CESAR: Corpus Editor for Syntactically Annotated Corpora

Reference manual
version 1.8
15 February 2021

This is the manual for CESAR, an online search interface that allows computationally naive researchers to perform quantitative corpus searches. The interface was designed and fine-tuned for the automatic analysis of coherence relations, but can be used to research other linguistic phenomena as well.

CESAR can be accessed at https://cesar.science.ru.nl/

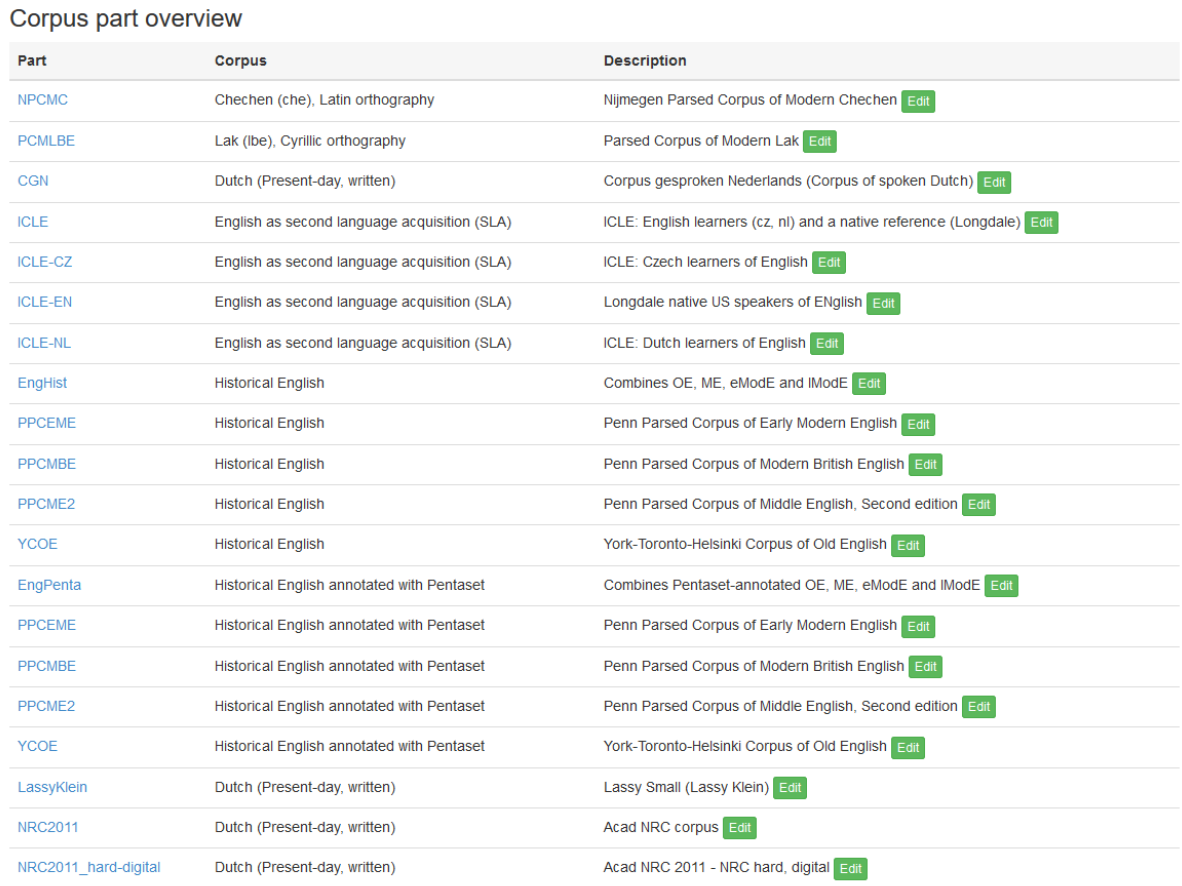
TABLE OF CONTENTS

1 Creating an account

Anyone interested in using CESAR can create an account by clicking one of the SIGN UP buttons on the main page. Alternatively, the sign up page can be found by clicking 'Sign up' under the 'Extra…' button in the top right corner of the screen.

2 Manually searching the available corpora

Signed in users can manually search several corpora in the CESAR portal by clicking 'Browse' in the yellow ribbon at the top of the screen. Under 'Browse', you can immediately select a specific corpus to search, or opt to search 'All texts'. Clicking 'Corpus parts' will send you to an overview of all available corpora with a short description, see Figure 1.



Corpus part overview

Part	Corpus	Description
NPCMC	Chechen (che), Latin orthography	Nijmegen Parsed Corpus of Modern Chechen Edit
PCMLBE	Lak (lbe), Cyrillic orthography	Parsed Corpus of Modern Lak Edit
CGN	Dutch (Present-day, written)	Corpus gesproken Nederlands (Corpus of spoken Dutch) Edit
ICLE	English as second language acquisition (SLA)	ICLE: English learners (cz, nl) and a native reference (Longdale) Edit
ICLE-CZ	English as second language acquisition (SLA)	ICLE: Czech learners of English Edit
ICLE-EN	English as second language acquisition (SLA)	Longdale native US speakers of ENglish Edit
ICLE-NL	English as second language acquisition (SLA)	ICLE: Dutch learners of English Edit
EngHist	Historical English	Combines OE, ME, eModE and lModE Edit
PPCEME	Historical English	Penn Parsed Corpus of Early Modern English Edit
PPCMBE	Historical English	Penn Parsed Corpus of Modern British English Edit
PPCME2	Historical English	Penn Parsed Corpus of Middle English, Second edition Edit
YCOE	Historical English	York-Toronto-Helsinki Corpus of Old English Edit
EngPenta	Historical English annotated with Pentaset	Combines Pentaset-annotated OE, ME, eModE and lModE Edit
PPCEME	Historical English annotated with Pentaset	Penn Parsed Corpus of Early Modern English Edit
PPCMBE	Historical English annotated with Pentaset	Penn Parsed Corpus of Modern British English Edit
PPCME2	Historical English annotated with Pentaset	Penn Parsed Corpus of Middle English, Second edition Edit
YCOE	Historical English annotated with Pentaset	York-Toronto-Helsinki Corpus of Old English Edit
LassyKlein	Dutch (Present-day, written)	Lassy Small (Lassy Klein) Edit
NRC2011	Dutch (Present-day, written)	Acad NRC corpus Edit
NRC2011_hard-digital	Dutch (Present-day, written)	Acad NRC 2011 - NRC hard, digital Edit

Figure 1. Overview of available corpora in CESAR

Selecting a specific corpus or 'All texts' will send you to the manual search interface.

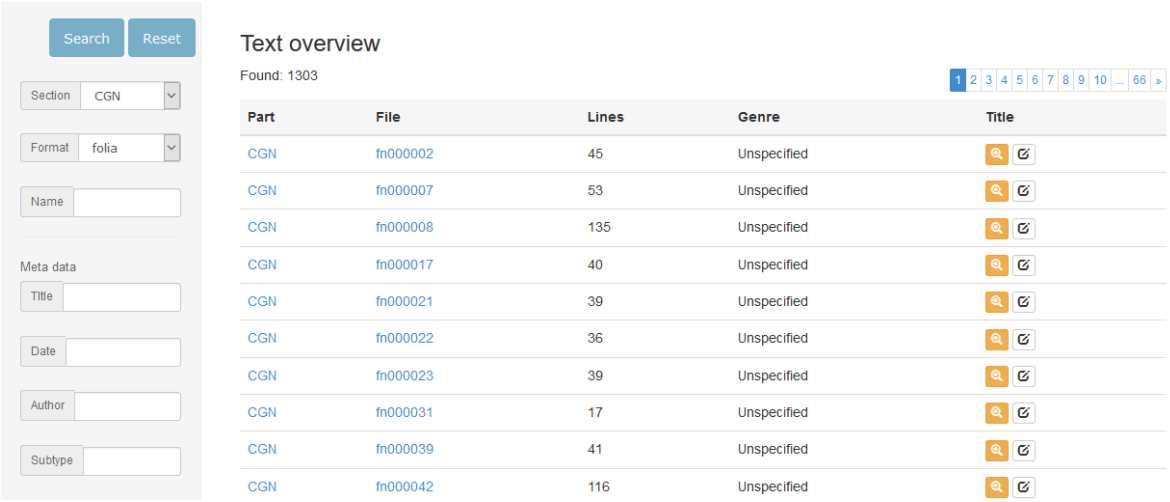


Figure 2. Manual search interface

Corpora can be searched on the basis of:

Section The specific (subpart of a) corpus

Format Either Folia or psdx. If you don't require a specific format, the format is best left unspecified. When selecting a corpus for which only one format is available, the format of the corpus will be automatically selected.

Name If you are looking to find a specific corpus file, you can enter the name of the file here. The name may also contain a wildcard asterisk (e.g. *04*)

Metadata

You can also search the available corpora on the basis of their metadata. Since the availability of metadata is contingent upon the original corpora, not all metadata information is available for each corpus. To see which metadata is available for a specific corpus file, click on the icon of a particular text.

Title The title of the corpus text. This may, but need not be the same as the name of the corpus file.

Date The date at which the text was originally published.

Author The author of the text.

Subtype If applicable, in which subpart of the corpus the file is located.

Genre The genre of the corpus text.

Clicking on the file name sends you to the text. Each new sentence in a text is presented on a new line. Clicking a sentence sends you to the syntactic tree of the sentence:

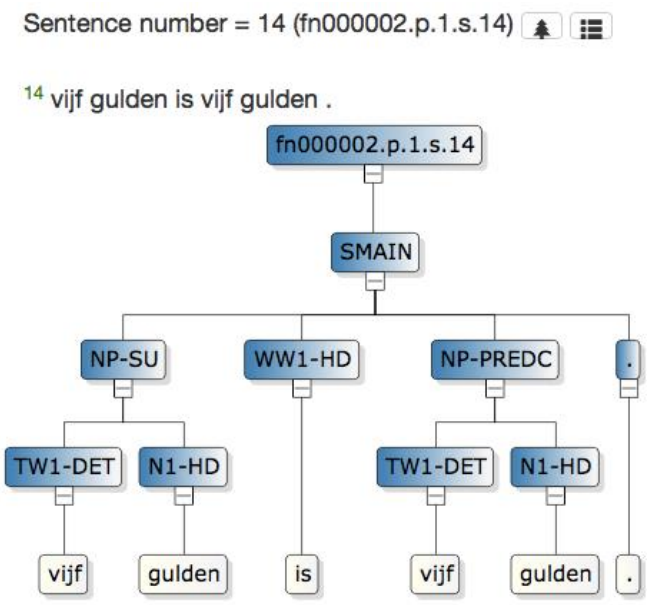


Figure 3. Syntactic structure in tree view

Clicking the icon gives you a schematic overview of the syntactic structure of the same sentence:

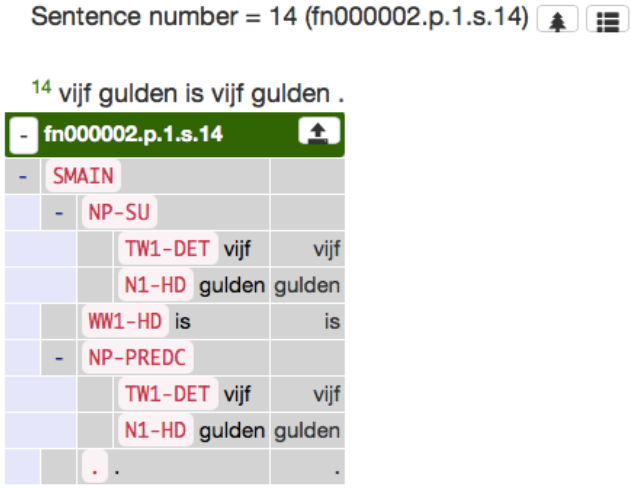


Figure 4. Syntactic structure in schematic overview.

Return to the tree view by clicking .

For citation information for all corpora that can be accessed through CESAR, see Section 9 of this manual.

Note: An alternative to completely manually searching could be to do a simple 'string search'. This finds all occurrences of a word or multi-word search item. To do a string search, create a search project (see Section 3), define your search element (see Section 4.1), execute the search project (see Section 5), and consult your results (see Section 6).

3 Simple searches

3.1 Word-based search

In order to facilitate finding examples of constructions quickly, CESAR contains a 'simple search' facility. Going to Search > Simple opens the interface as shown in Figure 5.

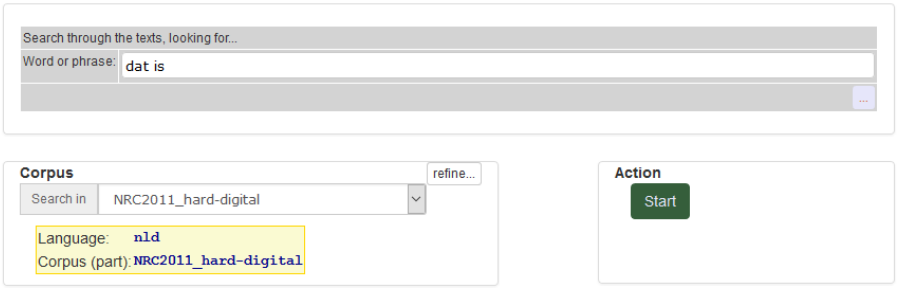


Figure 5. Defining a simple search

The simplest of searches that can be defined is the **'word'-based search**. Just enter one word or a phrase that needs to be found. Then specify a Corpus to be search and hit start. The results can be reviewed as explained in section 7.

3.2 Constituent category search

Instead of searching for one or more words, it is also possible to search for a syntactic category (a POS-tag). The category may contain wildcards (the asterisk '*'). To start a constituent-based search, click the "…" button. The interface now changes slightly, as can be seen in Figure 6.

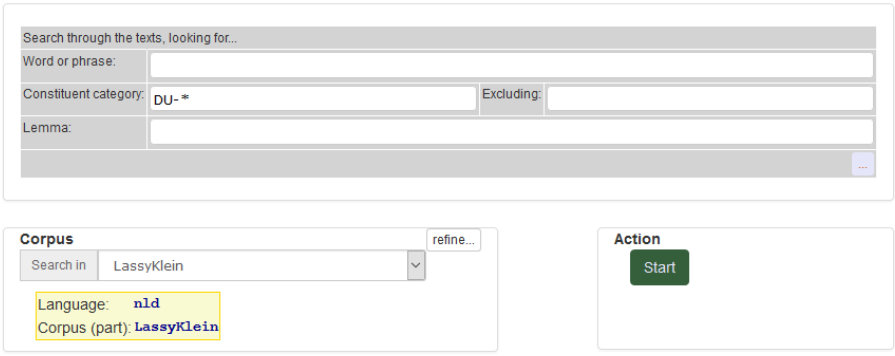


Figure 6. Searching for syntactic categories

The example shows a search for all constituents that have a syntactic category starting with **DU-**, where "DU" stands for "Discourse Unit" (in the LassyKlein and SoNaR Dutch corpora).

The standard 'output features' (see section 5.4) of simple searches contain: (a) the word or constituent that matches the search criteria (ft_searchWord), and (b) the syntactic category of that word or constituent (ft_searchPOS). So if a researcher would like to find out how many different kinds of DU- tags are being used in the 'LassyKlein' corpus, then simply exporting the search results (see section 7.3) as Excel, and applying a cross-table should give information similar to that in Table 1.

Table 1. Finding the frequency-of-occurrence of POS-tags

Tag	Frequency	Tag	Frequency	Tag	Frequency
DU-NUCL	763	DU-MOD-1-2	6	DU-CNJ-5-8	1
DU-DP	721	DU-MOD-3	6	DU-CNJ-9	1
DU-CNJ	461	DU-MOD-1	5	DU-MOD-1-3	1
DU-MOD	455	DU-SU	5	DU-MOD-1-4	1
DU-SAT	256	DU-DLINK	4	DU-MOD-2-4	1
DU-BODY	70	DU-CNJ-3	3	DU-MOD-2-5	1
DU-OBJ1	24	DU-MOD-1-5	3	DU-MOD-2-6	1
DU-APP	23	DU-MOD-4	3	DU-MOD-4-9	1
DU-TAG	18	DU-MOD-5	2	DU-MOD-8	1
DU-MOD-2	11	DU-NUCL-1	2	DU-PREDM-2	1
DU-PREDM	8	DU-CNJ-2	1	DU-SAT-3	1
				DU-VC	1

3.2.1 Lemma specification

Simple searches that are restricted to *words* can also look for a **lemma** (using wildcards if needed). (This only works for copora that have lemma annotation.) And *constituent* searches can also contain a constituent category (using wildcards) that is to be excluded. E.g. one could look for '`IP-*`', excluding '`*-PRN`'.

3.2.2 Feature restriction

Simple searches that look for *words* (or *constituents* that are restricted to words) can include testing for the presence of a particular 'feature' (see the text-producing function 'feature' in 11.3.1. This feature has been added in 2021.

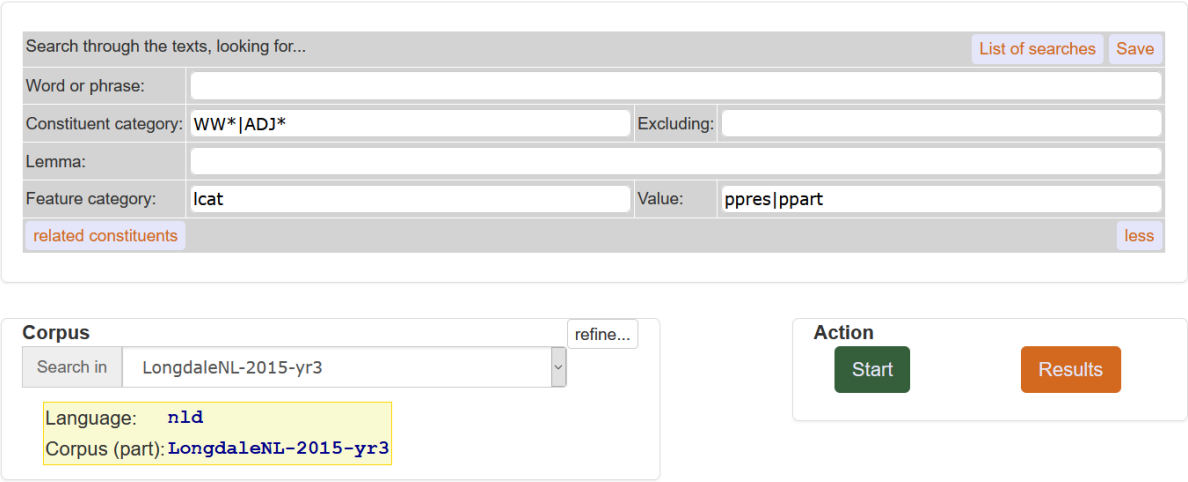


Figure 7. Restricting a search to a feature value

The search defined in Figure 7 looks for participles in the 'LongdaleNL' corpus by filtering on constituent categories WW* or ADJ* (since participles are sometimes tagged as verbs, sometimes as adjectives). But it adds a condition that the feature category **lcat** needs to be either **ppres** (present tense participle) or **ppart** (past tense participle).

Note: features are very much corpus-specific. The feature categories and values that are found in one corpus may differ completely from those found in other corpora.

3.3 Add related constituents

Both the word-based as well as the constituent-category-based searches can be extended by specifying any number of related constituents. Suppose a research wants to look for Dutch main clauses that have a subject, object and a prepositional phrase that is headed by the

preposition *voor* 'for'. A start would be to look for the word *voor*, i.e. a word-based simple search.

The next step would be to add 'related constituents'. This is done by pressing the 'related constituents' button and adding one or more lines.

Note that the example above makes use of rather specific **tag information**: it assumes main clauses are coded like `SMAIN`, subjects as `NP-SU` and so on. The simple search above will only work in the Dutch language corpora. They are not suited for e.g. English, because the texts from that language use a different tagset. More information on the tag sets that are used by the different corpora in CESAR is in Appendix 11.1.

3.4 Save a simple search

The last executed simple search is 'saved'. That search is available whenever a user returns to Simple Search.

But there is more that can be done with simple searches:

1) Save a simple search, **converting** it into a full-blown Cesar search project
2) Save a simple search under a **name**

As for (1), the **conversion** of a search *into a Cesar project*: as soon as a search has been executed on a particular corpus, and results have been received, a button “Save” will appear. Press this button, provide a name, and the research project becomes available.

> NOTE: a ‘normal’ CESAR search project does no longer show up under the simple search tabular interface. It has been converted under method (1) into a standard, full-blown, search project, and should be treated as such from now on.

As for (2), saving a simple search under a particular name: press the “Save as…” button that is available at the right hand corner of the first row on the “Simple search” interface. This brings up an interface (as in Figure 8) to provide a name and give it a description.

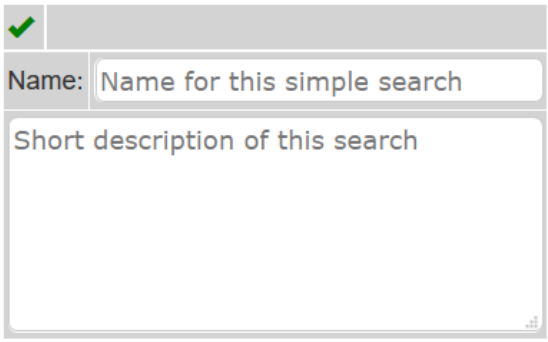


Figure 8 Save a simple search under a name

Upon filling in a name (and optionally a description), press the green 'save' button. The newly named simple search project can now be saved using the "Save" button at the right hand corner of the first row on the "Simple search" interface. The interface will show a wait symbol and then re-appear with the loaded simple search, that now has a name badge next to the words "Simple search" at the top of the page.

3.5 Listing simple searches

A list of the user's simple searches is available under Search > List simple searches. This view shows all the simple searched defined by the user, providing at-a-glance information, such as the name, the search description and the date of creation.

Simple searches can also be removed easily in this view by pressing the red "x" button on the right hand side of the search (a button will appear, asking to confirm the deletion).

3.6 More on related constituents

For users who would like to work with related constituents and use them to build simple search queries, there are a few more things that can be said in their support.

3.6.1 Be aware of the tag sets

Please be aware of the fact that the different corpora under the hood of CESAR can make use of slightly different tag sets. As soon as a search involves the syntactic category of a constituent, it means that the search is inteded to be used for a particular tag set. It will probably not give much results for texts from other corpora. More information on the tag sets that are used by the different corpora in CESAR is in Appendix 11.1.

3.6.2 Overview of relations

As in many parts of CESAR, the simple search makes use of hierarchical relations. X lists all available relations (which are inherited from XPath).

Table 2. Hierarchical relations

Ancestor	Any node hierarchically above the specified node
Ancestor-or-self	Any node hierarchically above the specified node, or the specified node itself
Child	The node directly below the specified node
Decendant	Any node hierarchically below the specified node
Decendant-or-self	Any node hierarchically below the specified node, or the specified node itself
Following	Any following node within the same tree
Following-sibling	A following node on the same level as the specified node
Parent	The node directly above the specified node

Preceding	Any preceding node within the same tree
Preceding-sibling	A preceding node on the same level as the specified node
Self	The specified node itself

How these relations work out can be illustrated by looking at one of the results of the 'voor' search specified above.

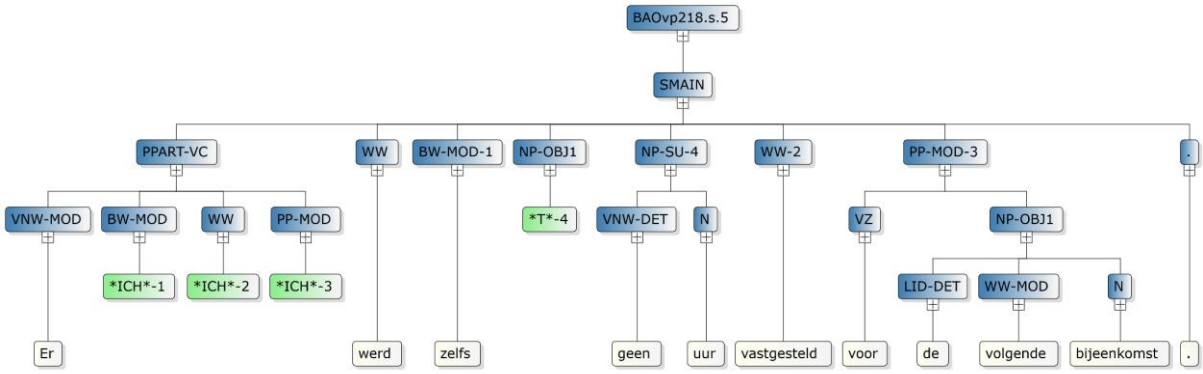


Figure 9 One of the results of looking for 'voor'

The word 'voor' has **ancestors**: `VZ`, `PP-MOD-3`, `SMAIN`
The main clause `SMAIN` has **children**: `PPART-VC`, `WW`, … `PP-MOD-3`, and the period (punctuation). Its **descendants** include everything below it.
The **preceding-siblings** of `NP-OBJ1` are: `PPART-VC`, `WW`, `BW-MOD-1`.

NOTE: the preceding-siblings (and the following-siblings) are listed in *document order*. This is a characteristic of XPath. This means that the **first** preceding-sibling of `NP-OBJ1` is `PPART-VC`.

3.6.3 Extended relation specifications

The specification of related constituents does not only consist of it relation to another constituent. The specification may also include: syntactic category to match (button "C"), position ("P"), constituent text to match ("T"), and lemma to match ("L").

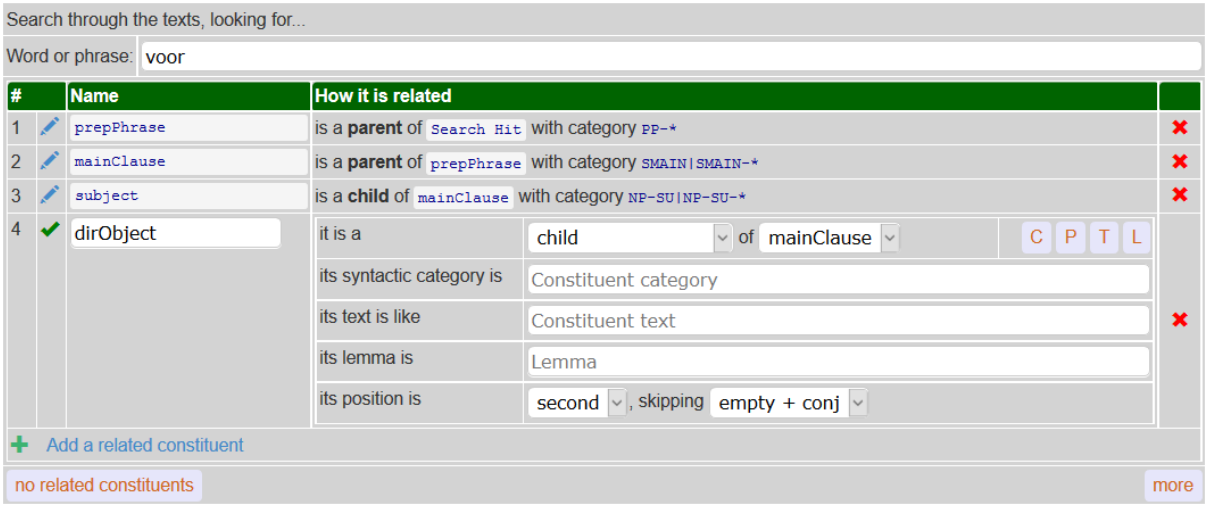


Figure 10 The specifiable components of related constituents

The 'position' follows the document order (see 3.6.2). The position is interpreted in two different ways, depending on whether a syntactic category is specified:

- Position *with* category specification: the position is within the constituents that satisfy the constituent category specification. So if the category is "NP-SU", and the position is "second", then the search looks for the second "NP-SU". If 'skipping' is set to "empty", no syntactically empty constituents of type "NP-SU" will be included.
- Position *without* category specification: the position is absolute, modulo the skipped categories. So the 'first', 'second' or 'last' constituent of the specified hierarchical relation (e.g. *child* in Figure 10) will be taken.

Example: If one would like to look for main clauses that start with a verb, then the position specification can be used in the way shown in Figure 11.

Figure 11 How to find verb-first instances

The search in Figure 11 starts from main clauses, specified by IP-MAT* (this assumes we are looking in an English language corpus, e.g. Longdale). The first related constituent is called **second**, and it is specified as the second child of Search Hit. This related constituent makes use of the *absolute* position, since no syntactic category is specified in it.
Line #2 has the definition of the finite verb **vFin**, and this is specified as the preceding-sibling of **second** that is of the category of a finite verb (VBP, for instance, is a present-tense verb in the English tag system). Crucial here is that **second** can only have one preceding-sibling, since **second** itself is in an absolute *second child* position per its specification.

4 Create a search project

The main function of the CESAR web portal is to enable students and researchers to formulate and execute automatic corpus searches. You can start your project from scratch (see 3.1) or use an existing project as the basis for you own project (see 3.2). At some point, you may decide to share your project with other users as well (see 3.3). Sharing projects can help eliminate the need for users to start from scratch, either for entire projects or for individual functions. Exporting and importing projects is another option to exchange projects with one another (e.g. for assignments in a course), see Section 7.3.

4.1 *Create a new project*

To create a new search project, go to Search > Specify. You can now name your project and provide a short description of the purpose of the project. Select whether your search project focuses on words (for instance 'want' or 'maar') or constituents (for instance 'VG', or 'BW-CRD'). You may want to list your project under a research project group; leave this box unspecified to have your project listed under your account only. Click 'save' to create your project. Click on 'Overview' to go to the main page of your project, see Figure 12.

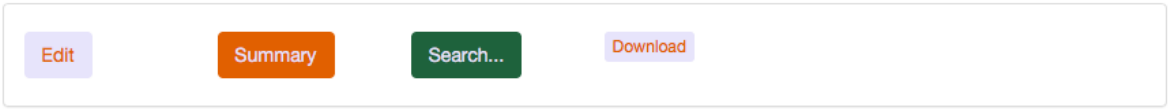


Figure 12. Main page of search project

4.2 Copy an existing project

Adapting an existing project prevents you from having to start from scratch. Projects available for copying can be found under Search > List_of_searches. This opens up the 'group view'of the research projects. Locating the project that needs to be copied can be done in two ways.

1) Find the project group tree under your name and then click the project's name so that the project-specific icons become visible (*open*, *copy*, *download* and *delete*).
2) Open the list of all projects (click 'recent projects' and then 'more') and locate the project there.

Copy the project by clicking the  icon. You can now rename the project and edit its contents.

4.3 Share a project

By default, research projects are not visible to other users, but you can manually change this by sharing your project. To share your project, go to the 'About' section of your project (see Figure 13) and click on "Share with another group". Select either a specific group or select 'seeker-user' to share your project with all users. Under 'Description' you can specify if other users can just view and copy your project ('reading') or also edit it ('reading and writing').

5 Edit a search project

Once you have created your project, you can start editing it. Clicking 'Edit' on the main project page will give you an overview of the structure of a search project, see . You first define what it is you are searching for under 'Search elements', see 4.1, after which you define fixed and data-dependant variables for your search, see 4.2. Under 'Conditions' you can use your variables to specify whether a search hit is relevant to your project, see 4.3. Finally, you can specify the information that will be supplied in the search results under 'Output features', see 4.4.

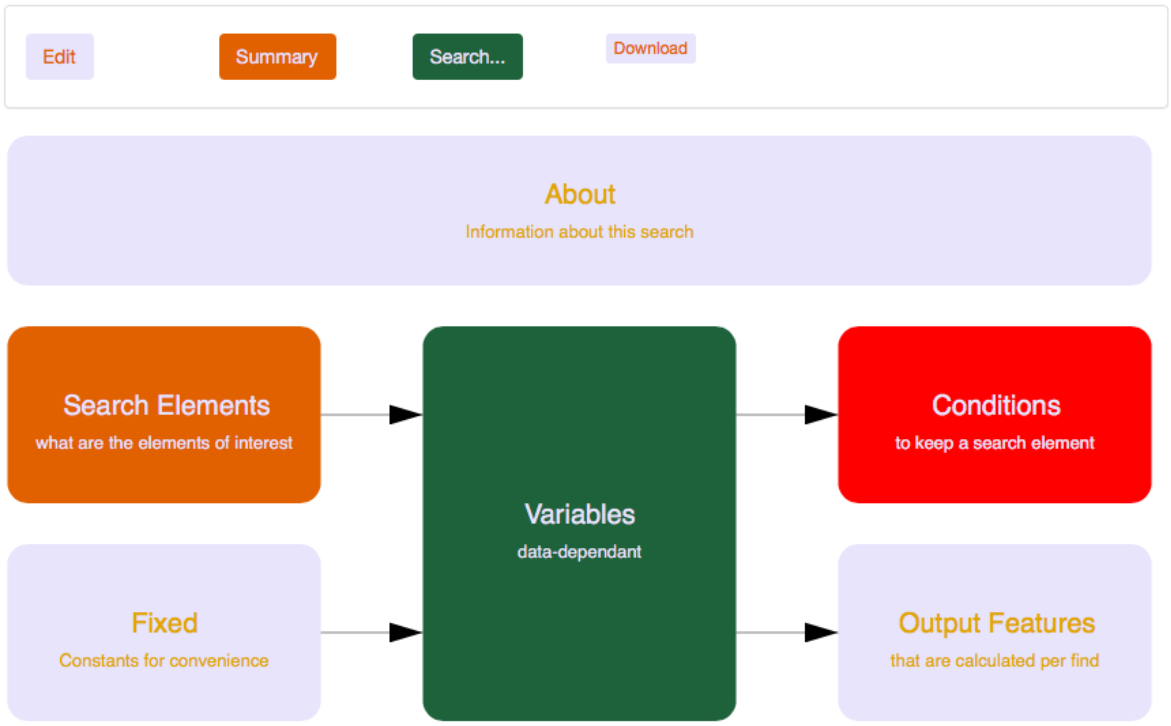


Figure 13. Editing page of search project

5.1 Define search elements

The main focus of any search project is defined under 'search elements'. You can search for words ('want'), POS-tags ('ADJ'), or syntactic components ('SU'). There is a slight difference between the specification of a 'word'-search and a 'constituent'-search.

Word-focused searching

Provide a short description of the element under 'Name' and the actual search items under 'Word(s)'.

When searching for multi-word elements, put the words on the same line, see Figure 14.

When searching for multiple different elements, put each element on a new line, see Figure 14.

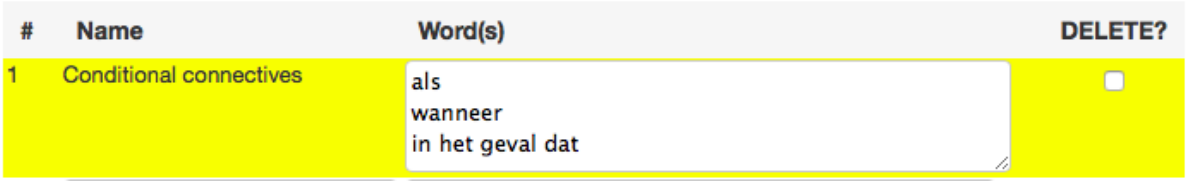


Figure 14. Defining search elements in a word-search

Save your search elements by clicking 'Save' before returning to the main search menu. You can delete a search element by selecting the delete box and clicking 'Save.'

Constituent-focused searching

Provide a short description of the element under 'Name' and specify the actual search information under 'Category' and '[Exclude]'. The 'Category' takes a list of syntactic category labels divided by vertical bars (the logical 'or' sign) and can make use of wildcards such as the asterisk *. The '[Exclude]' section is optional and need not be specified. The example in Figure 15 defines a search for main clauses, but it excludes main clauses that are labeled e.g. IP-MAT-PRN.

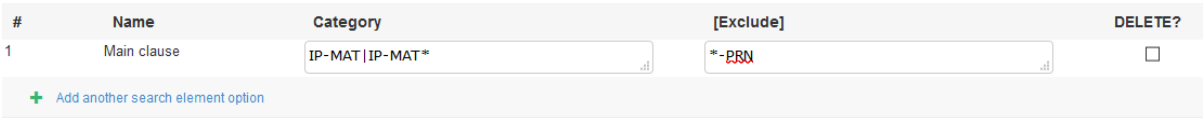


Figure 15. Defining search elements in a constituent-search

5.2 Fine-tune search

Defining your search element and leaving the rest of the project empty is equivalent to a simple 'string search', similar to if you would take a text in Word and find all instances of, for example, 'als' by using the basic search function. Defining variables under 'Fixed' and 'Variables' allows you to go beyond this. Using the variables, you can determine specific features of your search item, for instance its POS-tag and if it occurs sentence-initially. You may be interested in the output of these calculations itself; if you for instance want to manually compare the usage of sentence-initial 'als' versus 'als' in sentence-medial position, you can use the variable that tells you whether 'als' is used sentence-initially to sort your dataset. You may also calculate a variable to use in another variable; if you want to automatically extract the discourse segments linked together by 'als', the output of the variable that tells you whether 'als' is used sentence-initially to sort your data can be used to determine where your segmentation variable(s) looks for the discourse segments. Variables that you use multiple times throughout your search project, you define under 'Fixed'; other variables you define under 'Variables'.

> **Note:** Defining variables will often require using the syntactic corpus annotations (e.g., POS-tags or syntactic nodes). For more information on the syntactic annotations used in the Dutch-language corpora available in CESAR, see the appendix section 11.1.
>
> You can determine which POS-tags or syntactic nodes to use in your variables by only consulting the definitions of the tags and categories, but it can also be highly useful to find relevant examples by browsing the corpus (see Section 2) and check the annotations in those examples.

5.2.1 *'Fixed' – Global variables*

Under 'Fixed' you define variables that you use multiple times in your search project. An example of a situation in which it is useful to define a fixed variable is if you want to determine whether 'als' is followed by a finite clause. You can create a fixed variable 'finite_clause' and list nodes that refer to finite clauses as its 'value', for instance 'CP|SSUB*|SMAIN*'. Use a vertical bar to separate alternatives; use a * for 'wildcard searches' – this specifies that the

node does not have to completely match the search value, but that at least the beginning of the node has to correspond to the search value (i.e., searching for 'SMAIN*' will come back with nodes labeled 'SMAIN' but also nodes labeled 'SMAIN-DP' or 'SMAIN-NUCL').
To delete a variable, tick the 'Delete' box for the specific variable and click 'Save.'

5.2.2 'Variables' – Data-dependant variables

'Variables' is where you calculate features of your search items using pre-defined functions. Note that while you do not need any programming skills to choose functions and define which arguments they work with, it may initially be quite challenging to create data-dependant variables that do what you want them to do. A few tips to help ease the learning curve:

- Start by looking at some variables from existing search projects to get a feeling for what functions look like.
- Start by defining simple variables and slowly increase the complexity of your calculations.
- Try to find variables from existing projects that bear similarities to the variable you are trying to create. If possible, copy and/or adapt existing variables.

You can create a new variable by clicking '+ Add another construction variable definition.' You can now provide a name and a short description of your variable. Click on 'Specify for search elements' to define the function.

You can define the function for each search item specified under 'Search elements.' This allows you to adapt the function depending on the search item. This might be relevant in some situations, for instance if you are interested in a group of linguistic elements that are not all of the same word class (e.g., one is a conjunction while the other is an adverb) – this may influence how you calculate certain features. If the calculations are the same for different search items, you can create the variable for the first search item and copy it to other items, see Figure 16.

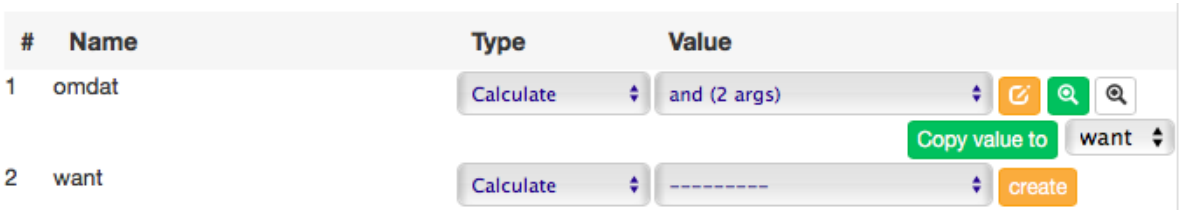


Figure 16. Defining and copying a data-dependant variable

To define a variable, first select the type of variable:

Calculate: This type of variable allows you to select functions under 'Value' that will allow you to calculate features of your search items.
Fixed value: This is a fixed variable that does not involve any calculations, such as a POS-tag, a syntactic node, or simply a word.
Global: This allows you to select a variable specified under 'Fixed'.

Upload JSON: This allows you to upload a JSON file that defines the variable. You can obtain a JSON file by downloading (part of) an existing function (icon in the table view, see section above Figure 18a)

If you choose 'Fixed value' or 'Global value', define the value (see also 4.2.1) or select the global variable from the drop-down menu and save your variable.

If you choose 'Calculate', select a function from the drop-down menu. You can find an overview of all available functions, along with a brief definition and the number of arguments each function takes, under 'Search – Functions.' Clicking opens the function and allows you to specify its arguments, see Figure 17 for an example. The number of arguments a function takes depends on what it is the function does and can range from 1 to 7.

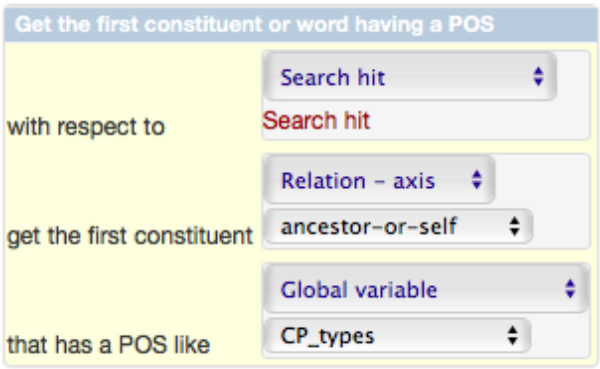


Figure 17. Specifying the arguments of a function

The form each argument can take also depends on the function. The drop-down menu for a function's argument can give you the following options:

Search hit:	Select this for the argument to be the search hit, i.e., the hit found by doing a simple string search for the items specified under 'Search elements'.
Fixed value:	Specify a fixed value (see also 4.2.1)
Dynamic variable:	Select a variable that has already been defined in your search project
Global variable:	This allows you to select a variable specified under 'Fixed'.
Relation – condition:	There are three comparative relations: `equals` – constituent nodes are equal `precedes` – one constituent precedes the other `follows` – one constituent follows the other
Relation – axis:	This allows you to select a relation between two constituents from the syntactic tree, see Table 3.
Relation – constituent:	This allows selecting a constituent relative to the current constituent within a search. Options are:

`self word` - the *word* itself (e.g. line 7, Figure 17)
`self constituent` - the *constituent* itself
`parent constituent` – the parent *constituent* of the current word or constituent (e.g. line 4, Figure 17)
`topmost` – the 'highest' constituent in the sentence

Function output: Select a function from the drop-down menu

Selecting 'Function output' thus allows you to select another function to embed in the first function. This process can be repeated many times and allows you to perform highly complex calculations. An example of a simple variable with no embedded functions can be found in Figure 18; Figure 19 is an example of a highly complex variable with many embedded functions. Functions can be viewed either in a simple view format (green search icon), as in Figure 18a, or in a table format (white search icon), as in Figure 18b. While the simple view format gives you an overview of the entire function at a single glance, the table function is more suited to view and edit specific subparts of the function. In addition, you can download (part of) a function in JSON format by clicking the ⓘ icon. You can upload the JSON file in other variables.

#	Function	Layout ⓘ
1	get_first_relative_cns	with respect to $search get the first constituent x:ancestor-or-self that has a POS like $_CP_types

Figure 18a. Simple variable in simple view

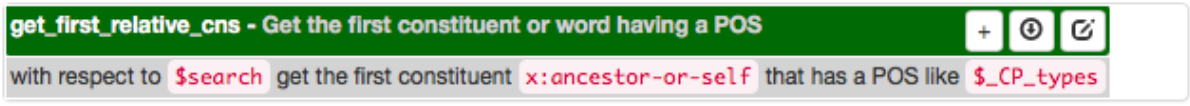


Figure 18b. Simple variable in table format

#	Function	Layout ⓘ
1	if_case_5	if holds line_2 then take line_3 else if holds line_5 then take line_6 else if holds line_8 then take line_9 else if holds line_11 then take line_12 otherwise take line_20
2	is_equal	• Return true if $ini_type equals 'ini_mrk-s1-s2'
3	get_related_exc_w	• Get all the words with relation x:descendant towards constituent $cpMod but exclude those for which holds line_4
4	cns_equals	• • True if constituent rc:parent constituent is the same as constituent $search
5	is_equal	• Return true if $ini_type equals 'medi-ini_s1-mrk-s2'
6	get_related_w	• get all the words with relation x:descendant towards constituent $cpMod provided that line_7
7	word_precedes	• • the word rc:self word precedes constituent $search
8	is_equal	• Return true if $ini_type equals 'ini_s1-mrk-s2'
9	get_related_w	• get all the words with relation x:descendant towards constituent line_10 provided that 'true'
10	get_line	• • Relative line number (negative is preceding) '-1'
11	matches	• Return true if $ini_type equals '*medi_*'
12	get_related_w	• get all the words with relation x:descendant towards constituent line_13 provided that line_19
13	cns_at_pos	• • With respect to line_14 take the constituent at position 'last()'
14	if_exists_2	• • • if exists, take line_15 otherwise if exists, take line_16 otherwise take line_17
15	get_related_2	• • • • get all the constituents with relation x:ancestor towards constituent $search with syntactic category 'CONJ-NUCL|SMAIN*|SSUB*|SV1-*|DU*' which have relation x:parent and syntactic category 'REL-*|WH*'
16	get_related_cat	• • • • get all constituents with relation x:ancestor towards constituent $search provided they have category 'CONJ-NUCL|SMAIN*|SSUB*|SV1-*|DU*'
17	get_related_w	• • • • get all the words with relation x:ancestor towards constituent $search provided that line_18
18	cns_equals	• • • • • True if constituent rc:parent constituent is the same as constituent rc:topmost
19	word_precedes	• • the word rc:self word precedes constituent $search
20	get_related_w	• get all the words with relation x:preceding towards constituent $search provided that 'true'

Figure 19. Complex variable in simple view

Table 3. Relationships in the syntactic tree

Ancestor	Any node hierarchically above the specified node
Ancestor-or-self	Any node hierarchically above the specified node, or the specified node itself
Child	The node directly below the specified node
Decendant	Any node hierarchically below the specified node
Decendant-or-self	Any node hierarchically below the specified node, or the specified node itself
Following	Any following node within the same tree
Following-sibling	A following node on the same level as the specified node
Parent	The node directly above the specified node
Preceding	Any preceding node within the same tree
Preceding-sibling	A preceding node on the same level as the specified node
Self	The specified node itself

To delete a variable, tick the 'Delete' box for the specific variable and click 'Save.'

5.3 Conditions

Under 'Conditions', you can specify which criteria a search hit has to meet to be included in your results. For instance, you might only be interested in 'als' used as a conjunction, but not in 'als' used as an adverb; formulating a condition that filters out al adverbial uses of 'als' will achieve that result.

Specifying conditions is similar to specifying data-dependant variables. Click '+ Add another condition', supply a name and a brief description and specify the type of condition:

Data-dependant variable:	You can only select data-dependant variables with a Boolean (true/false, yes/no; 1/0) output as a condition. By default, search hits with a numerical value 1 or a string value 'true' for the Data-dependant variables will be included in the results; search hits with a numerical value 0 or a string value 'false' will be excluded. You can reverse this by creating a condition with the function 'not' and supplying a data-dependant variable as its argument.
Function:	Create a new function (with a Boolean output) to serve as a condition, see 4.2.
Upload JSON:	Upload a JSON file that defines the condition.

You can select 'no' under 'Include' to (temporarily) de-activate a condition. Excluding condition instead of deleting is preserves your defined condition for future re-activation or copying.

To delete a condition, tick the ‘Delete’ box for the specific condition and click ‘Save.’

5.4 Output features

The output of the variables defined under ‘Variables’ is not necessarily included in the search results. What is more, variables are only calculated if they are used in another variable, in a condition (see 4.3) or if they are used in calculating an output feature. If you do not just want to use a variable to generate results, but also want to see the output of that specific variable, you can define this under ‘Output features’. Add a new feature, provide a brief description of the feature, and select ‘Data-dependant variable’. Select from the drop-down menu the data-dependant variable whose output you want to obtain for each search hit.

> **Note**: Output features need to boil down to strings (to text). So if data-dependant variables yield are of type string, number or boolean, they can be taken just like that as output feature. But data-dependant variables of type ‘constituent’ (or list of constituents) need to be transformed to a string by using a function that takes a constituent as input and outputs a string (e.g. `get_text` or `get_cat` or `get_cns_text`).

You can also calculate new features here by selecting ‘Function’ under ‘Feature type’ (see also 4.2.2). The output of functions defined here cannot be used in other calculations, but will automatically show up in the results file.

You can select ‘no’ under ‘Include’ to (temporarily) stop including a specific output feature in the results files. Excluding an output feature instead of deleting it preserves your defined feature for future re-inclusion or copying.

To delete an output feature, tick the ‘Delete’ box for the specific feature and click ‘Save.’

6 Execute a search project

You can execute your search project by clicking on ‘Search…’ on the main project page. You are sent to the search menu (see Figure 20), where you can specify in which of the available corpora you want to search. If you do not select a specific corpus to search, the search project will be run on all available corpus data.

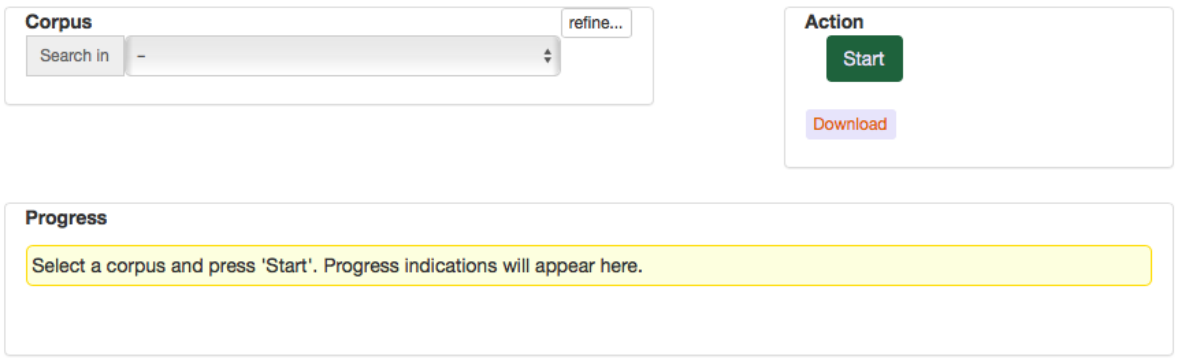


Figure 20. Search menu

For a first search, or when simply using the search function to test the functionality of your project, it may be useful to select the LassyKlein corpus (Dutch). The syntactic parsing of this corpus has been checked by hand, which eliminates the chance of finding unintended results due to parsing errors in the corpus. In addition, LassyKlein is fairly small, so executing the search is quick, which can save a lot of time in the finetuning phase of a search project.

Another way to limit the number of search hits and the duration of the search project is by clicking 'refine…' and selecting either 'first n' or 'random n' (in addition, there is the option of 'all', which is also the default). The default number of texts that is consulted under the 'first n' or 'random n' options is 1000, but this can be adjusted by clicking on the '1000' and changing the number.

Start running your search project by clicking 'Start.' The progress and any error messages will appear in the yellow box at the bottom of the screen. If for some reason you want to cancel the search, click 'Stop.'

7 Search results

If the search project has been executed without any errors, a 'Results' button appears on the screen. Clicking on it will send you to the Results center, see Figure 21. The overview includes some basic information about the project, the corpus on which the project was run, some statistics about the search, and the number of hits that were found.

The overview also contains a button to **download** the results as an Excel file. This allows using all the excellent sort and filter functions available in Excel to review results based on the output features that have been specified in the search.

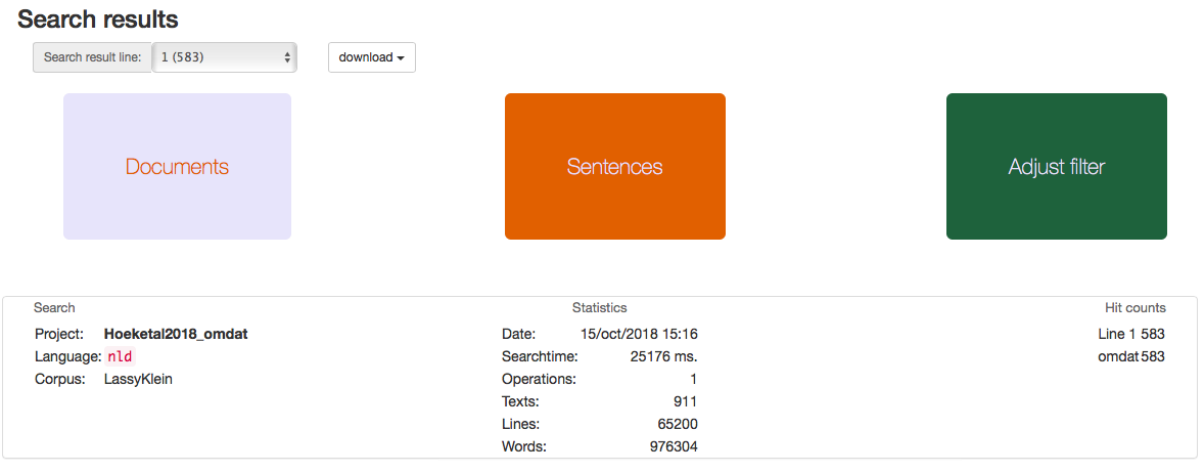


Figure 21. Results center

7.1 View results

You can choose view the results either by document, in which case all search hits are embedded under their document name, or by sentence, in which case all search hits are given in a single KWIC (Keyword in Context)-view list.

In both the by-document view and the by-sentence view, there are three ways to inspect search hits more closely: full results view ( ), tree view ( ), and table view ( ). In the full results view, you see the search hit within in context (sentence), along with a list of all specified output features. The tree view gives you the sentence containing the search hit in a syntactic tree. The table view gives you the sentence containing the search hit in a table, essentially a schematic equivalent of the syntactic tree. The table starts out with the entire sentence along with the syntactic node at the highest level of the syntactic tree, but can be made more detailed by clicking '+' at the very top or at any specific level.

7.2 Adjust filter

While downloading results and opening them in Excel allows for filtering, sorting and searching, Cesar too allows for some filtering. Pressing "adjust filter" opens a filter-specification window where one or more filter lines can be defined.

Each filter line is numbered and consists of the following parts:

Operator:	Take 'first' for the first line in the filter. Other values are 'and' and 'and not'.
Filter field:	Specify which field the filter value is for. There are a few 'standard' fields that allow filtering on text metadata (date, author and so forth). The fields starting with "ft_" are output features.
Value:	The (string) value on which filtering is to take place. This may contain the wildcard * as usual.

Be sure the 'Save' the filter before continuing to the result center.
The result center will show the specifics of the current filter that is being used.

> **Note:** Filters in the current version of Cesar only apply to the 'Sentences' view (not to the 'Documents' view)

7.3 Export results

You can export your results by clicking the 'Download' button and selecting a format in which to export it: Excel, CSV, XML, or SQlite. Your download will start immediately.

7.4 Revisit or delete results

All results are automatically saved for you. To revisit or delete the results from a previous search, go to 'Search – Results' and locate the search you are looking for in the list. Click on the icon to view results; click on the icon to remove results.

> **Note:** Deleting an entire project also automatically deletes all results generated by this project.

8 Other options

The main project page contains two buttons not yet discussed: ‘Summary’ and ‘Download’. You can also upload projects.

8.1 Summary

Clicking 'Summary' will give you an overview of the entire search project with basic information about the project and lists of all global and data-dependant variables, conditions, and output features, including overviews of each function in simple view.

8.2 Download

Clicking 'Download' will start a download of the entire project in JSON format.
Retaining a download of a Cesar project can be part of the '**research data management**' strategy for the research project of which the Cesar search is a part. Downloaded JSON projects can also be sent in as e.g. course assignment results.

8.3 Upload

Projects that have been downloaded in JSON format can be exchanged (e.g. by emailing them) and they can uploaded by other users. To upload a project, navigate to the list of searches by selecting Search >> List_of_searches in the top navigation area of Cesar, see Figure 22. Then click the green upload button to the top-right.

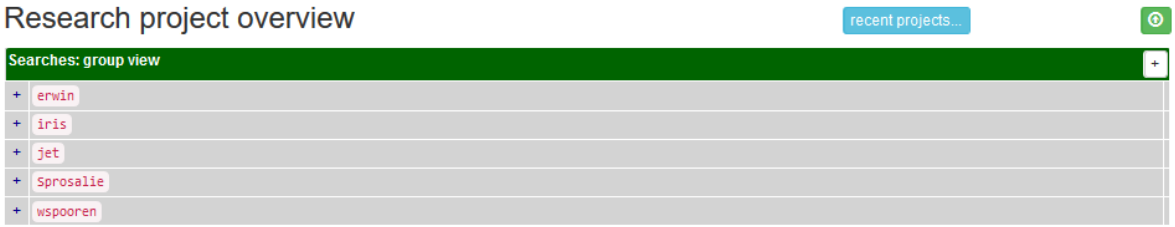


Figure 22. Uploading a research project

Clicking this button opens a dialogue where the location of the JSON file on your computer can be specified and the actual upload take place.

9 Sample projects

The CESAR web portal comes with several fully functional search projects that can be viewed and copied by all users. These projects serve as examples of what search projects look like. In addition, they can be used as a basis for developing other projects by either copying the entire project and adapting it to your specific research purpose or by downloading individual variables in JSON format and incorporating them in your own search project. We developed a simple project and a very complex project (in several versions).

For both the simple and the complex project, we give a short overview of the goals of the search project and the variables used to accomplish these goals. As such, these brief overviews serve as a guideline for finding existing variables or combinations of variables to be used in new search projects that use the CESAR web portal to research discourse coherence.

9.1 Simple project: Identifying 'maar' used as a connective

Project: Maar_sampleproject under JET

This simple project finds connective uses of *maar* (i.e., when it means 'but'), as in (1), and excludes other (adverbial) uses of *maar*, as in for instance (2).

(1) Ik hou heel erg van snoep, **maar** het is zo slecht voor je tanden.
I really love candy, **but** it is so bad for your teeth.
(2) Ik heb **maar** twee snoepjes gegeten.
I **only** had two pieces of candy.

The project accomplishes this in the following way (variable names in red):

- It searches for 'maar' under Search elements
- It defines a global variable 'Conjunction' that includes all POS tags that correspond to 'conjunction' (VG|VG*)
- It defines a fixed variable POStag that identifies the POS tag of each 'maar' found in the corpus
- It includes a condition IsConjunction that checks whether the POS tag of 'maar' corresponds to a 'conjunction' 'conjunction' (VG|VG*)

Running this project thus gives users a set of corpus examples of 'maar' used as a conjunction (save for any parsing errors). This a simple project can be set up very quickly (by an experienced user in a matter of minutes), but constitutes an enormous timesaver in a scenario where a student or researcher would otherwise have to manually filter out adverbial uses of *maar* to arrive at a comprehensive dataset.

9.2 Complex project: Identifying, segmenting, and determining the subjectivity of causal coherence relations

Project: Hoeketal2018_daarom under JET
Hoeketal2018_dus
Hoeketal2018_omdat
Hoeketal2019_want

All of the above projects are slight variations of the same project, which automatically finds causal connectives, such as *want* in (3), identifies the segments that are related to each other by the connective, as indicated by the square brackets in (3), and determines the position of the connective in relation to the segments. In addition, it locates and counts the number of subjective adjectives and adverbs, such as *ontzettend* in (3).

(3) [Ik moet vaak mijn tanden poetsen,]$_{S1}$ **want** [ik hou ontzettend van snoep.]$_{S2}$
[I have to brush my teeth often,]$_{S1}$ **because** [I really love candy.]$_{S2}$

Each project focuses on one search element (a causal connective: *daarom*, *dus*, *omdat*, or *want*). Each project contains many global and fixed variables. Initially, one project was

created, after which it was adapted for the four individual connectives. This is why some variables differ between connectives and why each project contains de-activated conditions.

All four projects share a common goal, which is accomplished by similar means. Variables that are not listed below, but that are nevertheless part of one of the projects above, are either prerequisites for other variables, or variables used exclusively for the calculation of output features. When copying a variable, make sure to also copy any other variables used by the variable you are copying.

The different sub-goals of the above search projects are accomplished in the following way (variable names in red):

Locating causal coherence relations

- Search for a causal connective under 'Search elements'
- Optional: condition on the POS tag of the search hit (see also 8.1)
- Optional: check whether the identified segments are clauses
  - s1nodeVerbal, s2nodeVerbal under 'Data-dependant variables'
  - s1nodeVerbal, s2nodeVerbal under 'Conditions'
- Optional: check whether the search hit is embedded under a PP or a VP
  - bIsPartOfPP, bIsPartOfVP under 'Data-dependant variables'
  - noPartOfPP, noPartOFVC under 'Conditions'

Determining the connective's position in the sentence

- Determine the number of words preceding the search hit in the sentence
  - bIni under 'Data-dependant variables'
- Determine whether the number of words preceding the search hit in the sentence is 0, in which case the search hit is labeled as "ini", or more than 0, in which case the search hit is labeled as "medi".
  - ini_type under 'Data-dependant variables'
- Optional: include only connectives that are located between the two segments in the results file (since not all connectives can appear before both segments, this can help make datasets maximally comparable)
  - MediOnly under 'Conditions'

Locating the discourse segments related to each other by the connective

- Determine the first segment (e.g., S1 in [3])
  - s1 under 'Data-dependant variables'
  - s1 under 'Output features'
- Determine the second segment (e.g., S2 in [3])
  - s2 under 'Data-dependant variables'
  - s2 under 'Output features'

Locating all subjective adverbs and adjectives in the segments related to each other by the connective

- Specify which adverbs and adjectives should be counted as subjective or objective
  - adj_subjective under 'Fixed variables'

- Determine which words in the sentence containing the search hit match a word from the list of subjective words and carry a POS tag that corresponds to either adverb or adjective
  - s1subjective under 'Data-dependant variables'
  - s2subjective under 'Data-dependant variables'
  - lemmas_s1_subjective under 'Output variables'
  - lemmas_s2_subjective under 'Output variables'
- Count the number of words in the sentence containing the search hit match a word from the list of subjective words and carry a POS tag that corresponds to either adverb or adjective
  - count_s1_subjective under 'Output variables'
  - count_s2_subjective under 'Output variables'

10 Citation information

When referring to CESAR, please use the following reference. Please also use this citation when you have used the CESAR web portal in your corpus analysis.

- Komen, E.R. & Hoek, J. (submitted). *Automatic coherence analysis for non-programmers*.

For an example of a study that used the CESAR interface for discourse analysis (using the search project listed in 8.2), see:

- Hoek, J., Sanders, T.J.M., & Spooren, W.P.M.S. (submitted). *Automatic coherence analysis of Dutch: Testing the subjectivity hypothesis on a larger scale*.

Below is a list with the citation information for all corpora that can be accessed through CESAR. When you have accessed a corpus through the CESAR web portal, please cite both CESAR and all corpora you have consulted.

Dutch corpora

CGN – Corpus Gesproken Nederlands

Oostdijk, N. (2002). The design of the Spoken Dutch Corpus. In Peters, P. Collins, P., & Smith, A. (Eds.), *New Frontiers of Corpus Research* (pp. 105-112). Amsterdam: Rodopi.

LassyKlein

Taalunie. 2016. Lassy Klein-corpus. https://ivdnt.org/downloads/tstc-lassy-klein-corpus.

Noord, Gertjan van, Ineke Schuurman & Vincent Vandeghinste. 2006. Syntactic annotation of large corpora in STEVIN. pdf.

NRC 2011

Spooren, W.P.M.S., Hoek, J., Komen, E.R., Hulsbosch, M.A. & Heuvel, H. van den. 2018. *NRC2011.* DANS EASY [Dataset]. url.

SoNaR

Oostdijk, N., Reynaert, M., Hoste, V. & Schuurman, I. (2013). The construction of a 500-million-word reference corpus of contemporary written Dutch. In: Spyns, P. & Odijk, J. (Eds.), *Essential speech and language technology for Dutch: Theory and applications of natural language processing* (pp. 219-247). Berlin: Springer.

Van Noord, G., Bouma, G., Van Eynde, F., de Kok, D., van der Linde, J., Schuurman, I., Tjong Kim Sang, E., & Vandeghinste, V. (2013). Large scale syntactic annotation of written Dutch: Lassy. In: Spyns, P. & Odijk, J. (Eds.), *Essential Speech and Language Technology for Dutch: Theory and Applications of Natural Llanguage Processing* (pp. 147-164). Berlin: Springer.

VU-DNC

Vis, K., Sanders, J., & Spooren, W. (2012). Diachronic changes in subjectivity and stance - a corpus linguistic study of Dutch news texts. *Discourse, Context & Media*.

WhatsApp corpus Lieke (also known as 'Verheijen')

Verheijen, L., & W. Stoop (2016). Collecting Facebook posts and WhatsApp chats: Corpus compilation of private social media messages. In P. Sojka et al. (Eds.), *Text, Speech and Dialogue: 19th International Conference, TSD 2016*, LNAI 9924 (pp. 249–258). Springer.

Spooren, W.P.M.S., Verheijen, L., Hulsbosch, M.A., Komen, E.R. & Heuvel, H. van den. 2018. *Whatsapp corpus Berntzen.* DANS EASY [Dataset]. url.

WhatsApp corpus Manon (also known as 'Berntzen')

Spooren, W.P.M.S., Berntzen, M., Hulsbosch, M.A., Komen, E.R. & Heuvel, H. van den. 2018. *Whatsapp corpus Berntzen.* DANS EASY [Dataset]. url.

English corpora

Granger, Sylviane & Inge de Cock. 2018. Louvain corpus of native English essays. url.

Haan, Pieter de, Inge de Mönnink & Jan Aarts. 2018. The international corpus of learner English - Dutch. url.

ICLE. 2018. The international corpus of learner English. url.

Kroch, Anthony, Beatrice Santorini & Ariel Diertani. 2009. *Penn-Helsinki Parsed Corpus of Early Modern English*. url.

Kroch, Anthony & Ann Taylor. 2009. *Penn-Helsinki Parsed Corpus of Middle English, second edition.* url.

Kroch, Anthony, Beatrice Santorini & Ariel Diertani. 2010. *Penn parsed corpus of modern British English*. url.

Minovska, Vladimira. 2018. The international corpus of learner English - Czech. url.

Taylor, Ann, Athony Warner, Susan Pintzuk & Frank Beths. 2003. The York-Toronto-Helsinki Parsed Corpus of Old English Prose. *Electronic texts and manuals available from the Oxford Text Archive*. url.

Taylor, Ann. 2009. *The York-Toronto-Helsinki Parsed Corpus of Old English Prose* (Syntactic Annotation Reference Manual). University of York. url.

Caucasian corpora

Komen, Erwin R. 2015. *The Nijmegen Parsed Corpus of Modern Chechen (NPCMC)*. Nijmegen, Netherlands: Radboud University Nijmegen. url.

Komen, Erwin R. 2015. *The Nijmegen Parsed Corpus of Modern Lak (NPCML)*. Nijmegen, Netherlands: Radboud University Nijmegen. url.

11 Appendix

The appendices contain information that are meant for reference for the user.

11.1 Tag sets of corpora used in CESAR

The table below contains links

Table 4. Links to tag sets used in CESAR

Corpus	Syntactische annotatie
Lassy = LassyKlein = Sonar = NRC2011 = VU-DNC = Whatsapp	http://urd.let.rug.nl/~vannoord/Lassy/sa-man_lassy.pdf or consult: http://nederbooms.ccl.kuleuven.be/test/tags
CGN	http://lands.let.ru.nl/cgn/doc_Dutch/topics/version_1.0/annot/syntax/syn_prot.pdf or consult: https://www.tonvanderwouden.nl/index_files/papers/NT2002-vdw-etal.pdf
Caucasian: Chechen Lak	http://erwinkomen.ruhosting.nl/che/crp/NCPMC-annotation.htm
English: historical	https://www.ling.upenn.edu/ppche/ppche-release-2016/annotation/index.html
English: SLA	Adapted from Stanford parser, see https://catalog.ldc.upenn.edu/docs/LDC99T42/tagguid1.pdf

11.2 Empty categories

Empty categories (e.g. `*con*`, `*pro*`) are annotated only in some of the corpora, such as the historical English ones and the Caucasian ones. Note that these empty categories also occur in the **Dutch** corpora, which have all been 'surfaced': discontinuous constituents have been divided into continuous parts, and the relation between them makes use of empty categories like `*T*` (trace) and `*ICH*` (interpret constituent here).

11.3 Overview of built-in functions

The functions that are available to the CESAR users can be broken up into a few groups.

Group	Section	Goal
Text-producing	11.3.1	Yield a string as output.
Calculating	11.3.2	Calculate a number. The number can be interpreted as string.
Testing	11.3.3	Test a condition and return 'true' or 'false'.
One constituent	11.3.4	Locate one particular constituent
More Constituents	11.3.5	Locate all constituents matching a criterion
Word node(s)	11.3.6	Locate one (or more) word nodes
Selection		Select one of the cases based on conditions

The subsections below discuss the functions in more detail.

11.3.1 Text-producing functions

There are a few functions that have as their task to produce text (a string of letters) based on the characteristics of one or more constituents.

Name:	**get_cat**
Argument(s):	cns (constituent)
Goal:	get the syntactic label (category) of the cns.

The constituent can be found by choosing a data-dependant constituent, by choosing the 'search hit' (which always is a constituent) or by calculating a constituent using one of the 'constituent-producing functions' (see X).

Name:	**concat_3**
Argument(s):	arg1, arg2, arg3 (strings)
Goal:	Combine strings arg1, arg2, arg3 into one new string.

Name:	**feature**
Argument(s):	cns (constituent) name (string)
Goal:	If constituent cns has a feature called name, then get the (string) value of that feature. Otherwise an empty string is returned.

Name:	**get_cns_text**
Argument(s):	cns (constituent)
Goal:	get the words of the child-constituents of cns, including the syntactic categories of these child constituents.

Name:	**get_text**
Argument(s):	cns (constituent)
Goal:	get the text of all the words in cns.

Name:	**get_wordcat_list**
Argument(s):	cns (constituent)
Goal:	Make a space-separated list of all the part-of-speech tags of the *words* within the constituent cns.

11.3.2 Calculating functions

There are a few funtions that provide a numerical output. This output can be used as string, but it can also be used as numerical input to some functions.

Name:	**ant_distance**
Argument(s):	cns (constituent)
Goal:	Provided the text that is searched contains coreference information, give the number of 'lines' the antecedent finds itself with relation to cns. This function can be used in some of the historical English texts.

Name:	**cns_count**
Argument(s):	cns (constituent)
Goal:	Count the number of child constituents in cns. If cns is a list of objects or constituents, then **cns_count** returns the number of items in that list.

Name:	**relative_wordcount**
Argument(s):	rel (integer) cns (constituent) exc (string)
Goal:	Count the number of words that have a relation named rel towards constituent cns. Exclude those that have a POS-tag that matches the pattern in exc.

Name:	**subtract**
Argument(s):	first (integer) second (integer)
Goal:	Calculate first - second.

Name:	**wrd_count**
Argument(s):	cns (constituent)
Goal:	Count the number of words in cns.

11.3.3 Test functions

The test functions look at their argument(s) and the return either 'true' or 'false'. Some test functions just combine the results of other tests (e.g. and, or_2, or_3). Other test functions look for different criteria.

Test functions that only *combine* the results of other tests:

Name:	**and**
Argument(s):	arg1 (boolean) arg2 (boolean)
Goal:	Return true if both arg1 as well as arg2 are true.

Name:	**not**
Argument(s):	arg (boolean)
Goal:	Return the opposite of arg: true becomes false, false becomes true

Name:	**or_3**
Argument(s):	arg1 (boolean) arg2 (boolean) arg3 (boolean)
Goal:	Return true if either arg1 or arg2 or arg3 is true.

Name:	**or_2**
Argument(s):	arg1 (boolean) arg2 (boolean)
Goal:	Return true if either arg1 or arg2 is true.

Test functions that have their own criteria:

Name:	**cns_equals**
Argument(s):	cns1 (constituent) cns2 (constituent)
Goal:	Return 'true' if cns1 is the same constituent as cns2.

Name:	**current_relates**
Argument(s):	rel (comparison relation: before, after, equal) cns (constituent)
Goal:	Return 'true' if the current constituent has a relation rel with cns.

Name:	**exists**
Argument(s):	cns (constituent)
Goal:	Return 'true' if constituent cns exists and is not empty.

Name:	**has_cat**
Argument(s):	cns (constituent) arg2 (string)
Goal:	Return 'true' if the constituent cns has a syntactic category arg2.

Name:	**has_lemma**
Argument(s):	cns (constituent) arg2 (string)
Goal:	Return 'true' if the *lemma* of the constituent cns matches the pattern arg2

Name:	**has_relation**
Argument(s):	cns (constituent) rel (relation) cat (string)
Goal:	Return 'true' if the constituent related with rel to cns has a syntactic category that matches cat. This could e.g. be used to see if the parent (rel is x:parent) of constituent cns has a particular syntactic category.

Name:	**has_relation2**
Argument(s):	cns (constituent) rel1 (relation) cat1 (string) rel2 (relation) cat2 (string)
Goal:	Evaluate three levels of relations towards cns:

	Level 1: all constituents with relation `rel1` towards `cns`, provided the syntactic categories of these constituents match `cat1`. Level 2: all constituents with relation `rel2` towards `level1`, provided the syntactic categories of these constituents match `cat2`. Return 'true' if there are any constituents on `level2`.

Name:	**has_relation3**
Argument(s):	`cns` (constituent) `rel1` (relation) `cat1` (string) `rel2` (relation) `cat2` (string) `rel3` (relation) `cat3` (string)
Goal:	Evaluate three levels of relations towards `cns`: Level 1: all constituents with relation `rel1` towards `cns`, provided the syntactic categories of these constituents match `cat1`. Level 2: all constituents with relation `rel2` towards `level1`, provided the syntactic categories of these constituents match `cat2`. Level 3: all constituents with relation `rel3` towards `level2`, provided the syntactic categories of these constituents match `cat3`. Return 'true' if there are any constituents on `level3`.

Name:	**has_relation_with**
Argument(s):	`cns1` (constituent) `rcond` (relation) `cns2` (constituent)
Goal:	Return 'true' if constituent `cns1` has a relation `rcond` (*precedes*, *follows*, *equals*) with constituent `cns2`.

Name:	**is_equal**
Argument(s):	`arg1` (string) `arg2` (string)
Goal:	Return true if the string `arg1` is completely equal to the string `arg2`. This function can be used to check if e.g. the syntactic category of a constituent is equal to some string.

Name:	**matches**
Argument(s):	`arg1` (string) `arg2` (string)
Goal:	Return true if the string `arg1` fits the string or the pattern `arg2`. The pattern `arg2` may contain wildcards such as the asterisk '*'

Name:	**word_follows**
Argument(s):	`cns1` (constituent) `cns2` (constituent)
Goal:	The constituent or word `cns1` occurs in the text somewhere *after* `cns2`

Name:	**word_precedes**
Argument(s):	`cns1` (constituent) `cns2` (constituent)
Goal:	The constituent or word `cns1` occurs in the text somewhere *before* `cns2`

11.3.4 Select one constituent

A key element of CESAR is the ability to locate constituents within sentences. That is why CESAR has a wide variety of functions aimed at locating *one* particular constituent. All these functions need to start with another constituent. That constituent can be the 'search hit', a constituent identified with a previous data-dependant variable, or the output of another function in section 11.3.4.

Name:	**antecedent**
Argument(s):	`cns` (constituent)
Goal:	Get the constituent to which `cns` refers back. Note: antecedents can only be found if a corpus has been annotated for coreference. There are very few corpora that have this.

Name:	**cns_at_pos**
Argument(s):	`lst` (constituent list) `pos` (position: `1`, `2`, … or '`last()`')
Goal:	Return the constituent at position `pos` within the list of constituents `lst`. This list of constituent should be calculated by one of the functions treated in section 11.3.5. Note: the notion 'last()' can be a tricky one. Wherever a list is created separately, the 'last()' node is taken to be the one that is furthest down in the text. So even if a list is created by selecting 'preceding' nodes, the 'last' one still is the last one in the physical text.

Name:	**get_first_cns_cat**
Argument(s):	`cns` (constituent) `rel` (relation) `skip` (string) `cat` (string)
Goal:	Take the first constituent with relation `rel` towards `cns` that has syntactic category `pos`. But skip all constituents that fulfill any of the following: (a) it is a punctuation sign (b) it has an end-node with one of the empty categories[1] (c) its syntactic category matches `skip`

Name:	**get_first_relative_cns**
Argument(s):	`cns` (constituent) `rel` (relation) `cat` (string)
Goal:	Take the first constituent with relation `rel` towards `cns` that has syntactic category `cat`. Note: nothing is skipped. If empty categories must be skipped, use `get_first_cns_cat`, and leave `skip` empty.

Name:	**get_last**
Argument(s):	`lst` (constituent list)
Goal:	Return the last constituent within the list of constituents `lst`. This list of constituent should be calculated by one of the functions treated in section 11.3.5. Note: the notion 'last()' can be a tricky one. Wherever a list is created separately, the 'last()' node is taken to be the one that is furthest down in the text. So even if a list is created by selecting 'preceding' nodes, the 'last' one still is the last one in the physical text.

Name:	**get_line**
Argument(s):	`arg` (integer)
Goal:	Return the whole syntactic tree in the line following (positive `arg`) or preceding (negative `arg`). The function `get_line(-3)`, for instance, gives the syntax tree of three lines back.

[1] Empty categories (e.g. `*con*`, `*pro*`) are annotated only in some of the corpora, such as the historical English ones and the Caucasian ones. Note that these empty categories also occur in the **Dutch** corpora, which have all been 'surfaced': discontinuous constituents have been divided into continuous parts, and the relation between them makes use of empty categories like `*T*` (trace) and `*ICH*` (interpret constituent here).

Name:	**get_related_at_pos**
Argument(s):	rel (relation) cns (constituent) pos (position: 1, 2, … or 'last()') *default: 1* nopunc (boolean) *default: true* skip (string)
Goal:	Put all constituents with relation rel towards cns into a list. Take the one at position pos in that list. Skip constituents whose syntactic category matches skip. If nopunc is 'true', then also skip: (a) punctuation sign constituents (b) constituents that have an end-node with one of the empty categories[2]

Name:	**get_relative**
Argument(s):	rel (relation) cns (constituent) cat (string)
Goal:	Take the first constituent with relation rel towards cns that has syntactic category cat. Note: this is the same as get_first_relative_cns()

Name:	**get_relative_cnd**
Argument(s):	rel (relation) cns (constituent) cnd (boolean)
Goal:	Take the first constituent with relation rel towards cns provided that cnd is true. Note: the condition cnd can be fixed to 'true' or 'false', or it can be the output of one of the testing functions in section 11.3.3.

Name:	**get_relative_word**
Argument(s):	rel (relation) cns (constituent) cnd (boolean) pos (position: 1, 2, … or 'last()')
Goal:	Create a list of constituents with relation rel towards cns where condition cnd is true. Return the constituent at position pos from this list. Notes: the position 'last()' can be a tricky one. Wherever a list is created separately, the 'last()' node is taken to be the one that is furthest down in the text. So even if a list is created by selecting 'preceding' nodes, the 'last' one still is the last one in the physical text.

[2] Empty categories (e.g. *con*, *pro*) are annotated only in some of the corpora, such as the historical English ones and the Caucasian ones. Note that these empty categories also occur in the **Dutch** corpora, which have all been 'surfaced': discontinuous constituents have been divided into continuous parts, and the relation between them makes use of empty categories like *T* (trace) and *ICH* (interpret constituent here).

	The condition `cnd` can be fixed to 'true' or 'false', or it can be the output of one of the testing functions in section 11.3.3.

Name:	**`getAncNonEmpty`**
Argument(s):	`cns1` (constituent) `rcond` (constituent) `cns2` (constituent)
Goal:	Get the first *ancestor* (higher constituent) of `cns1`, provided that this ancestor has at least one descendant that follows after constituent `cns2` (e.g. the descendant *precedes*, *follows* or *equals* `cns2`).

Name:	**`getAncNonEmptyAfter`**
Argument(s):	`cns1` (constituent) `cns2` (constituent)
Goal:	Get the first *ancestor* (higher constituent) of `cns1`, provided that this ancestor has at least one descendant that follows after constituent `cns2`. Note: this is the same as function **`getAncNonEmpty`**, but with `rcond` being *follows*.

Name:	**`parent_if`**
Argument(s):	`cns` (constituent) `cat` (string)
Goal:	Return the parent of constituent `cns` provided it has a syntactic category that matches `cat`.

11.3.5 Select many constituents

There are a number of functions that find groups of constituents with respect to some other constituent. That last constituent can be the 'search hit', a constituent identified with a previous data-dependant variable, or the output of a function in section 11.3.4.

Name:	**`get_related`**
Argument(s):	`rel` (relation) `cns` (constituent) `cnd` (boolean)
Goal:	Get all the constituents with relation `rel` towards `cns` (or the first in the list of `cns` is a list), provided `cnd` is true. Note: The condition cnd can be set to 'true' or 'false', or it can be calculated by any of the functions discussed in 11.3.3.

Name:	**get_related_2**
Argument(s):	rel1 (relation) cns (constituent) cat1 (string) rel2 (relation) cat2 (string)
Goal:	Get all the constituents with relation rel1 towards cns, where two conditions hold: (a) the syntactic category matches cat1, and (b) there is at least one constituent with relation rel2 and syntactic category matching cat2.

Name:	**get_related_cat**
Argument(s):	rel (relation) cns (constituent) cat (string)
Goal:	Get all the constituents with relation rel towards cns (or the first in the list of cns is a list), provided their syntactic category matches cat.

Name:	**get_related_exc**
Argument(s):	rel (relation) cns (constituent) cat (string)
Goal:	Get all the constituents with relation rel towards cns (or the first in the list of cns is a list), provided their syntactic category does **not** match cat.

Name:	**get_related_list** = **get_related_simple**
Argument(s):	rel (relation) cns (constituent)
Goal:	Get all the constituents with relation rel towards cns.

Name:	**if_exists**
Argument(s):	cns1 (constituent / list) cns2 (constituent / list)
Goal:	If constituent cns1 exists, then return cns1 otherwise return cns2.

Name:	**if_exists_2**
Argument(s):	cns1 (constituent / list) cns2 (constituent / list) cns3 (constituent / list)
Goal:	If constituent cns1 exists, then return cns1 otherwise if constituent cns2 exists, then return cns2 otherwise return cns3.

Name:	**lemma_cat_list**
Argument(s):	`lst` (constituent list) `lem` (string) `cat` (string)
Goal:	Go through the list `lst` and return all the constituents in that list that have a lemma matching `lem` and that have a syntactic category matching `cat`.

11.3.6 Select a word node

Many corpora distinguish between syntactic constituents and word nodes. The FoLiA and the psdx encoded corpora available in CESAR make such a distinction, each in its own way, as exemplfied in Figure 23.

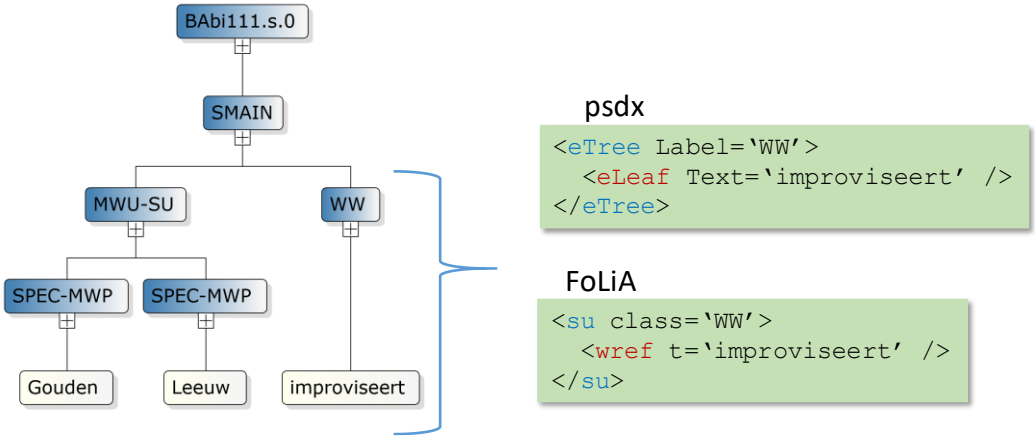


Figure 23. Word nodes in psdx *and* FoLiA

The 'word node' (the `eLeaf` for psdx, the `wref` for FoLiA) physically contains just the word 'improviseert' itself. The part-of-speech tag of the word (or its syntactic category) 'WW' is encoded as attribute of the enclosing node (an `eTree` in psdx, an `su` in FoLiA).

While this distinction can be disregarded in most cases (e.g. by using `get_text` or `get_cat` on a syntactic node to get the node's text or syntactic category), users may sometimes find it easier to be working with the actual word nodes. This is why a few functions in CESAR provide such word nodes as output. There are no functions that must work with word nodes as input.

Name:	**get_first_relative_wrd**
Argument(s):	`cns` (constituent) `rel` (relation) `cat` (string)
Goal:	Take the first word node with relation `rel` towards `cns` where the word node's host constituent has syntactic category `pos`.

Name:	**get_related_exc_w**
Argument(s):	`rel` (relation) `cns` (constituent) `cnd` (boolean)
Goal:	Get all the word nodes with relation `rel` towards `cns` (or the first in the list of `cns` is a list), excluding those for which `cnd` is true.

	Note: The condition cnd can be set to 'true' or 'false', or it can be calculated by any of the functions discussed in 11.3.3.

Name:	**get_related_w**
Argument(s):	`rel` (relation) `cns` (constituent) `cnd` (boolean)
Goal:	Get all the word nodes with relation `rel` towards `cns` (or the first in the list of `cns` is a list), provided `cnd` is true. Note: The condition cnd can be set to 'true' or 'false', or it can be calculated by any of the functions discussed in 11.3.3.

11.3.7 Conditional selection

There are a number of functions that allow choosing between two possibilities, based on a particular condition. The output of these functions is of the same type as the input of the function. So that can be a number, a string, a boolean, a constituent or a list of constituents.

Name:	**if_else**
Argument(s):	`cnd` (boolean) `use1` (anytyhing) `use2` (anything)
Goal:	If condition `cnd` holds, then return `use1`, otherwise return `use2`.

Name:	**if_case_3**
Argument(s):	`cnd1` (boolean) `use1` (anytyhing) `cnd2` (boolean) `use2` (anything) `use3` (anything)
Goal:	If condition `cnd1` holds, then return `use1`, otherwise check if condition `cnd2` holds; if so, then return `use2`, if all else failes, then return `use3`.

Name:	**if_case_5**
Argument(s):	`cnd1` (boolean) `use1` (anytyhing) `cnd2` (boolean) `use2` (anything) `cnd3` (boolean) `use3` (anything) `use4` (anything)
Goal:	If condition `cnd1` holds, then return `use1`, otherwise check if condition `cnd2` holds; if so, then return `use2`, otherwise check if condition `cnd3` holds; if so, then return `use3`, otherwise check if condition `cnd4` holds; if so, then return `use4`,

	if all else failes, then return `use5`.

Name:	**`if_equals_else`**
Argument(s):	`arg1` (string) `arg2` (string) `use1` (anytyhing) `use2` (anything)
Goal:	If string (or integer, or boolean) `arg1` equals `arg2`, then return `use1`, otherwise return `use2`.