} \end{array} $	
\$*	Common ancestor node	$ \begin{array}{c} x \\ \\ \dots \\ / \quad \backslash \\ \mathbf{A} \quad \mathbf{B} \end{array} $	
#x:arity=n	Arity	$ \begin{array}{c} x \\ / \quad \quad \backslash \\ \mathbf{1} \quad \dots \quad \mathbf{n} \end{array} $	Specifies the amount of directly dominated children that the searched node has
#x:length=n	Length	$ \begin{array}{c} x \\ \dots \\ / \quad \backslash \\ \mathbf{1} \quad \dots \quad \mathbf{n} \end{array} $	Specifies the length of the span of tokens covered by the node
#x:root	Root	$ \begin{array}{c} \mathbf{x} \\ \dots \\ / \quad \backslash \\ \mathbf{1} \quad \dots \quad \mathbf{n} \end{array} $	node x is the root of a subgraph (i.e. it is not dominated by any node)

5 Configuring Visualizations

5.1 Triggering Visualizations with the Resolver Table

By default, ANNIS2 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 8.4 > 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:

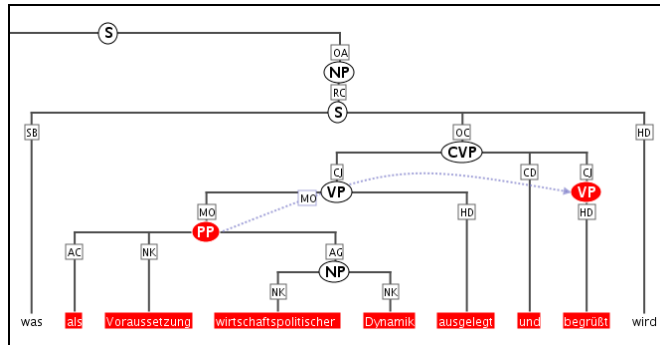
	id [PK] serial	corpus character var	version character var	namespace character var	element character var	vis_type character var	display_name character var	order numeric	mappings character var
1	1			tiger	node	tree	tree	101	
2	2			exmaralda	node	grid	exmaralda	102	
3	3			mmax	node	grid	mmax	103	
4	4			mmax	edge	discourse	coref	104	
5	5			urml	node	old_grid	urml	105	
6	6			external		file	external file	106	
7	7					paula	paula	107	
8	8					paula_text	paula text	108	
9	10	b3.parses.1		bitpar	node	tree	bitpar	1	
10	11	b3.parses.1		lingenio	node	tree	lingenio	2	
11	12	parallel_tree_		tiger-de	node	tree	Syntax (German	1	
12	13	parallel_tree_		tiger-en	node	tree	Syntax (English	2	
13	14	b3.parses.1				discourse	Whole Text	4	
14	15	SMULTRON_E		german	node	tree	Syntax (German	1	
15	16	SMULTRON_E		english	node	tree	Syntax (English	2	
*									

Resolver table (resolver_vis_map)

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

- *corpus* determines the corpora for which the instruction is valid (null values apply to all corpora)

- *namespace* specifies relevant namespace which triggers the visualization
- *element* determines if a node or an edge should carry the relevant annotation for triggering the visualization
- *vis_type* determines the visualizer module used and is one of:
 - *tree* (constituent syntax tree)



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

exmaralda									
Select Displayed Annotation Levels ▾									
Focus_newInf									
nf-unsol									
Inf-Stat	acc-gen								giv-active
NP	NP								NP
PP	PP								exmaralda:Inf-Stat = giv-active
Sent	s								
Topic	fs								ab
tok	die	Ukraine	stürzte	der	1,62	Meter	große	Gennadi	Subow

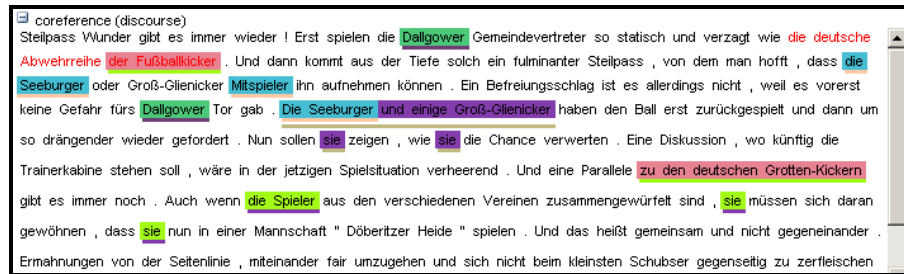
- *grid_tree* (a grid visualizing hierarchical tree annotations as ordered grid layers; note that all layers represent the same annotation name at different hierarchical depths, marked level:0,1,2,... etc. on the left)

topo (grid)						
level: 0	TOP					
level: 1	VF					
level: 2	C		C	MF	VC	
tok	Daß	und	wie	Demokratie	funktionieren	kann

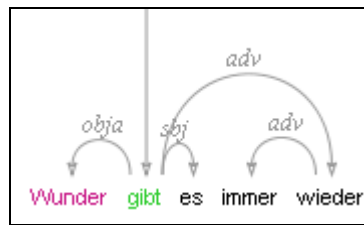
- *old_grid* (deprecated version of *grid*)

Select Displayed Annotation Levels ▾									
Focus_newInfSeg									
nf-unsol									
nf-unsol									
Inf-StatSeg	acc-gen								acc-gen
NP	NP								NP
NP	NP								exmaralda:Inf-Stat = giv-active
PP	PP								
Topic	fs								ab
Sent	s								
tok	die	Ukraine	stürzte	der	1,62	Meter	große	Gennadi	Subow

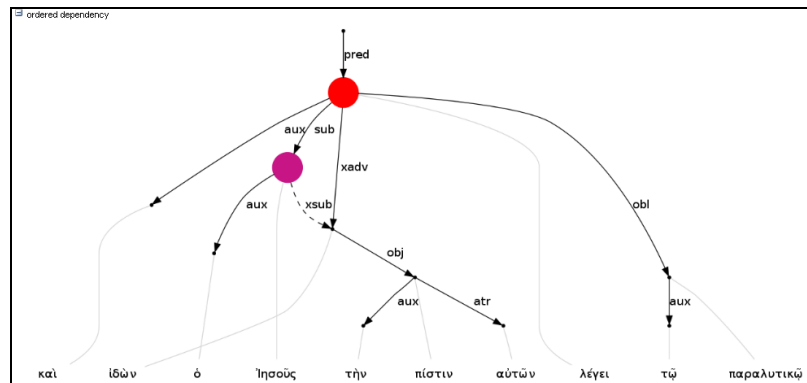
- *discourse* (a view of the entire text of a document, possibly with interactive coreference links. It is possible to use this visualization to view entire texts even if you do not have coreference annotations)



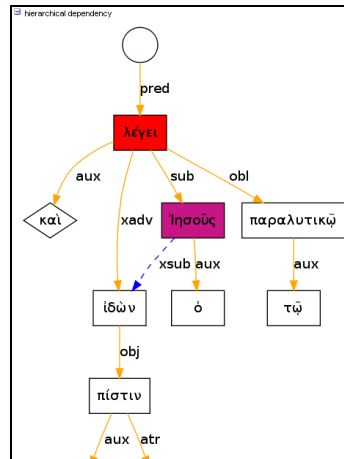
- *arch_dependency* (dependency tree with labeled arches between tokens; requires SVG enabled browser, see 5.2)



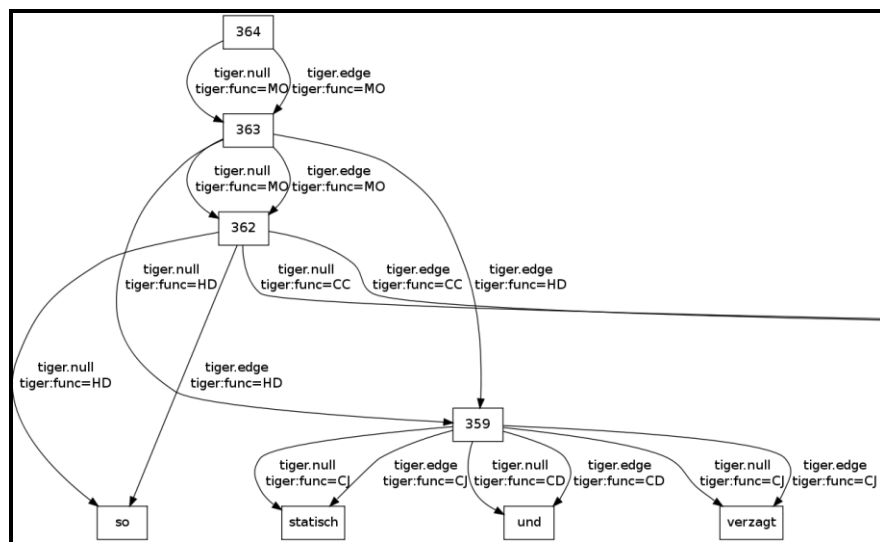
- *ordered_dependency* (arrow based dependency visualization for corpora with dependencies between non terminal nodes; requires GraphViz, see 5.2)



- *hierarchical_dependency* (unordered vertical tree of dependent tokens; requires GraphViz, see 5.2)



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



- *file* (a linked multimedia file)



The additional system internal debug views *paula* and *paula_text* deliver an XML representation of hits and entire texts respectively.

- *display_name* determines the heading that is shown for each visualizer in the interface
- *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 or 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.
- the field *version* is reserved for future development.

5.2 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.

5.3 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
```

To configure which steps are actually shown in the front-end (up to the maximum allowed by the service above) and the default context selected on login, configure the file:

```
<annis-web-home->/javascript/annis/config.js
```

With the syntax:

```
var search_context = [[0], [1], [2], [5], [10]];
var search_context_default = 5;
```

6 Converting Corpora for ANNIS using SaltNPepper

ANNIS2 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/>