

# **Corpus Studio**

## **The web application**



*Version 1.2*  
*First edition: June 30, 2015*  
*This edition: July 31, 2015*

Erwin R. Komen  
Meertens Instituut // CLARIN-NL // SIL-International

# Contents

1	Introduction.....	3
2	Getting started.....	4
2.1	Requirements .....	4
2.2	First use.....	4
3	Corpora .....	5
4	Projects.....	5
4.1	The “Project” page.....	6
4.2	The “Input” page .....	6
4.3	The “Definitions” page.....	6
4.4	The “Queries” page .....	6
4.5	The “Pipeline” page.....	6
4.6	Query execution.....	6
4.7	The “Results” page .....	8
4.7.1	The results overview .....	8
4.7.2	Result details: “per hit” .....	10
4.7.3	Result details: “per document” .....	11
5	Bibliography .....	12
6	Appendix.....	12
6.1	Schema for xml formats used .....	13
6.2	Useful Xquery function definitions .....	13
6.2.1	Convert a chain into a sequence .....	13
6.3	Built-in functions for Xquery-psdx projects.....	14
6.3.1	Back.....	14
6.3.2	Conv .....	14
6.3.3	Feature .....	15
6.3.4	Lex.....	15
6.3.5	Line.....	15
6.3.6	Matches .....	15
6.3.7	NodeText.....	16

# 1 Introduction

The ‘Corpus Studio’ web application is a web-orientend environment that aims to facilitate in-depth quantitative syntactic research for linguists. It does so by supporting researchers in writing queries that operate on syntactically parsed *xml* text corpora. Queries that belong together are kept in *xml* documents that are called ‘Corpus Research Projects’ (CRPs). These documents contain the queries, the order in which they are to be executed, meta-information about the queries and the project as a whole, as well as a specification of the input used for the project. The use of CRPs should improve the replicability of corpus research.

Potential users of the Corpus Studio web application should be aware of what the program offers and what its limitations are.

- (1) **Texts** – While the *xml* query engine is, in principle, able to support queries on any kind of *xml*-encoded texts, the current version has been developed to handle two *xml* encodings: (a) FoLiA, and (b) psdx.
- (2) **Corpora** – The CLARIN-NL version of the Corpus Studio web application offers a limited number of syntactically annotated corpora to users. Please consult the program’s “corpora” page for an overview of them. Wishes to add texts (or corpora) can be communicated to CLARIN-NL.
- (3) **Versions** – Developers are allowed to deploy the web version of Corpus Studio on their own servers in order to provide users access to the corpora they offer.

The query language used in the Corpus Studio web application is “Xquery” (Boag et al. 2010). The particular implementation of Xquery currently used is a freely available one developed by Saxon. The Xquery language (and its associated Xsl as well as Xpath) is a well guided public domain initiative for general research work in XML databases. When corpora are coded in XML, then Xquery provides one of the most generally accepted ways to query them.

The basic unit in CorpusSearch is the *Corpus Research Project*, containing the following elements:

- a) General information about a corpus research project, such as date, author, purpose.
- b) All the queries used by the project.
- c) The order in which the queries are to be used.
- d) The output files produced by successive application of the queries.
- e) A list of input files to be used for the queries.

Since the corpus research projects comprise all the data needed to perform a particular task on a (selectable) set of input files, they offer many advantages. Some of them are:

- **Exchange** of corpus research projects between researchers. It will be easy to see just how your colleague has dealt with the data.
- Assistance in **teaching** courses on corpus research.
- A form for students to hand in **assignments** on corpus research work.
- An easier way to track **errors** in your corpus research projects.

## 2 Getting started

Since the Corpus Studio web application is web-based, the requirements to run it are minimal. This document mainly concentrates on the CLARIN-NL version of the Corpus Studio web application, which facilitates a limited number of corpora. Those interested to use the Corpus Studio web application for other text corpora should read on in section **TODO:REF**.

### 2.1 Requirements

No installation is necessary to work with the Corpus Studio web application. What is needed is a browser such as Internet Explorer (version 9 or higher), FireFox, Chrome or Safari. The web application only uses HTML and JavaScript on client computers. The URL of the Corpus Studio web application can be found by visiting the CLARIN-NL project page:

<http://www.clarin.nl/node/2091>

### 2.2 First use

To get to the CLARIN-NL [project page](#), one has to login as a CLARIN user. Clicking through to the Corpus Studio web application's page brings one to another login request. This is the login facility of Corpus Studio. Any CLARIN user can choose to make use of Corpus Studio Web through one or more named accounts (see **TODO:REF**).

For now it suffices to pick a user login name of your liking, and a new password that accompanies it. Once a password has been used for the first time, it will automatically be checked upon subsequent login attempts.

The main menu of the web program consists of:

- **Corpora**. Find out more about the corpora that are currently available within the CLARIN-NL version of the Corpus Studio web application. (Note: it is possible to install other corpora in one's own, custom, version of the web program; see **TODO:REF**.)
- **Projects**. This is the place where CRPs (Corpus Research Projects) can be loaded, edited and executed. The results of executing a CRP also appear in this location..
- **About**. Background information about the program and a bibliographic reference that can be used to refer to it.

### 3 Corpora

The corpora that are currently supported by the CLARIN-NL Corpus Studio web application are listed in Table 1.

*Table 1 Corpora available in the CLARIN-NL version of Corpus Studio*

Collection	Language	Type	Notes
<b>YCOE</b>	Old English	psdx	This is the York-Toronto-Helsinki corpus of Old English that has been enriched with several features (e.g. lemma's and PGN).
<b>PPCME2</b>	Middle English	psdx	The Penn Parsed Corpus of Middle English, converted to <i>xml</i> and enriched with features.
<b>PPCEME</b>	Early Modern English	psdx	The Penn Parsed Corpus of Middle English, converted to <i>xml</i> and enriched with features.
<b>PPCMBE</b>	Modern British English	psdx	The Penn Parsed Corpus of Middle English, converted to <i>xml</i> and enriched with features.
<b>CGN</b>	Spoken Dutch	psdx, folia	The “Corpus Gesproken Nederlands” (Corpus of spoken Dutch), converted from tiger (negra) to psdx and folia xml.
<b>NPCMC</b>	Chechen (NE Caucasian)	psdx, folia	The Nijmegen Parsed Corpus of Modern Chechen. This was created in psdx, but a folia conversion is available.
<b>PCMLBE</b>	Lak (NE Caucasian)	psdx, folia	The Parsed Corpus of Modern Lak. This is a small corpus of Lak texts in the Cyrillic script that is gradually being annotated.

### 4 Projects

The “Projects” menu item leads to a number of different sub-menu's, all of which are intended to facilitate working with CRPs (corpus research projects). The general work-flow is as follows:

- 1) **Project.** Select, upload or create a project (CRP).
- 2) **Input.** Define the language and corpus that needs to be searched.
- 3) **Definitions.** Edit definitions of variables and functions.
- 4) **Queries.** Edit the queries for this project.
- 5) **Pipeline.** Order the queries.
- 6) **Execute.** BUTTON: execute the queries.
- 7) **Results.** View the results of the queries.

Selecting the “Projects” menu leads the user to the “Project” page (step 1), where a project should be selected. The meta-data of a selected project can be edited on this page too. The selection of a corpus that serves as input takes place on the “Input” page (step 2). Changes in the definitions, queries or query order can be done on the pages “Definitions” (step 3), “Queries” (step 4) and “Pipeline” (step 5). Once a Corpus Research Project has been fully defined, execution of the project on the selected corpus can be instantiated by clicking the “Execute Queries” button. The page “Results” (step 7) becomes available as soon as query execution has finished. The results page allows detailed inspection of the query results, as well as exporting of data.

## 4.1 The “Project” page

### TEST VERSION:

The test version of the Corpus Studio web application can be used by logging in as “guest” (password “crpstudio”). Good combinations of CRPs and corpora are:

CRP	Corpus	Goal
<b>ChechenMarkerA</b>	Chechen NPCMC	Finds the particle ‘a’, which can be used for focus, negation, conjunction and subordination
<b>cheFPq</b>	Chechen NPCMC	Finds clitics/particles “m” and “q”, related to focus and topic
<b>DemRef</b>	Historical English (any)	
<b>LakProDem</b>	Lak PCMLBE	Finds pronouns and demonstratives
<b>ObjectRC_versie1</b>	Historical English (any)	Relative clauses on main-clause objects
<b>ObjectSubjectRC_versie1</b>	Historical English (any)	Relative clauses on subjects and objects
<b>V2_test_versie11</b>	Historical English (any)	Syntactic features for main clauses with overt subjects and finite verbs

## 4.2 The “Input” page

## 4.3 The “Definitions” page

## 4.4 The “Queries” page

## 4.5 The “Pipeline” page

## 4.6 Query execution

There are two buttons to execute the queries. The ‘normal’ button that says “Execute queries” tries to make optimal use of information that has already been collected during previous executions. If the CRP has not changed, the available results will be shown instantaneously.



Figure 1 Two query execution buttons

The button labelled “Excute queries (no cache)” executes the queries completely from the start. Use this button if there is reason to believe something went wrong on your computer or on the server.

The normal reaction to pressing the **Execute** button is the revealing of two progress indicators: one keeping track of the files that have been turned over to the query execution engine, and one that keeps track of the files that have been completed.

Should there be an error in the Xquery code of one of the queries, this will usually become apparent immediately. An error message like the one in Figure 2 will appear.

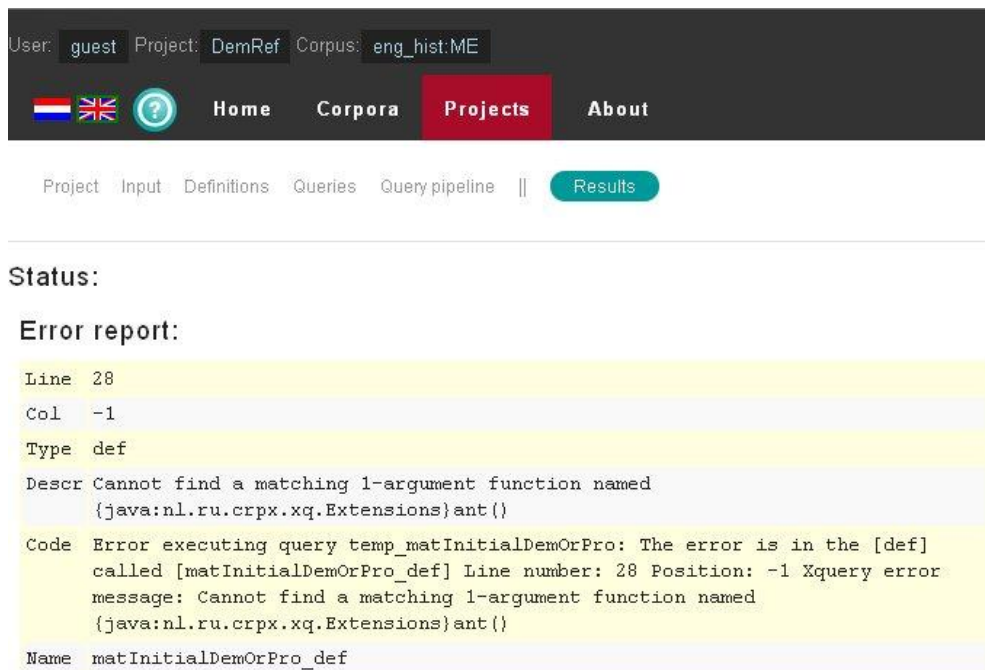


Figure 2 An error in the Xquery code

The error report identifies the name of the query or definition (see “Name”, last row) where the error has occurred, and the line number (see “Line”) within the query or definition. The combination of the location with that of the description of the error should be enough to identify the problem and adapt the program accordingly.

The current error says that the Xquery engine cannot find a function named `ru:ant()` that takes just one argument. The reason for this is that this function has not yet been implemented in the version of CorpusStudio. Note: if the function `ru:ant()` would have existed, but it would have required *two* arguments, the same error message would have been given.

## 4.7 The “Results” page

When query execution leads to results, the “Results” page is automatically selected, and a concise overview of the number of hits is shown, as in Figure 1.

#	Zoekregel	Subcategorie	Aantal	Aanpassen	Details
1	matVf+NPS	(all together)	48776	EDIT	
2	matV2q	(all together)	43320	EDIT	

Figure 3 Concise overview of the results

The overview in Figure 1 shows the number of hits for each of the two query lines: 48776 for line 1 (matVf+NPS) and 43320 for line 2 (matV2q).

### 4.7.1 The results overview

There are a few actions that can be undertaken from this overview table:

- 1) **Export.** Provide the overview as a tab-separated text file.
- 2) **Edit.** Turn to the query editor of the selected line.
- 3) **Selection.** Select one of the QC (query constructor) lines by clicking it.

The **Export** button here yields a tab-separated table with an overview of the number of hits for both the query lines as well as the sub categories:

```
date: 2015-07-31 13:45:03

overview table




QC  Result Label  Category  Number
1   matVf+NPS    (all together)  48776
2   matV2q       (all together)  43320
2   matV2q       1:[iSbj-XP-Vf]  2333
2   matV2q       1:[iSbj-iVf]   25219
2   matV2q       2:[XP-Sbj-YP-Vf]  714
2   matV2q       3:[XP-Sbj-iVf]  13307
2   matV2q       4:[Vf-Sbj-Vn]  511
2   matV2q       4:[Vf-Sbj]     1182
2   matV2q       5:[Vf-Vn-Sbj]  54
```

The **Edit** button opens the “Queries” page and turns to the selected Query.



Pressing the red subcategory or frequency in the overview table results in the **Selection** of that Query line. If the query has results that are divided over different sub categories (see **TODO: crossref**), then selecting the Query line results in an adapted overview that shows the hit frequencies for each of the categories. These category-lines can also be selected.

Gebruiker: **guest** Project: **V2\_test\_versie11** Corpus: **eng\_hist:ME**

   **Home** **Tekstverzamelingen** **Projecten** **Info**

Project Zoekdoel Definities Zoekopdrachten Volgorde || **Resultaten**

**Zoekregels:** Search time: **17.077 s.** **Exporteer**

#	Zoekregel	Subcategorie	Aantal	Aanpassen	Details
1	matVf+NPS	(all together)	72184		
2	matV2q	(all together)	60474		
2	matV2q	1:[iSbj-XP-Vf]	2504		
2	matV2q	1:[iSbj-IVf]	29121		
2	matV2q	2:[XP-Sbj-YP-Vf]	937		
2	matV2q	3:[XP-Sbj-IVf]	17275		
2	matV2q	4:[Vf-Sbj-Vn]	2491		
2	matV2q	4:[Vf-Sbj]	7590		<b>DETAILS</b>
2	matV2q	5:[Vf-Vn-Sbj]	556		

*Figure 4 Selection of a category line*


The selection of either a line (either of a query or of a category) causes a **Details** button to appear. Clicking this button opens a window with the detailed results for the selected query-category combination.


## 4.7.2 Result details: “per hit”

The details of the results can be viewed in a number of different ways: (a) hit-by-hit, (b) hits-per-document, (c) hits-per-group and (d) hits-per-division. The “hit-by-hit”, as illustrated in Figure 5, is the default one.

Figure 5 Result details per hit

The detailed results information offers the following possibilities:

- 1) **Back.** The  button returns to the general result table overview (see 4.7.1).
- 2) **Pagination.** Adjust number of hits per page, and select the page to be shown.
- 3) **Export.** Export the information shown on this page.

The results are, generally speaking, not shown all together, but they are divided into *pages*. The number of hits per page can be adjusted in the “Show *nn* per page” selector, while the “Page *nn* of *mmm*  ” combination allows one to select the page to be shown.

The **Export** button causes a tab-separated text file to be made that contains the results shown in the currently selected page. Each line in the tab-separated text file contains the following eight fields:

<i>n</i>	line number
<i>file</i>	name of the text file
<i>locS</i>	location of the sentence
<i>locW</i>	location of the constituent
<i>preC</i>	context preceding the hit
<i>hitC</i>	the hit itself (the part of the sentence returned by the Xquery <code>ru:back()</code> function)
<i>folC</i>	context following the hit
<i>hits</i>	a syntactic break-up of the hit

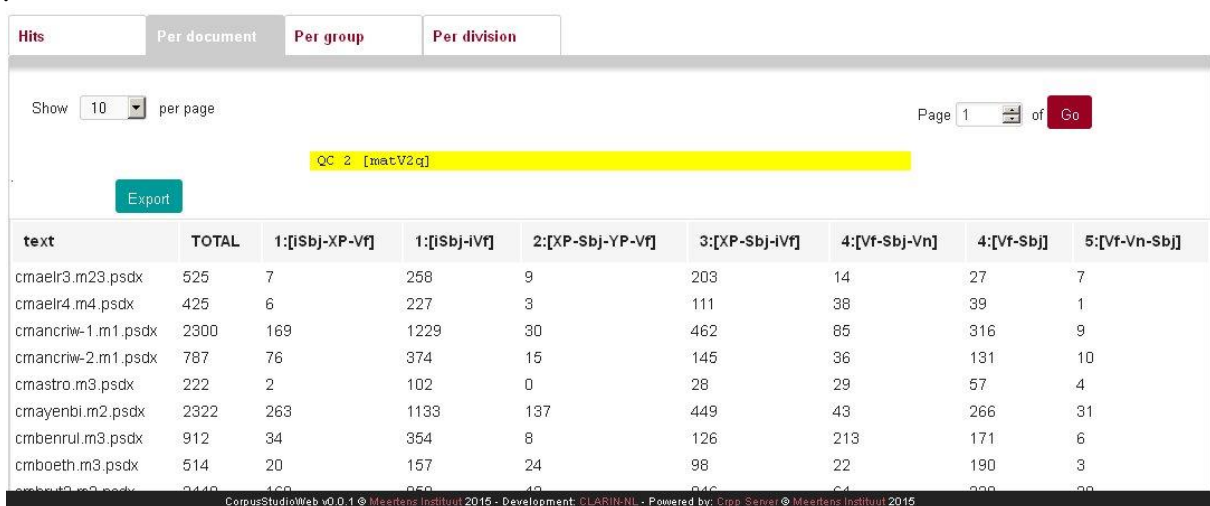
### 4.7.3 Result details: “per document”

Selection of the “Per document” tab within the “Results” page gives an overview of the results that is grouped per document. When one category has been selected in the Results overview page, the frequencies are shown per document for that category (Figure 6), but when the query line has been selected as a whole, the frequencies for all the categories are shown in a table (Figure 7).



text	4:[Vf-Sbj]
cmaelr3.m23.psdX	27
cmaelr4.m4.psdX	39
cmancr1w-1.m1.psdX	316
cmancr1w-2.m1.psdX	131
cmastro.m3.psdX	57
cmayenbi.m2.psdX	266
cmbenrui.m3.psdX	171
cmboeth.m3.psdX	190
cmboeth.m3.psdX	222

Figure 6 Result details per document: one category selected



text	TOTAL	1:[Sbj-XP-Vf]	1:[Sbj-IVf]	2:[XP-Sbj-YP-Vf]	3:[XP-Sbj-IVf]	4:[Vf-Sbj-Vn]	4:[Vf-Sbj]	5:[Vf-Vn-Sbj]
cmaelr3.m23.psdX	525	7	258	9	203	14	27	7
cmaelr4.m4.psdX	425	6	227	3	111	38	39	1
cmancr1w-1.m1.psdX	2300	169	1229	30	462	85	316	9
cmancr1w-2.m1.psdX	787	76	374	15	145	36	131	10
cmastro.m3.psdX	222	2	102	0	28	29	57	4
cmayenbi.m2.psdX	2322	263	1133	137	449	43	266	31
cmbenrui.m3.psdX	912	34	354	8	126	213	171	6
cmboeth.m3.psdX	514	20	157	24	98	22	190	3
cmboeth.m3.psdX	2442	169	859	19	816	64	222	22

Figure 7 Result details per document: a query line has been selected

The **Export** button in the “per document” view results in a tab-separated file to be made that contains the frequencies of the selected results, e.g:

```
date:                2015-07-31 15:21:31
QC line:             matV2q
Sub category:        4:[Vf-Sbj]
Count:              7590
cmaelr3.m23.psdX    525      27
cmaelr4.m4.psdX    425      39
cmancriw-1.m1.psdX 2300     316
cmancriw-2.m1.psdX 787      131
cmastro.m3.psdX    222      57
cmayenbi.m2.psdX   2322     266
cmbenrul.m3.psdX   912      171
cmboeth.m3.psdX    514      190
cmbrut3.m3.psdX    2448     238
```

The top lines in the exported file contain general information: the selected query line, the sub category and the total number of hits for this sub category (which can be verified with the table produced from the general results overview).

Subsequent lines contain the information per document: the total number of hits for the document for this query line, and the number of hits taken up by the currently selected sub category. This allows for the calculation of relative frequencies.

The output that has been produced by the query can be inspected for each individual document, by selecting that document. Selection of, for example, “cmaelr4.m4.psdX” in Figure 6 shows all the hits for query line 2 (“matV2q”), category “4:[Vf-Sbj]” and file “cmaelr4.m4.psdX”, as in Figure 8.

Figure 8 Result details for one document

## 5 Bibliography

## 6 Appendix

This appendix contains vital information for those who would like to process the results of a corpus research project themselves (see **TODO:REF**), and also for those who would like to

know more about the different xml formats used for corpus research projects, Xquery-psdx etc (see **TODO:REF**).

## 6.1 Schema for xml formats used

CorpusStudio makes use of several different *xml* formats. Each of these formats is described by a schema, an *xsd* file. These schema are available on the CorpusStudio [homepage](#) which you can reach from the Radoud University website. Table 2 gives an overview of the different schema files that should be available.

Schema file	Root xml tag	Usage
<b>CrpResult.xsd</b>	<CrpOview>	
<b>CorpusResearchProject.xsd</b>	<CorpusResearchProject>	
<b>PeriodDef.xsd</b>	<PeriodList>	
<b>Psd.xsd</b>	<TEI>	

*Table 2 Xml schema definitions in use for CorpusStudio*

## 6.2 Useful Xquery function definitions

You are, of course, at liberty to make your own Xquery functions, but there are a number of functions you may find useful, and these are included in this section. Some of these may actually be available already on the homepage of CorpusStudio.

Note that some of these functions use built-in Xquery functions which are described in section 6.3. Since they make use of the CorpusStudio in-built facilities, such user-defined Xquery functions will obviously not work outside of CorpusStudio.

### 6.2.1 Convert a chain into a sequence

You might want to convert a following or preceding coreferential chain into a sequence of nodes, which you can then process using the standard Xquery sequence processing FLOWR functionality. The following Xquery function recursively converts a chain into such a sequence, and it starts at the node you provided.

```
(: -----
  Name : tb:Chain
  Goal : Get all the nodes on the chain started by [$ndThis]
  History:
    03-04-2012 ERK    Created
  ----- :)
declare function tb:Chain($ndThis as node()*) as node()*
{
  (: Get the next element -- if existing :)
  let $nxt := if (count($ndThis)=1) then ru:chnextidt($ndThis)
              else ru:chnextidt( ($ndThis/.)[last()])

  (: Combine the new node with the others :)
  return
    if (exists($nxt)) then $ndThis union tb:Chain($nxt)
    else $ndThis
};
```

### 6.3 Built-in functions for Xquery-psdx projects

Several Xquery functions have been built into CorpusStudio that should facilitate working with Xquery-psdx projects. The functions are described in this section. All functions are part of the **ru** section.

#### 6.3.1 Back

Definition: **ru:back**(*ndArg1*, *strMsg*, *strCat*)

Types: *ndArg1* node()  
*strMsg* xs:string (optional)  
*strCat* xs:string (optional)  
 output node()

Description: This built-in function is an alternative to the **tb:Forest** functions that are supplied in the standard “definitions” file. The function should be placed at the end of the main `for ... let ... where ... return` (FLOWR) loop, as part of the `return` statement. The function returns the `<forest>` node, supplied with additional attributes, such as the identifier of the *ndArg1* node. There are two optional (string) variables that can be supplied. The variable *strMsg* may contain a string (produced for example using Xquery standard **concat**) with information that is supplied with the output of this particular result. The variable *strCat* may contain a subcategorization string. The output for this particular queryline will then, in addition to the normal tabular output, be subdivided over the values of *strCat*. One may, for example, take NP features such as the **NPtype** or the **GrRole** as subcategorization values. A user implementation that can be used instead of the built-in `ru:back` function should minimally look as follows:

```
declare function tb:back($ndThis as element()?) as element()
{
  (: Get the <forest> element of which we are part :)
  let $src := $ndThis/ancestor-or-self::forest
  (: Get the ID of ourself :)
  let $id := $ndThis/@Id

  (: Copy the attributes to a new forest element :)
  return element forest { attribute TreeId { $id }, $src/@* }
};
```

#### 6.3.2 Conv

Definition: **ru:conv**(*strText*, *strType*)

Types: *strText* xs:string  
*strType* xs:string → ‘OE’, ‘Lcase’, ‘Ucase’, ‘clean’  
 output xs:string

Description: The string *strText* is converted on the basis of the `convert` type specified in *strType*. Two conversion types are currently supported. The first one, signalled by “OE”, converts +a, +t, +d etc into their corresponding unicode strings æ, þ, and ð respectively. The second and third one, signalled by “Lcase” and “Ucase”, convert the string into its lower-case or upper-case equivalent respectively.

A few combinations are allowed too: ‘Lcase+OE’ and ‘Ucase+OE’.



### 6.3.3 Feature

Definition: **ru:feature**(ndArg1, strName)

Types: ndArg1 node()  
 strName xs:string  
 output xs:string

Description: Get the feature value of the feature named **strName**. This is equivalent to `ndArg1/child::fs/child::f[@name='strName']/@value`. The string with this value is returned.

### 6.3.4 Lex

Definition: **ru:lex**(strWord, strPos)

Types: strWord xs:string  
 strPos xs:string  
 output xs:boolean

Description: Add the word **strWord** to a list of lexicon/dictionary entries, keeping track of the frequency. Forms with different part-of-speech tags (**strPos**) are kept separate. The resulting dictionary can be opened using Cesax.  
 Words taken from the English historical corpora that are passed on to the lexicon are advised to be treated first through **ru:conv**(..., 'Lcase+OE'). This makes sure the lexemes are kept case-insensitive and the possible leading \$-signs are filtered out.

### 6.3.5 Line

Definition: **ru:line**(intNumber)

Types: intNumber xs:integer  
 output node()

Description: Return the sentence that is **intNumber** lines further away from me (if positive) or before me (if negative). The “sentence” is a `<forest>` element for the *psdx* projects.  
 A special case is **ru:line**(0), which returns the current `<forest>` element in such a way, that the preceding and following forests (as well as the hierarchically higher nodes) can be accessed.

### 6.3.6 Matches

Definition: **ru:matches**(strText, strPattern)

Types: strText xs:string  
 strPattern xs:string  
 output xs:boolean

Description: The string **strText** is compared with the pattern supplied by **strPattern**. The behaviour is much like the visual basic function `like`. The **strPattern** can contain a set of different patterns, separated by a vertical bar “|”. Here is an example: `ru:matches("NP-POS-3", "NP|NP-*")`. This function will return *true*, since `NP-POS-3` matches with the second pattern in the list: `NP-*`.

Definition: **ru:matches**(strText, strPatYes, strPatNo)

Types: strText xs:string  
 strPatYes xs:string  
 strPatNo xs:string  
 output xs:boolean

Description: The string **strText** should match the pattern supplied by **strPatYes**, but should not match that of **strPatNo**. The behaviour is much like the visual basic

function `like`. The `strPatYes` and `strPatNo` can contain a set of different patterns, separated by a vertical bar “|”. Here is an example: `ru:matches("NP-POS-3", "NP|NP-*", "NP-VOC")`. This function will return *true*, since `NP-POS-3` matches with the second pattern in the list: `NP-*`, and it does *not* match with the pattern `NP-VOC`.

### 6.3.7 NodeText

Definition: `ru:NodeText(ndArg1)`

`ru:NodeText(ndArg1, strType)`

Types: `ndArg1` node()

`strType` xs:string → ‘OE’, ‘Lcase’, ‘Ucase’, ‘clean’

output xs:string

Description: Return the text of the terminal nodes within `ndArg1`. The text is delivered as it is, separated by spaces where needed. A second argument `strType` allows implicit use of the function `ru:conv` to streamline the output.

See also: `ru:PhraseText`