



User Guide – Version 3.3.5

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

title: ANNIS User Guide
ANNIS version: 3.3.5
guide version: 1.0.0
date: 2015-08-26

author: Amir Zeldes
e-mail: annis-admin@ling.uni-potsdam.de
homepage: <http://annis-tools.org>

Contents

1	Introduction.....	2
2	New Features in Version 3.3.5.....	2
3	Installing ANNIS	3
3.1	Installing a Local Version (ANNIS Kickstarter)	3
3.2	Installing an ANNIS Server.....	4
4	Querying Corpora in ANNIS	6
4.1	The ANNIS Interface.....	6
4.2	Using the ANNIS Query Builder.....	9
4.3	Searching for Word Forms.....	10
4.4	Searching for Annotations	10
4.5	Searching using Regular Expressions.....	12
4.6	Searching for Trees	13
4.7	Searching for Pointing Relations – Coreference and Dependencies	16
4.8	Exporting Search Results.....	16
4.9	Frequency Analysis.....	18
4.10	Complete List of Operators.....	23
5	Configuring Visualizations	25
5.1	Triggering Visualizations with the Resolver Table	25
5.2	Visualizations with Software Requirements	32
5.3	Changing Maximal Context Size, Context Steps and Result Page Sizes	32
5.4	Configuring the Document Browser.....	33
5.5	Configuring Right-to-Left Visualizations.....	35
6	Importing and Configuring Corpora	36
6.1	Converting Corpora for ANNIS using SaltNPepper.....	36
6.2	Importing Corpora in the relANNIS format	36
6.3	Configuring Settings for a Corpus	36
6.4	Multiple Instances of the Interface	37
6.5	User management.....	39
6.6	The administration web interface.....	41

1 Introduction

ANNIS is an open source, browser-based search and visualization architecture for multi-layer corpora, developed at Humboldt-Universität zu Berlin, Georgetown University and Potsdam University. 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, rhetorical structure 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 New Features in Series 3.3.X

- Much faster conversion and import using new relANNIS version 3.3 (corpora in relANNIS 3.2 are still supported)
- Syntax highlighting for AQL queries
- Frequency analysis using metadata
- Result ordering (random, ascending, descending) and limiting
- HTML visualizer can be embedded in external pages
- Support for searching by document name
- New demo corpus and tutorial (using the GUM corpus)
- Numerous bug fixes

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

3 Installing ANNIS

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 9.4 (or 9.2 or above, which are all supported) 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, 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 PostgreSQL installer tells you to.

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. Download and unzip the ANNIS Kickstarter ZIP-file from the ANNIS website.
3. Start AnnisKickstarter.bat if you're using Windows, AnnisKickstarter.cmd on Mac 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 (after java or javaw in the .sh or .bat script respectively):

```
-Xss1024k -Xmx1024m
```

(To accelerate searches it is also possible to give the PostgreSQL database more memory, see 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 PostgreSQL administrator password from step 1. If you are upgrading from version 3.0.1 of ANNIS Kickstarter or higher, you

will be given the option to reimport your corpora, assuming they can still be found at the paths from which they were originally imported.

5. Download and unzip the GUM demo corpus from the ANNIS website: <http://annis-tools.org/corpora.html>.
6. Press “Import Corpus” and navigate to the directory containing the directory GUM_relAnnis/. Select this directory (but do not go into it) and press OK.
7. Once import is complete, press “Launch Annis frontend” test the corpus (click on one of the example queries displayed on the screen, or try selecting the GUM corpus, typing pos="NN" in the AnnisQL box at the top left and clicking “Show Result”. See the section “Querying and importing corpora in ANNIS” in this guide for some more example queries, or press the Tutorial button in the Help/Examples tab of the interface for more information).

3.2 Installing an ANNIS Server

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

1. Download and install PostgreSQL 9.4 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, 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 PostgreSQL installer tells you to.

Note: Under Linux, you might have to set the PostgreSQL password manually.

2. Install a Java Servlet Container ("Java web server") such as Tomcat or Jetty
3. Make sure you have installed JDK 6 or JDK 7
4. Download the ANNIS service distribution file `annis-service-<version>-distribution.tar.gz` from the website and then unzip the downloaded file:

```
tar xzvf annis-service-<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 <new user password> -P <postgres superuser password>
```

You can omit the PostgreSQL administrator password option (-P). Then the database and user must already exist. E.g. you should execute the following as PostgreSQL administrator:

```
CREATE LANGUAGE plpgsql; -- ignore the error if the
language is already installed

CREATE USER myuser PASSWORD 'mypassword';

CREATE DATABASE mydb OWNER myuser ENCODING 'UTF8';
```

Now you can import some corpora:

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

Warning

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.

7. Now you can start the ANNIS service:

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

8. To get the ANNIS front-end running, first download annis-gui-<version>.war from our website and deploy it to your Java servlet container (this depends on the servlet container you use).

Note

We also **strongly** recommend reconfiguring the PostgreSQL server's default settings as described [here](#). For more information on configuration options see the developers' documentation at: <http://korpling.github.io/ANNIS/>.

4 Querying Corpora in ANNIS

The screenshot displays the ANNIS (Annotation-based Network for Natural Language) interface. On the left, a sidebar contains a search form with a query: `"Pharmakonzern" | pos=V.FIN / ->dep[func="subj"] "Jugendliche" & cat="S" & #4 >secedge #3`. Below the search form is a table of corpora, with 'pcc2' selected. The main workspace on the right shows the results for the selected corpus. It includes a video player, a list of tokens with their part-of-speech tags, a dependency graph (arcs), an information structure grid, a constituent tree, and a coreference graph. The text being analyzed is: `darüber streiten , was Jugendliche wollen und brauchen , ohne auf`. The interface also shows a list of corpora on the left, with 'pcc2' selected.


4.1 The ANNIS Interface

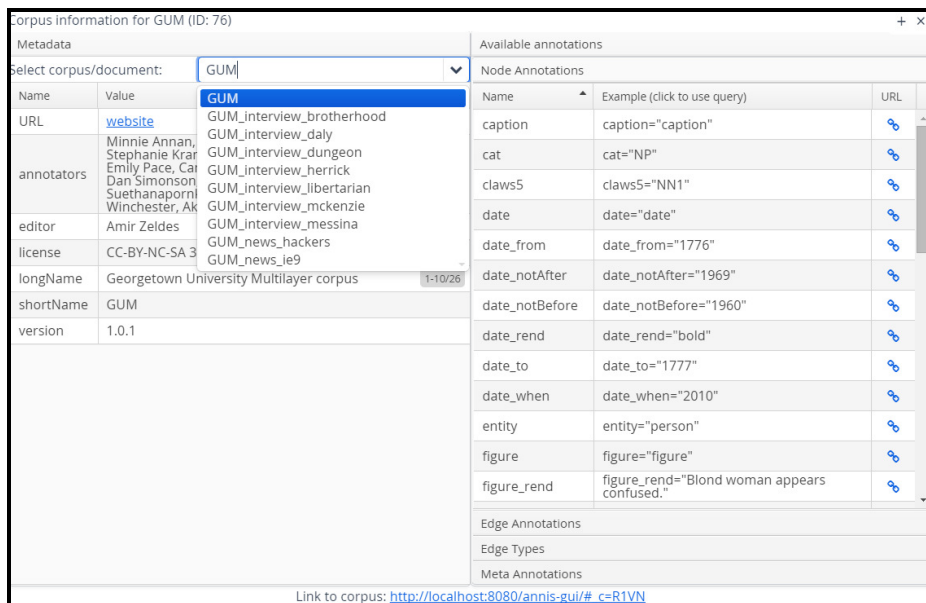
The ANNIS interface is comprised of two main areas: the search form on the left and the tabbed workspace on the right in the picture above. The search form may be hidden to provide more space by clicking on the “hide sidebar” button with the lines and arrow at the top left corner of the interface shown above.

If you have imported corpora with example queries (the demo corpus GUM includes some, but see Section 6.3 on how to generate your own), then you will see some clickable example queries in the Help/Examples tab of the workspace on the right. You can always return to these by clicking the Help/Examples tab, and filter example queries for specific corpora by selecting each corpus. If no example queries are in the database, the interface will show you the ANNIS tutorial, which also uses the GUM corpus as an example. You can switch between the tutorial and the examples in the Help/Examples tab. It is recommended to import the GUM demo corpus when working with the system for the first time. For more information on generating example queries, see Section 6.3.














The Search Form

The Search Form, on the left of the interface window shown above, contains a list of all corpora available to the current user. If you are not logged in, you will only have access to the corpora that the user "anonymous" is allowed to see (in the local Kickstarter version, all corpora are available by default). Additionally, it is possible to filter the visible corpora by group using the selection box above the list (by default showing 'all', as in the image above). For user right management and **corpus group configuration** in the ANNIS server version, see Sections 6.3 and 6.4. If the list is very full, it may also be useful to type some text into the **Filter box** above the corpus list, which causes only corpora whose name contains that text to be shown.

Using the checkboxes on the left of each corpus, it is possible to select which corpora should be searched in (hold down 'control' to select multiple corpora simultaneously). The list also gives the number of texts and tokens in each available corpus. Pressing the  button next to a corpus in the list will open the **corpus explorer** window (see picture below). The left side of this window shows metadata for the entire corpus, and using the box "select corpus/document" also allows you to browse the metadata for individual documents within the corpus. On the right hand side, a list of available annotations and simple example queries are shown. The list has four parts for node annotations (referring to elements covering some text in the corpus), edge types (the types of edges that apply between such elements), edge annotations (referring to those edges) and meta-data annotations.





Metadata	
Select corpus/document: GUM	
Name	GUM
URL	website
annotators	Minnie Annan, Stephanie Kraw, Emily Pace, Cai Dan Simonson, Suethanaporn Winchester, Ak
editor	Amir Zeldes
license	CC-BY-NC-SA 3.0
longName	Georgetown University Multilayer corpus
shortName	GUM
version	1.0.1

Available annotations		
Node Annotations		
Name	Example (click to use query)	URL
caption	caption="caption"	
cat	cat="NP"	
claws5	claws5="NN1"	
date	date="date"	
date_from	date_from="1776"	
date_notAfter	date_notAfter="1969"	
date_notBefore	date_notBefore="1960"	
date_rend	date_rend="bold"	
date_to	date_to="1777"	
date_when	date_when="2010"	
entity	entity="person"	
figure	figure="figure"	
figure_rend	figure_rend="Blond woman appears confused."	
Edge Annotations		
Edge Types		
Meta Annotations		

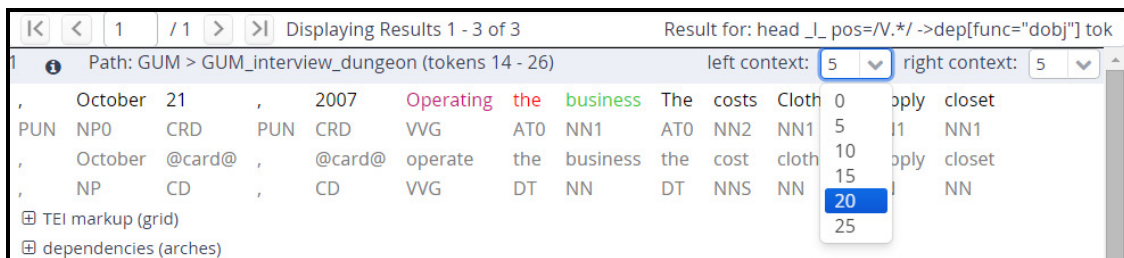
Link to corpus: http://localhost:8080/annis-gui/#_c=R1VN

Clicking on a query will copy it to the AQL (ANNIS Query Language) box at the top of the form and pressing the link icon will give you a citation link that can be used to access the query from any browser. The queries in this window are rather simple, e.g. an annotation name and some frequent value for that annotation. To create more complex, user defined example queries, see Section 6.3.

Next to the  button, you will find the **document browser** button: . Clicking on this button will open a list of all documents in the corpus as shown below, and allow you to view a plain text output of each entire document. It is also possible to add and sort by specific metadata fields, as well as to define other document visualizations beyond or instead of plain text. For more information on configuring and enabling/disabling the corpus browser, see Section 5.4.

The AQL field at the top of the search form is used for inputting queries manually (see the tutorials on the ANNIS Query Language below). Once a query has been entered, pressing the "Search" button (or using the shortcut ctrl+Enter) will retrieve the number of matching positions in the selected corpora, as well as the number of documents they come from, in the Status box. On the right side of the interface, the Query Result tab will display the first set of matches. Queries from the current session are saved in the **query history** and can be accessed using the drop down list underneath the AQL field. Pressing the history button also allows you to open an extended history list with even more of your recent queries.

The context size surrounding the matching expressions in the result list can be changed in the "Search Options" tab of the search form, using the boxes "Left Context" and "Right Context". By default, context 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. To **change the maximum context** for all or for specific corpora, see the information in Section 5.3. It is also possible to **alter context for individual search results**, up to the maximum allowed by the corpus, by using the left and right context drop-downs at the top right of each search result, as shown below.

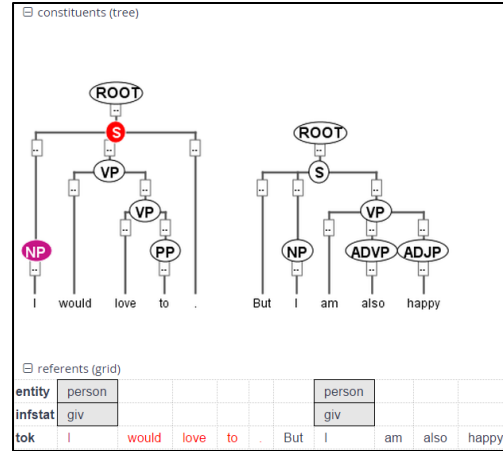


The Result Window

The result window shows search results in pages of 10 hits each by default (this can be changed in the Search Form under Search Options). To change the available **hits per page**, see Section 5.3. The toolbar at the top of the window allows you to navigate between the result 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 you convenience. The query is also repeated at the top right of the window for your reference, and is represented in a masked form in the browser's URL. To send a query by e-mail or cite it in a paper or on a web page you can simply copy the URL from your browser.

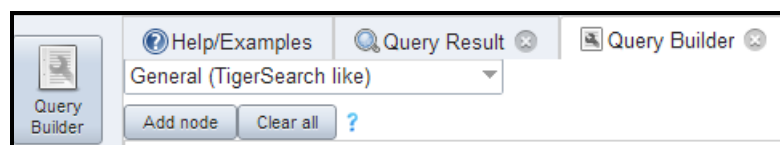
1	201	363-0555	.	See	Fort	Lee	Historic	Park	,	a	reconstruction	of	a
CRD	CRD	CRD	PUN	VVB	NP0	NP0	AJ0	NN1	PUN	AT0	NN1	PRF	AT0
1	@card@	@card@	.	see	Fort	Lee	historic	Park	,	a	reconstruction	of	a
CD	CD	CD	SENT	VV	NP	NP	NP	NP	,	DT	NN	IN	DT

By default, the result list itself initially shows only a KWIC (key word in context) concordance of matching positions in the selected corpora, though other default visualizations can be chosen (e.g. a grid for dialogue corpora, see Section 5). The region matching the query is marked in color and the context in black on either side. If the query contains multiple annotations, they will be highlighted in different colors within the search result. 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. 'referents' and 'constituents' for the annotations in the grid and tree views in the picture.

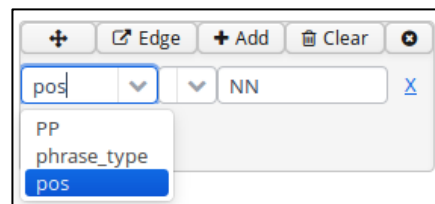


4.2 Using the ANNIS Query Builder

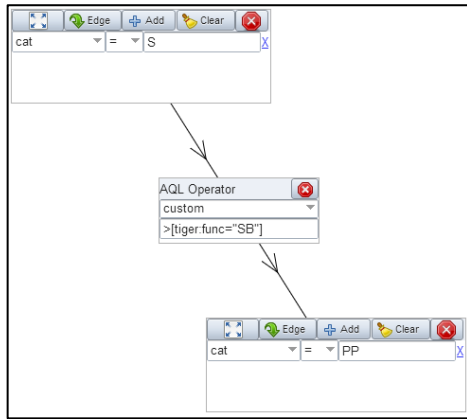
To open the graphical query builder, click on the Query Builder button on the Search Form (clicking the button again will close the Query Builder). On the left-hand side of the toolbar at the top of the query builder canvas, you will see the Create Node button. Use this button to define nodes to be searched for (tokens, non-terminal nodes or annotations). Creating nodes and modifying them on the canvas will immediately update the AQL 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 the +Add button to specify an annotation value. The annotation name can be typed in 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), and negated versions of these operators with a '!'. 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 the "Clear" broom



will delete the values in the node. To search for word forms, simply leave the default field name 'tok' on the left and type directly on the right side of the node. 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 create more than one node. Then click on the “Edge” button of the source node, and then click the word "Dock", 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 "tiger:func" label in the image to the left. Note

that the node clicked on first (where the arrow 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. You can also move and reposition nodes for your convenience by clicking on the square button at the top left of each node and dragging the nodes across the canvas.

4.3 Searching for Word Forms

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

"do"

Note that the search is **case sensitive**, so it will not find cases of capitalized 'Do', 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 (|):

"do" | "Do"

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

/[Dd]o/

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:

"do" & "n't" & #1 . #2

The expression `#1 . #2` signifies that the first element ("do") precedes the second element ("n't"). Alternatively, you can also place the operator directly between the search elements as a **shortcut**. The following shortcut query is equivalent to the one above:

```
"do" . "n't"
```

For **indirect precedence** (where other tokens may stand between the search terms), use the `.*` operator:

```
/[Dd]o/ & "n't" & "any" & #1 . #2 & #2 .* #3
```

OR using shortcuts:

```
/[Dd]o/ . "n't" .* "any"
```

The queries above find sequences beginning with the token "Do" or "do", followed directly by "n't", which must be followed either directly or indirectly (`.*`) by "any". A range of allowed distances can also be specified numerically as follows:

```
/[Nn]ot/ & "all" & #1 .1,5 #2
```

OR:

```
/[Nn]ot/ .1,5 "all"
```

Meaning the two words "not" and "all" 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.¹

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):

```
"want" & "to" & #1 .1,5 #2 & meta::type="interview"
```

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.

¹ If your corpus contains multiple segmentations, such as subtokens, morphemes or syllables, data from multiple overlapping speakers, or larger segmentation units (lines, sentences), it is also possible to query for precedence within *n* segmentation units with `#1 .unit_name,1,2 #2`. See the ANNIS Multiple Segmentation Corpora Guide for more details.

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 verb *be* in the GUM corpus, simply select the GUM corpus and enter:

```
lemma="be"
```

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

```
lemma!="be"
```

OR:

```
tok!="be"
```

Metadata can also be negated similarly:

```
lemma="be" & meta::type!="interview"
```

To only find finite forms of a verb in GUM, use the part-of-speech (`pos`) annotation concurrently with `lemma`, and specify that both the `lemma` and `pos` should apply to the same element. For example for inflected forms of the verb *give*:

```
lemma="give" & pos=/VV.+/ & #1 _=_ #2
```

OR (using a shortcut):

```
lemma="give" _=_ pos=/VV.+/
```

The regular expression `/VV.+/` means a part of speech that begins with `VV` (verb), but has additional characters (`.+`), such as for past tense (`VVD`) or gerund (`VVG`). 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 GUM, the annotation `entity` signifies the entity type of a discourse referent. This annotation can also apply to phrases longer than one token. The following query finds spans containing a discourse referent who is a person:

```
entity="person"
```

If the corpus contains more than one annotation type named `entity`, a namespace may be added to disambiguate these annotations (for example, the `entity` annotation in the GUM corpus has the namespace `ref:`, so we can search for `ref:entity="person"`). The namespace may always be dropped, but if there are multiple annotations with the same name but different namespaces, dropping the namespace will find all of those annotations. If you drop the value of the annotation, you can also search for any corpus positions that have that annotation, without constraining the value. For example, the following query finds all annotated entities in the GUM corpus, whether or not they are a person:

```
entity
```

In order to view the span of tokens to which the `entity` annotation applies, enter the query and click on "Search", then open the *referents* layer 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'):

```
head & infstat="new" & #1 _i_ #2
```

OR (using a shortcut):

```
head _i_ infstat="new"
```

This query finds information structurally new discourse referents (`infstat="new"`) contained within headings (`head`).

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 [here](#) 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=/ca./
```

This finds word forms such as "cat", "can", "car", "cap" etc. It is also possible to make characters optional by following them with a **question mark** (`?`). The following example finds cases of "car" and "cart", since the "t" is optional:

```
tok=/cart?/
```

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 "o", "of", and "off" (since the asterisk means zero or more times the preceding "f"), while the second finds "of" and "off", since at least one "f" must be found:

```
tok=/of*/
```

```
tok=/of+/
```

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 "VV", corresponding to all forms of lexical verbs (the auxiliaries "be" and "have" are tagged VB... and VH... respectively). The string "VV" means that the result must begin with "VV", the period stands for any character, and the asterisk means that 'any character' can be repeated zero or more time, as above.

```
pos=/VV.* /
```

This finds both finite verbs ("VVZ", "VVP", "VVD") and non-finite ones ("VV") or gerunds ("VVG"). 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 symbols**. The first example below finds either "of" or "on" (i.e. "o" followed by either "f" or "n") while the second example finds lemma annotations that are either "be" or "have".

```
tok=/o[nf] /
```

```
lemma=/ (be|have) /
```

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](#)) where a lexical verb dominates another token with a dependency edge not containing 'obj', we can use a wildcard to rule out all edges labels containing those letters. This will give us all non-object dependants of lexical verbs:

```
pos=/VV.* / & tok & #1 ->dep[func!=/*obj.* /] #2
```

OR (using a shortcut):

```
pos=/VV.* / ->dep[func!=/*obj.* /] tok
```

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, functional dependencies and their values (for dependencies see [Searching for Pointing Relations](#)). A simple search for prepositional phrases in the GUM corpus looks like this:

```
const:cat="PP"
```

If the corpus contains no more than one annotation called `cat`, the optional namespace, in this case `const:`, may be dropped. This finds all PP nodes in the corpus. You can also search for the NP being dominated by the PP like this:

```
cat="PP" & cat="NP" & #1 > #2
```

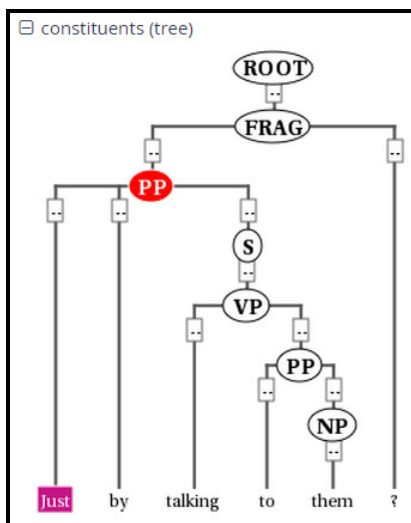
OR (using a shortcut):

```
cat="PP" > cat="NP"
```

To find all PP nodes directly dominating an adverb, you can combine a search for syntactic category and part-of-speech (pos) values (in this case "RB" for adverb). The query below gives the shortcut form:

```
cat="PP" > pos="RB"
```

The operator `>` signifies **direct dominance**, which must hold between the first and the second element. Once the Query Result tab is shown you may open the "constituents" annotation layer 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, though you can change the amount of tokens at the top of each search result, or for all search results in the Search Options tab. To make sure that the whole clause is always included, you may want to specifically search for the clause or sentence dominating the PP. To do so, specify the sentence in another element and use the **indirect dominance** (`>*`) operator:

```
cat="ROOT" >* cat="PP" > pos="RB"
```

If the annotations in the corpus support it, you may also look for edge labels. Using the following query will find all adverbial modifier NPs, dominated by some node through an edge labeled ADV. Since we do not know anything about the modified node, we simply use the `node` element as a place holder. This element can match any node or annotation in the graph:

```
node >[const:func="ADV"] cat="NP"
```

Again, the namespace `const:` is optional and only important if there are multiple 'func' annotations. It is also possible to negate the label of the dominance edge as in the following query:

```
cat >[func!="TMP"] cat
```

which finds all syntactic categories (value unspecified) dominating another syntactic category with a label other than "TMP".

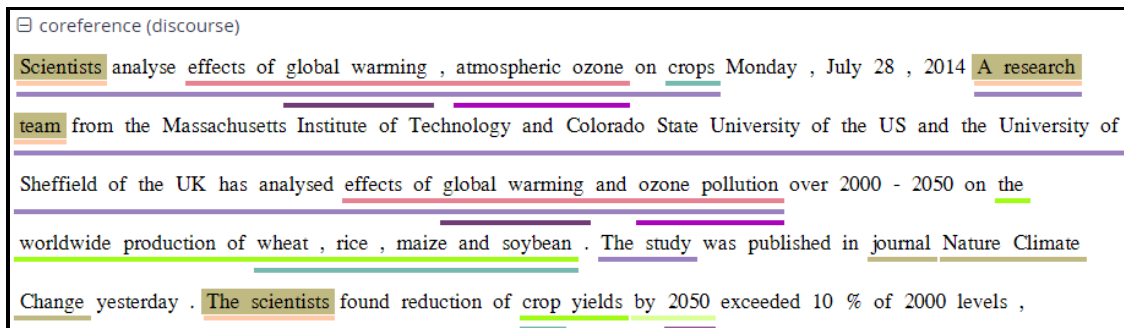
4.7 Searching for Pointing Relations – Coreference and Dependencies

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 GUM corpus, elements in the `ref:` namespace may point to each other to express coreference or anaphoric relations. The following query searches for two `entity` annotations, which specify whether a discourse referent is a person, or an animal, a location, an object, an abstract etc.

```
entity="person" &  
entity!="person" &  
#1 ->coref #2
```

Using the pointing relation operator `->` with the type `coref`, the first `entity`, which should be a person, is said to be coreferent with its antecedent, the second `entity`, which is not a person. In practice, this will usually occur due to "bridging", where something like a whole (e.g. an organization such as a 'research team') implies the existence of its part (e.g. persons, such as 'the scientists'). To see a visualization of the coreference

relations, open the coreference annotation layer in the GUM 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.



The pointing relation operator can also search for longer chains of coreference, using the asterisk extension shown below:

```
entity="organization" ->coref*
entity="person" &
```

This finds all organizations that point back to a person at any point along the preceding coreference chain. It is also possible to specify annotations of pointing relations, which for coreference in the GUM corpus mark what kind of coreference is used: anaphoric, cataphoric, lexical coreference, apposition, or bridging. To find appositions of place entities, use the following query:

```
entity="place" ->coref[type="appos"] entity="place"
```

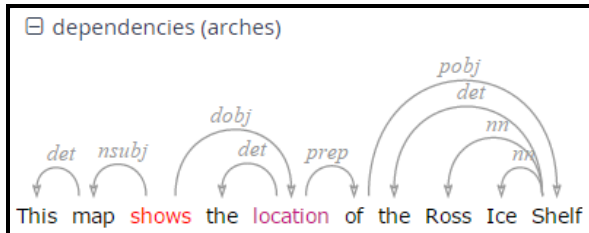
Another way to use pointing relations is found in syntactic dependency trees. The queries in this case can use both pointing relation types and annotations too, as in the following query:

```
pos=/VV[PZ]/ & tok & #1 ->dep[func="dobj"] #2
```

OR (using a shortcut):

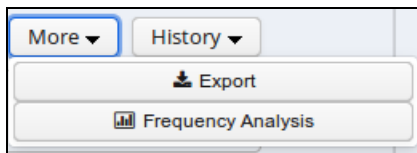
```
pos=/VV[PZ]/ ->dep[func="dobj"] tok
```

This query searches for a present tense lexical verb (with the part-of-speech VVZ or VVP) and a token, with a pointing relation of the type 'dep' (for dependency) between the two, annotated with 'func="dobj"' (the function 'direct object'). The result can be viewed with the arch dependency visualizer, which shows the verb 'shows' and its object 'location'.



4.8 Exporting Search Results

To export search results, open the menu "More" between the Search and History buttons and select "Export":



Enter the query whose results you want to export as usual in the AQL box. Note that you **do not need to carry out the query first**. You can enter the query and export without pressing Search before. Several exporter modules can be selected from the Export tab shown below.

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:

0. of the International Brotherhood of [Magicians] Wednesday , October 9 ,

```

1. Magic Month in the United [States] . Wikinews spoke with William
2. of the International Brotherhood of [Magicians] , about the current state
3. - " Scarne on Card [Tricks] " and " Scarne on
4. and " Scarne on Magic [Tricks] " . That started me

```

The TextExporter adds all annotations of each token separated by slashes (e.g. dogs/NNS/dog for a token dogs annotated with a part-of-speech NNS 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` in the ‘Parameters’ box). The user can specify annotation layers to be exported in the additional ‘Annotation Keys’ box, and annotation names should be separated by comas, as in the image above. Metadata annotations can also be exported by entering “metakeys=” and a list of comma separated metadata names in the Parameters box. If nothing is specified, all available annotations and no metadata will be exported. Multiple options are separated by a semicolon, e.g. the Parameters `metakeys=type,docname;numbers=false`. An example output with token numbers and the part of speech (pos) and syntactic category annotations looks as follows.

```

0.      tok      of the International Brotherhood of Magicians Wednesday
      pos      IN[1-1] DT[2-2] NP[3-3] NP[4-4] IN[5-5] NPS[6-6] NP[7-7]
      cat      S[1-6] VP[1-6] NP[1-6] PP[1-6] NP[2-4] PP[5-6] NP[6-6] NP[7-12]

```

Meaning that the annotation `cat="NP"` applies to tokens 1-6 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_span string
@attribute #1_anno_const:cat string
@attribute #2_id string
@attribute #2_span string

```

```

@attribute #2_anno_GUM:claws5 string
@attribute #2_anno_GUM:lemma string
@attribute #2_anno_GUM:pos string

@data

'11318611','the current state','NP','11318616','current','AJ0','current','JJ'
'11318686','magic','NP','11318688','magic','AJ0','magic','JJ'
'11318757','some basic tricks','NP','11318760','basic','AJ0','basic','JJ'

```

The export shows the properties of an NP node dominating a token with the part-of-speech JJ. Since the token also has other attributes, such as the lemma and part of speech tags, these are also retrieved.

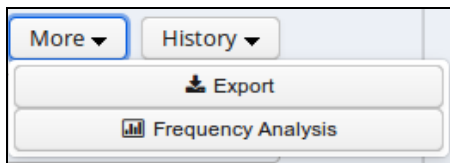
It is also possible to output metadata annotations per hit using the WekaExporter. To do so, use the parameter `metakeys=meta1,meta2` etc. For example, if your documents have a metadata annotation called 'genre', you may export it for each search result as a further column using `metakeys=genre` in the parameters box.

The CSVExporter behaves much like the WekaExporter, except that the Weka header specifying the content of the columns is not used (useful for importing into spreadsheet programs such as Excel or Calc).

Note that exporting may be slow if the result set is large.

4.9 Frequency Analysis

To perform a frequency analysis, enter the query whose results you want to analyze as usual in the AQL box. Note that you **do not need to carry out the query first**. Next, open the menu “More” between the Search and History buttons and select “Frequency Analysis”:



The interface will open the frequency analysis tab shown below. Initially, rows will be generated for the nodes present in the query. For example, two rows are automatically generated for the following query, which finds any pair of consecutive tokens:

```
tok . tok
```

[Help/Examples](#)
[Query Result](#)
[Frequency Analysis](#)

selected corpora:

GUM

query to analyze:

tok . tok

	Node number/r	Selected annotation of node	Comment
1	1	tok	automatically created from tok
2	2	tok	automatically created from tok

Metadata

[Select](#) type

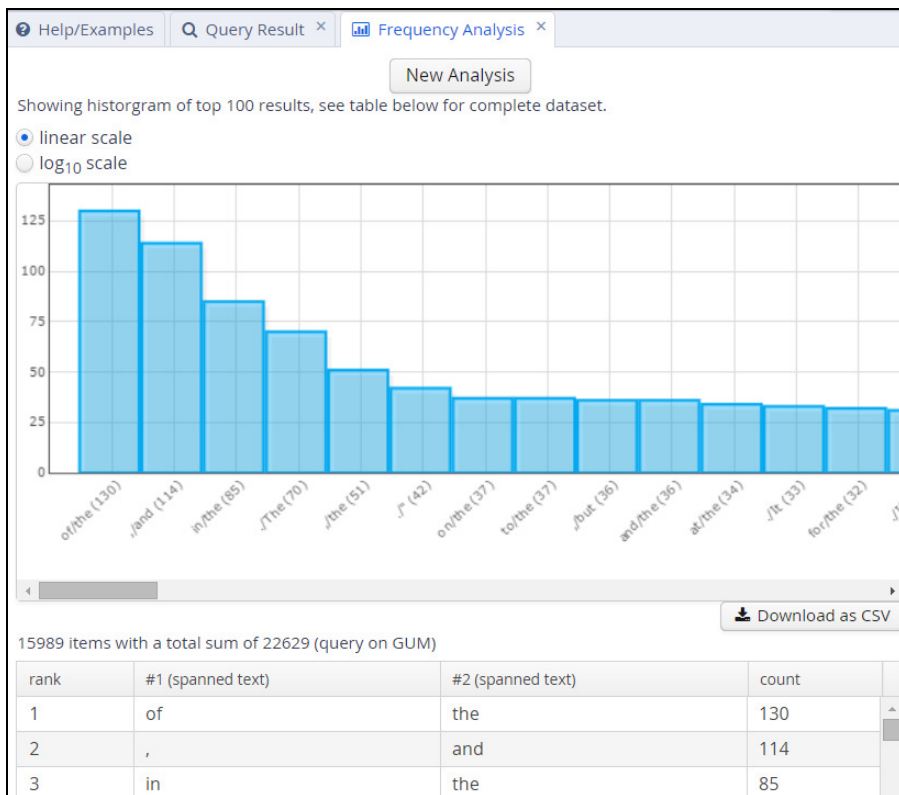
Add

Delete selected row(s)

Automatic mode

Perform frequency analysis

You may also add metadata to the frequency breakdown from the metadata selection link. Clicking on “Perform frequency analysis” will produce a breakdown of all consecutive token bigrams in the corpus. The frequency graph will only show the first 500 elements, but the table below it will give the entire list of values, which can also be **exported as a CSV file**.



To edit the analysis or analyze a new query, click the **New Analysis** button. It is also possible to **add annotations** to the analysis that were not in the original query,

provided that these are expected to belong to some other node in the query. For example, the tokens in the GUM corpus also have part-of-speech and lemma information. We can replace the lines in the analysis specifying that tok values should be counted with pos values, which gives us part-of-speech bigrams. We can also add a lemma annotation belonging to the first search element, by clicking the Add button and entering the node definition number and annotation name we are interested in:

[Help/Examples](#)
[Query Result](#)
[Frequency Analysis](#)

selected corpora:

GUM

query to analyze:

tok . tok

	Node number/i	Selected annotation of node	Comment
1	1	pos	automatically created from tok
2	2	pos	automatically created from tok
3	1	lemma	

Metadata

[Select](#)

Add

Delete selected row(s)

☐ Automatic mode



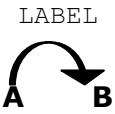
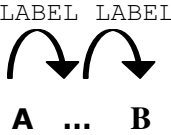

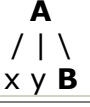





Perform frequency analysis

As a result, we will get a count for each combination of values grouped by the first and second tokens' parts-of-speech, as well as the first token's lemma.

4.10 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. Use <code>.seg_name</code> for precedence on a specific segmentation layer.
.*	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. Use e.g. <code>.seg_name,3,4</code> for 3 to 4 unit distance in another segmentation.
^	directly near	A B or B A	Same as precedence, but in either order. In corpora with multiple segmentations the layer on which consecutivity holds may be specified with <code>^layer</code>
^*	indirectly near	A x y z B or B x y z A	Like indirect precedence in either order. The form <code>^n,m</code> can be used, e.g. <code>^3,4</code> - between 3 and 4 token distance; the default maximum distance for <code>^*</code> is 50 tokens. As above, segmentation layers may be specified, e.g. <code>^layer,3,4</code>
>	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
=	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
o	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		Both elements span an area beginning with the same token
<code>_r_</code>	right aligned		Both elements span an area ending with the same token
<code>==</code>	value identity	$A = B$	The value of the annotation or token A is identical to that of B (this operator does not bind, i.e. the nodes must be connected by some other criteria too)
<code>!=</code>	value difference	$A \neq B$	The value of the annotation or token A is different from B (this operator does not bind, i.e. the nodes must be connected by some other criteria too)
<code>->LABEL</code>	labeled pointing relation		A labeled, directed relationship between two elements. Annotations can be specified with <code>->LABEL[annotation="VALUE"]</code>
<code>->LABEL *</code>	indirect pointing relation		An indirect labeled relationship between elements. The length of the chain is specified with <code>->LABEL n,m</code> for relation chains of length <i>n</i> to <i>m</i>
<code>>@l</code>	left-most child		
<code>>@r</code>	right-most child		
<code>\$</code>	Common parent node		
<code>\$*</code>	Common ancestor node		
<code>#x:arity=n</code>	Arity		Specifies the amount of directly dominated children that the searched node has.
<code>#x:tokenarity=n</code>	Tokenarity		Specifies the length of the span of tokens covered by the node
<code>#x:root</code>	Root		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, ANNIS displays all search results in the Key Word in Context (KWIC) view in the "Query Result" tab, though in some cases you may wish to turn off this visualization (specifically dialog corpora, see below). 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
Nodes with the namespace mmax:	grid visualizer
Edges with the namespace mmax:	discourse view

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 (for more information, see the PAULA XML documentation at <http://www.sfb632.uni-potsdam.de/en/paula.html>). Data converted from EXMARaLDA can also optionally use speaker names as namespaces. For other formats and namespaces, see the SaltNPepper documentation of the appropriate format module (details in Chapter 6).

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. This can either be done by editing the relANNIS file `resolver_vis_map.tab` in the corpus directory before import, or within the database after import. To edit the table in the database after import, 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.[X] > Databases > anniskickstart > Schemas > public > Tables* (for ANNIS servers replace “anniskickstart” with your database name, determined as <dbname> in the installation instructions in Section 3.2). 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:

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

exmaralda									
Select Displayed Annotation Levels ▾									
Focus_newInf									
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	1,52	Meter	große	Gennadi	Subow

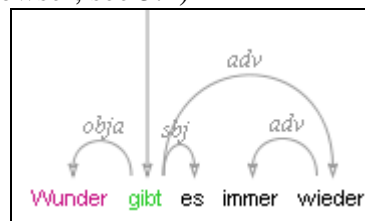
- *grid_tree* (a grid visualizing hierarchical tree annotations as ordered grid layers; note that all layers represent the same annotation name at different hierarchical depths, marked level:0,1,2,... etc. on the left)

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

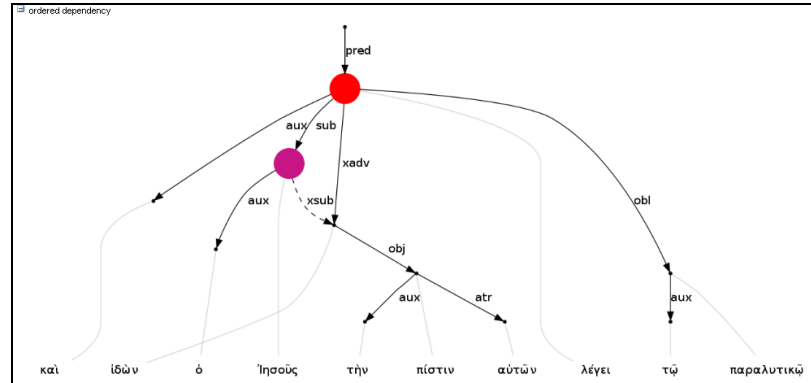
- *discourse* (a view of the entire text of a document, possibly with interactive coreference links. It is possible to use this visualization to view entire texts even if you do not have coreference annotations)

coreference (discourse)	
Steilpass Wunder gibt es immer wieder ! Erst spielen die Dallgowner 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 Dallgowner 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	

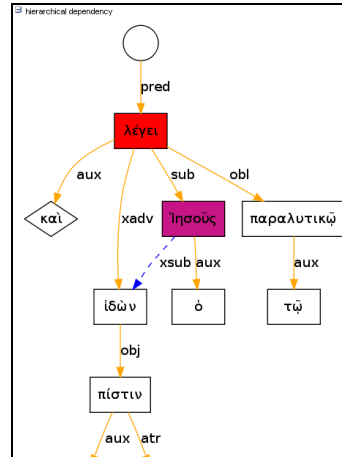
- *arch_dependency* (dependency tree with labeled arches between tokens; requires SVG enabled browser, see 5.2)



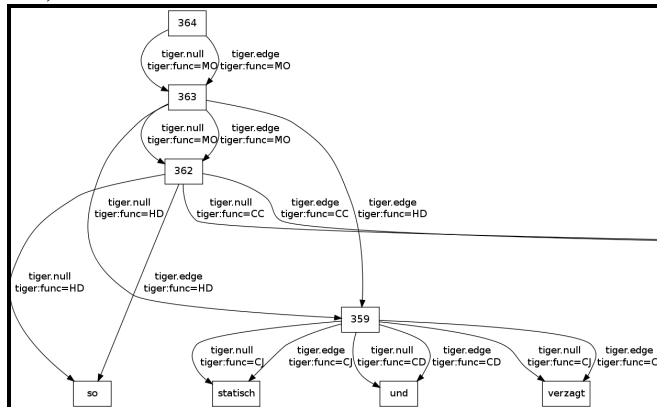
- *ordered_dependency* (arrow based dependency visualization for corpora with dependencies between non terminal nodes; requires GraphViz, see 5.2)



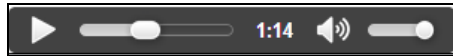
- *hierarchical_dependency* (unordered vertical tree of dependent tokens; requires GraphViz, see 5.2)



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



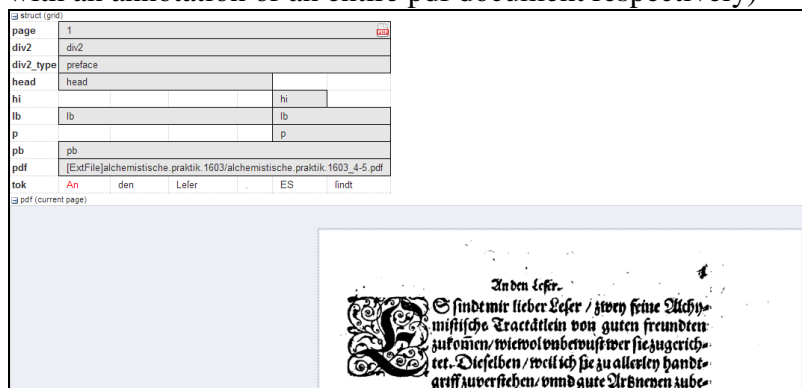
- *audio* (a linked audio file)



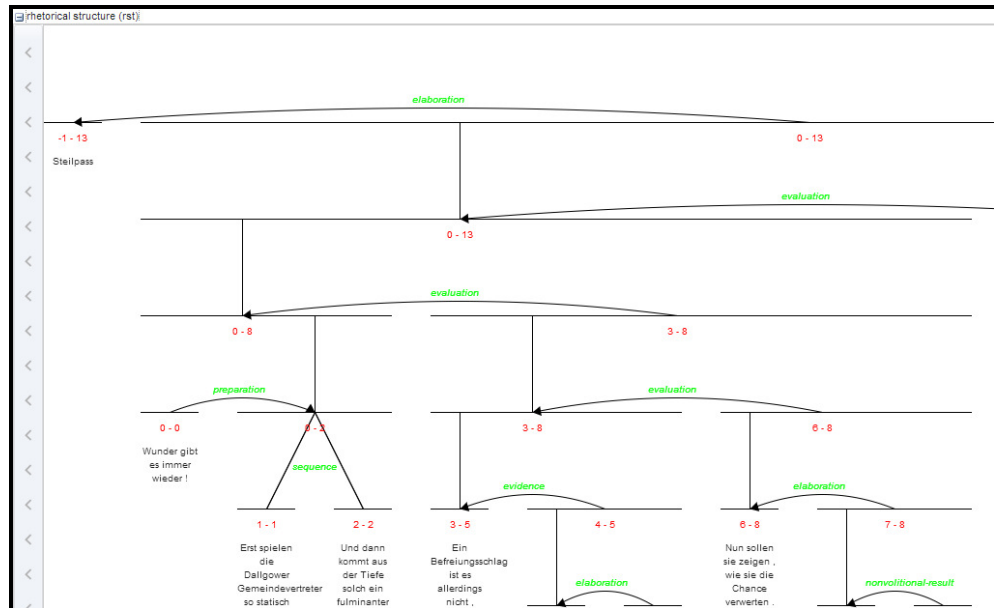
- *video* (a linked video file)



- *pdf* or *pdfdoc* (a linked pdf file, showing either a specific page aligned with an annotation or an entire pdf document respectively)



- *rst* or *rstdoc* (a visualization for rhetorical structure theory annotations, of either just the search result with context or the entire document respectively)



- *html* or *htmldoc* (a versatile annotation-triggered css-based visualization of either the immediate search result context or the entire document respectively; see the ANNIS HTML Visualization Guide for more details and some example stylesheets)

- *raw_text* (a space separated plain text view of the tokens in the entire document)
- *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: (separate multiple values using a semicolon)
 - *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). It is also possible to use a different annotation layer for the leaves of the tree instead of the default tokens by specifying: *terminal_name* for the annotation name and *terminal_ns* for the namespace. Multiple instructions are separated using semicolons.
 - *arch_dependency* – to use a different annotation layer (e.g. *my_annotation*) for the leaves of the tree instead of the default tokens, enter *node_key:my_annotation*.
 - *dot_vis* – 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. It is also possible to add annotations applying to the tokens to the visualization, rather than only span element annotations, by using *tok_anno:true*. Finally, you may hide the tokens that normally appear at the bottom of the grid using *hide_tok:true*.
 - *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.
 - *rst / rstdoc* – the names of *rst* edges can be configured with the setting *edge*. Additionally, some graphical parameters can be modified: *siblingOffset* defines the distance between sibling nodes; *subTreeOffset* defines the distance between node and parent node; *nodeWidth* defines the width of a node; *labelSize* defines the font size of a node label; *edgeLabelColor* specifies an HTML Color for the font color of an edge label; *nodeLabelColor* specifies an HTML Color for the font color of a node label.
 - *pdf / pdfdoc* – it is possible to configure the height of the pdf window using the *height* instruction (in pixels), as well as the name (*node_key*) of the node annotation to be used to give individual page numbers aligned with a span of tokens (relevant for *pdf* only, *pdfdoc* always shows all pages). The instructions can be combined as follows: *node_key:pp;height:400*.
 - *html / htmldoc* – you must specify the name of the css stylesheet (*.css) and configuration file (*.config) for the visualization, which are placed in the ExtData folder of the relANNIS corpus (see HTML Visualization Guide for details). To configure the stylesheet name, use the value

config:filename, where filename is the common name of both the .config and the .css files, without the extension.

- *visibility* is optional and can be set to:
 - *hidden* – the default setting: the visualizer is not shown, but can be expanded by clicking on its plus symbol.
 - *permanent* – always shown, not closable
 - *visible* – shown initially, but closable by clicking on its minus symbol.
 - *removed* – not shown; this can be used to hide the kwic visualization in corpora which require a grid by default (e.g. dialogue corpora).
 - *preloaded* – like hidden, but actually rendered in the background even before its plus symbol is clicked. This is useful for multimedia player visualizations, as the player can be invoked and a file may be loaded before the user prompts the playing action.

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 *dot_vis* 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 means the user's browser must be SVG capable (e.g. Firefox, Chrome, or IE9 or above) or else a plugin must be used (e.g. for Internet Explorer 8 or below). This is the case for the *arch_dependency* visualizer.

5.3 Changing Maximal Context Size, Context Steps and Result Page Sizes

The maximal context size of $\pm n$ tokens from each search result (for the KWIC view, but also for other visualizations) 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, edit the setting `annis.max-context` in the `annis-service.properties`. By default, the context steps 1, 2, 5 or 10 tokens are available. To change the default step and step increment, edit the parameters `default-context=5` and `context-steps=5` respectively.

It is also possible to set context sizes individually per corpus. This is done by editing or adding the file `corpus.properties` to the folder ExtData within the relANNIS corpus folder before import. The names of the parameters are the same, i.e. `default-`

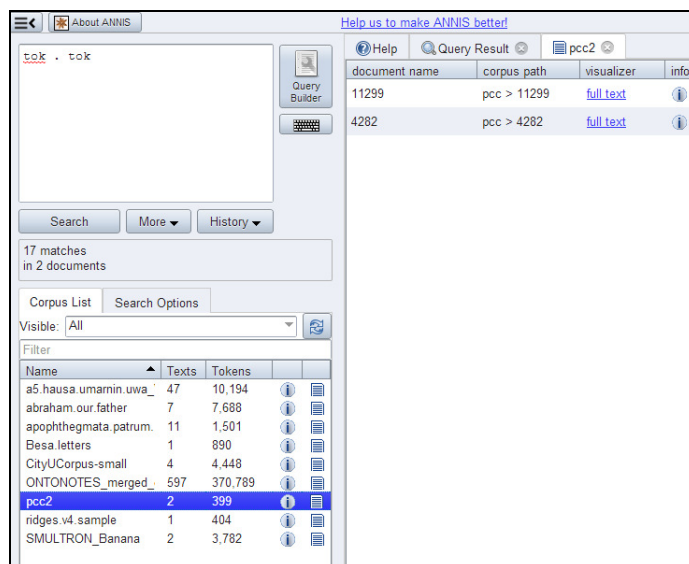
`context=5` and `context-steps`, and their values override the default values in `annis-service.properties`.

To change the available setting for the amount of hits per result page, edit the setting `results-per-page` in `annis-service.properties` as explained above for all corpora, or for specific corpora in `corpus.properties` within the relevant corpus.

Note that for all these settings, if multiple corpora with conflicting instructions are selected, the interface will revert to system defaults **up to** the most restrictive settings imposed by one of the selected corpora (i.e. if one of the selected corpora limits context to ± 5 tokens, the search will obey this limit even if other corpora and the default setting allow more context).

5.4 Configuring the Document Browser

Starting in ANNIS3.1.X, it is possible to view a list of documents for each corpus and visualize documents independently of any queries posed by the user. To open the document browser for a corpus, click on the document icon in the corpus list next to each corpus. By default, a list as shown below is generated with a link to a plain text representation of each document.



document name	corpus path	visualizer	info
11299	pcc > 11299	full text	info
4282	pcc > 4282	full text	info

Name	Texts	Tokens	
a5.hausa.umamin.uwa_	47	10,194	info full text
abraham.our.father	7	7,688	info full text
apophthegmata.patrum.	11	1,501	info full text
Besa.letters	1	890	info full text
CityUCorpus-small	4	4,448	info full text
ONTONOTES_merged_	597	370,789	info full text
pcc2	2	399	info full text
ridges.v4.sample	1	404	info full text
SMULTRON_Banana	2	3,782	info full text

The default configuration for the document browser is stored in the `annis-service.properties` file. It can be overwritten for specific corpora using the `corpus.properties` file in the `ExtData` folder in `relANNIS`. Available keys are:

```
browse-documents=true|false
browse-document-visualizers= {...}
```

Note that the `browse-documents` configuration has only an effect when it is set within `corpus.properties`.

Automatic on/off switch for corpora with no plain text tokens

The ANNIS importer tries to detect corpora containing no underlying token text. This is usually the case if some higher annotation layer is used to represent the base text, e.g. in dialogue corpora, where the token layer is used as an alignment base for annotations representing different speakers. If the relANNIS file `text.tab` contains only white space within documents, then the document browser is disabled for that corpus on import, unless a `corpus.properties` file configures the document browser otherwise.

Custom document visualizers and sorting

It is also possible to use custom visualizers for browsing documents. The configuration is in JSON-Syntax and placed in a file called `document_browser.json`, which can be added to the ExtData directory of each corpus.

```
{
  "visualizers": [
    {
      "type" : "htmldoc",
      "displayName" : "diplomatic text",
      "mappings" : "config:dipl"
    },
    {
      "type" : "rstdoc",
      "displayName" : "rhetorical structure",
    }
  ],
  "metadataColumns" : [
    {
      "namespace" : "annis",
      "name" : "title"
    },
    {
      "namespace" : "annis",
      "name" : "genre"
    }
  ],
  "orderBy" : [
    {
      "namespace" : "annis",
      "name" : "title",
      "ascending" : "false"
    }
  ]
}
```

Details:

- **visualizers – type:** All visualizers from the list above with the suffix "doc" in their name are suitable for use as document visualizers (rstdoc, htmldoc) as well as the discourse visualizer.
- **metadataColumns (optional):** For every defined metadata field an additional column is generated in the corpus browser table with the metadata key as a column header and the metadata value as the table cell value. This is useful for

viewing, and sorting by, different metadata available to the documents. The line “namespace” can be left out if the namespace is null.

- **orderBy** (optional): In the default state the table is sorted by document name. Alternatively it is possible to define a custom sort by the metadata fields, even if the column is not visible. 'namespace' and 'ascending' are optional (if namespace is not specified, null is assumed). 'ascending' is 'true' by default.

5.5 Configuring Right-to-Left Visualizations

The KWIC, grid and tree visualizers support right to left layouting of Arabic and Hebrew characters. As soon as such a character is recognized in a search result, the visualization is switched into right-to-left mode for these visualizers. If this behavior is not desired (e.g. a left-to-right corpus with only a few incidental uses of such characters), this behavior can be switched off for the entire ANNIS instance by setting:

```
Disable-rtl=true
```

in the file `WEB-INF/conf/annis-gui.properties`

6 Importing and Configuring Corpora

6.1 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, generic XML and TreeTagger directly into relANNIS. 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/>

6.2 Importing Corpora in the relANNIS format

Corpora in the relANNIS format can be imported into the ANNIS database. For information on converting corpora from other formats into relANNIS, see the SaltNPepper documentation.

Importing a relANNIS Corpus in ANNIS Kickstarter

To import a corpus to your local Kickstarter, press the “Import Corpus” button on the Kickstarter program window and navigate to the directory containing the relANNIS directory of your corpus. Select this directory (but do not go into it) and press OK. Note that you cannot import a second corpus with the same name into the system: the first corpus must be deleted before a new one with the same name is imported.

Importing a relANNIS Corpus into an ANNIS Server

Follow the steps described in Section 3.2 for importing the demo corpus GUM. Multiple corpora can be imported with `annis-admin.sh` by supplying a space-separated list of paths to relANNIS folders after the import command:

```
bin/annis-admin.sh import path1 path2 ...
```

6.3 Configuring Settings for a Corpus

Generating Example Queries

User created example queries are stored in the file `example_queries.tab` within the relANNIS corpus folder. The file contains two columns (tab delimited), the first with a valid AQL query for your corpus and the second with a human readable description of the query. These queries are then visible in Example Queries tab of the workspace on the right side of the ANNIS interface.

It is also possible to have ANNIS automatically generate queries for a corpus (instead of, or in addition to user created examples). ANNIS will then create some randomized, typical queries, such as searches for a word form appearing in the corpus or a regular

expression. To determine whether or not example queries are generated by default, change the following setting in `annis-service.properties`:

```
annis.import.example-queries=false
```

On an ANNIS server console it is also possible to generate new example queries on demand, replacing or adding to existing queries, and to delete queries for individual corpora. For more information on the exact commands and options see the help under:

```
bin/annis-admin.sh --help
```

Setting Default Context and Segmentations

In corpora with multiple segmentations, such as historical corpora with conflicting diplomatic and normalized word form layers, it is possible to choose the default segmentation for both search context and the KWIC visualization. To set the relevant segmentations, use the following settings in the `corpus.properties` file in the folder `ExtData` within the relANNIS corpus:

```
default-context-segmentation=SEGNAME  
default-base-text-segmentation=SEGNAME
```

For more details on segmentations, see the ANNIS Multiple Segmentation Corpora Guide.

6.4 Multiple Instances of the Interface

Creating 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 these 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, e.g. the home folder of the user running the ANNIS front end (on a server often the tomcat user, or under Windows Kickstarter, in `C:\Users\username\.annis`, or under Mac OSX under `/Users/username/.annis/`, which is a hidden folder; to view hidden folders you may need to reconfigure your Finder application). In ANNIS server scenarios where it is not possible to deploy the home directory of the user running the front end (e.g. no home folder for tomcat), you may prefer to manually set the configuration path parameter `ANNIS_CFG` for ANNIS, by adding something like the following to the shell script starting ANNIS:

```
export ANNIS_CFG=/etc/my_annis_cfg_path/
```

Instances can then be specified under this folder. The name of the JSON file describing the contents of an instance 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. By default "tigersearch" and "flatquerybuilder" are available; if you want to add your own query builder, see <http://korpling.github.io/ANNIS/dev-querybuilder.html>.

Any defined instance is assigned a special URL at which it can be accessed: `http://<server>/<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>).

Embedding Web Fonts

It is also possible to set an embedded font for query result display in your instance, using the same JSON file described in the previous section. To do so, add a **font** entry like the following:

```
"font" :
{
  "name" : "foo",
  "url": "https://example.com/foo.css",
  "size": "12pt" //size is optional
}
```

The .css file referred to must contain a corresponding font-face instruction, as follows:

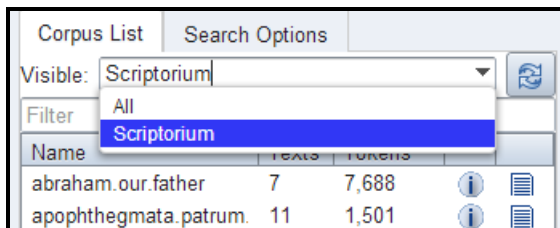
```
@font-face {
```

```
font-family: 'bar';
font-style: normal;
font-weight: normal;
font-size: larger;
src:
    local('bar'),
    url(bar.woff) format('woff');
}
```

Further explanation about the @font-face rule is available on the W3C websites. If you need to have a different font configuration for the frequency chart described in Section 4.9, add a **frequency-font** entry, which has the same structure as 'font'.

Using Corpus Groups

It is possible to group corpora into groups, which are selected above the corpus list in the search form:



While any user can group corpora into corpus sets for their own use, you can define corpus sets for the whole instance using the "corpus-sets" in the JSON file described above. Each corpus set is itself a JSON-object with a name and a list of corpora that belong to the corpus set.

6.5 User management

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 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.

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/
```

1. Create a file "groups" in the user-configuration directory (e.g. /etc/annis/user_config_trunk/groups) with read/write rights for the ANNIS user:

```
group1=pcc3,falko,tiger2
group2=pcc3
group3=tiger1
anonymous=GUM,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 the annis-admin.sh list command).

2. Create a subdirectory users/ with read/write rights for the ANNIS user.
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$1$tQNwUIxEQhrDn6FKcY1yNg==$Xq8ZCb3RFBwn3GfQ7pav3G3vHg4T
KRGD1ItpfDW+JvI=
given_name=userGivenName
surname=userSurname
permissions=adm:*,query:*
expires=2018-04-25
```

The optional entries permissions and expires allow you to give the user special rights to administrate corpora over the web interface (see Section 6.6) and set an expiry date for the user (note that the format is yyyy-mm-dd, as shown above).

Notes:

- A superuser who has access to every corpus can be created with groups=*
- given_name and surname can contain any string
- The password must be hashed with SHA256 (one iteration and using a Salt) and formatted in the Shiro1CryptFormat.
- 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>) which can be downloaded from: <http://shiro.apache.org/download.html#Download-1.2.1.BinaryDistribution> .
 - 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)

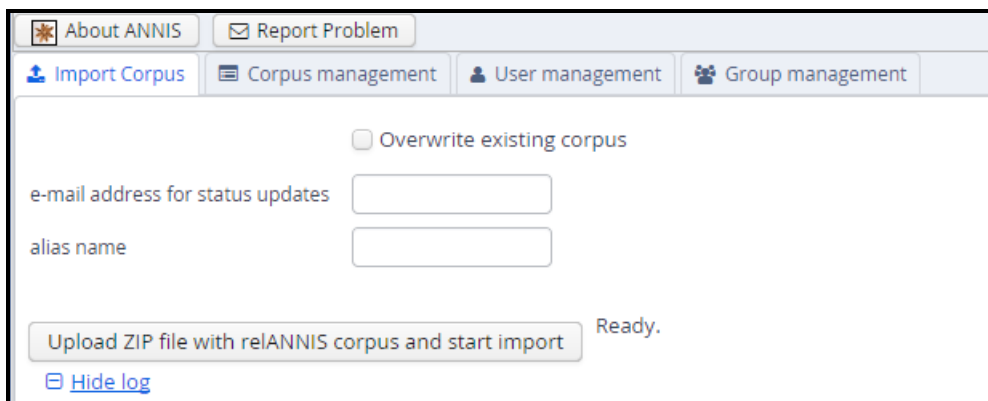
- Type the password
- Retype the password
- It will produce the following output:

```
$ java -jar shiro-tools-hasher-1.2.1-cli.jar -i 1 -p
Password to hash:
Password to hash (confirm):
$shiro1$SHA-
256$1$krMX+Et6w7XJgwSEAgq9nw==$sQOgObXsQdO76wnNxvN0aesvTSPo
Bsd/2bjxasydB+I=
```

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

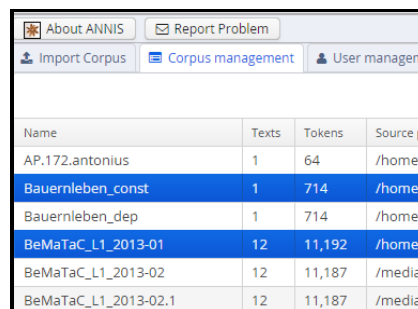
6.6 The administration web interface

New in ANNIS 3.2.X is the administration web interface, which allows you to import and delete corpora, as well as to define users from the ANNIS web application directly. To log into the administration interface, use the URL of you ANNIS webapp and add /admin. You should see the screen below:



The screenshot shows the 'Import Corpus' tab of the ANNIS administration interface. At the top, there are links for 'About ANNIS' and 'Report Problem'. Below these are tabs for 'Import Corpus', 'Corpus management', 'User management', and 'Group management'. The 'Import Corpus' tab is active, displaying a form with an 'Overwrite existing corpus' checkbox, an 'e-mail address for status updates' input field, and an 'alias name' input field. At the bottom, there is a button labeled 'Upload ZIP file with relANNIS corpus and start import' and a 'Ready.' status indicator. A 'Hide log' link is also present.

Here you can upload a zipped relANNIS corpus for import and set the e-mail address in the configuration file. The corpus management tab allows you to select corpora for deletion:



The screenshot shows the 'Corpus management' tab of the ANNIS administration interface. It displays a table with the following data:

Name	Texts	Tokens	Source path
AP.172.antonius	1	64	/home/t
Bauernleben_const	1	714	/home/t
Bauernleben_dep	1	714	/home/t
BeMaTaC_L1_2013-01	12	11,192	/home/t
BeMaTaC_L1_2013-02	12	11,187	/media/t
BeMaTaC_L1_2013-02.1	12	11,187	/media/t

And finally the user and group management tabs allow you to add or remove users, give them special permissions, and determine which groups can see which corpora.

About ANNIS

Report Problem
 [Help us to make ANNIS better!](#)
 logged in as "amir"
 Logout

Import Corpus
 Corpus management
 User management
 Group management

New group name
 Add new group
 Delete selected group(s)

Name	Allowed corpora (seperate with comma)
anonymous	Select a5.hausa.umarnin.uwa_V2, apophthegmata.patrum, besa.letters, GUM, GUM.magic, pcc2, sahid
gu	Select Wall Street Journal (dep)

About ANNIS

Report Problem
 [Help us to make ANNIS better!](#)
 logged in as "amir"
 Logout

Import Corpus
 Corpus management
 User management
 Group management

New user name
 Add new user
 Delete selected user(s)

Username	Groups (seperate with comma)	Additional permissions (seperate with comma)	Expiration Date	
amir	Select *	Select admin:*, query:*	<input type="checkbox"/> expires	Change password
gu_student	Select anonymous, gu	Select	2015-12-01 <input checked="" type="checkbox"/> expires	Change password

These functions edit the files described in Section 6.5. Note that the user management function treats any file in the users/ directory from Section 6.5 as a user entry, and no files except for user files may exist in that directory (otherwise the administration functions will not load correctly).