Corpus Studio

the web application

Version 1.6
First edition: June 30, 2015
This edition: December 23, 2015

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

Contents

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 text corpora in a number of major *xml* formats. 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 helps 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. There is some support for Alpino (the alpino_ds format).

(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-coded texts and 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, user-defined functions and variables used by the project.

c) The hierarchical order in which the queries are to be executed.

d) The names and order of the database features, should a query lead to the creation of a 'Result Database'.

e) A directory or 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 advantages over the traditional approach of using command-line functions for corpus research. To name but a few of the advantages:

- **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.
- CRPs are a form for students to hand in **assignments** on corpus research work.

- An easier way to track **errors** in corpus research projects, since all the steps that are taken to find the constructions one is looking for are together in one place.

2 Getting started

The Corpus Studio web application is web-based, which means that 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 **8.1**.

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 section 3).

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 8.1)
- **Databases**. Collections of results created by Projects. Handling them is a future option for the CorpusStudio web application.
- **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.

2.3 This documentation

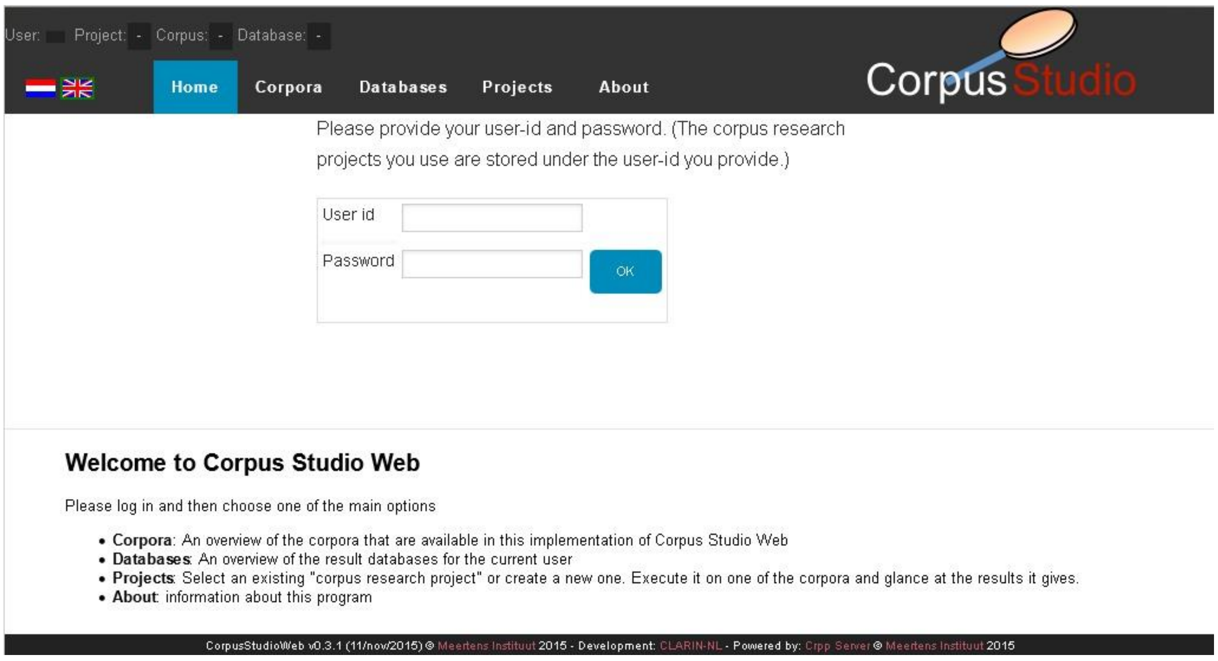
This document is to be used as background an information where the application itself is not clear. The web application of CorpusStudio relieves the user from the necessity to keep track of input and output directories, locations of corpora and so on. This is an improvement with respect to the Windows version of CorpusStudio.

A good starting point for an overview is section 5, which runs through all the essential parts of the **Projects** main menu.

Several specific tasks are dealt with in sections **Fout! Verwijzingsbron niet gevonden.** and **Fout! Verwijzingsbron niet gevonden.**.

3 Users and accounts

Accessing the CorpusStudio web application brings one to a login page (Figure 1).



Please provide your user-id and password. (The corpus research projects you use are stored under the user-id you provide.)

Welcome to Corpus Studio Web

Please log in and then choose one of the main options

- **Corpora**: An overview of the corpora that are available in this implementation of Corpus Studio Web
- **Databases**: An overview of the result databases for the current user
- **Projects**: Select an existing "corpus research project" or create a new one. Execute it on one of the corpora and glance at the results it gives.
- **About**: information about this program

CorpusStudioWeb v0.3.1 (11/nov/2015) © Meertens Instituut 2015 - Development: CLARIN-NL - Powered by: Crpp Server © Meertens Instituut 2015

Figure 1 Logging in to CorpusStudio

Any CLARIN-NL user may have one or more user names to log in to the web application. Each user account has a separate space on the web application's server. Since it is in this space that a user's Corpus Research Projects and Corpus Result Databases are kept, making use of multiple user names allows one to keep projects separated.

4 Corpora

The corpora menu is, first of all, meant to give a quick glance at the corpora that are supported by the CorpusStudio web application. Future releases of CorpusStudio will see more uses for the Corpora menu, some of which are already hinted at in subsequent sections.

4.1 The “Corpora/Explore” page

The **Explore** page of the Corpora menu gives a birds-eye view of the corpora that are currently supported by the CLARIN-NL Corpus Studio web application.

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

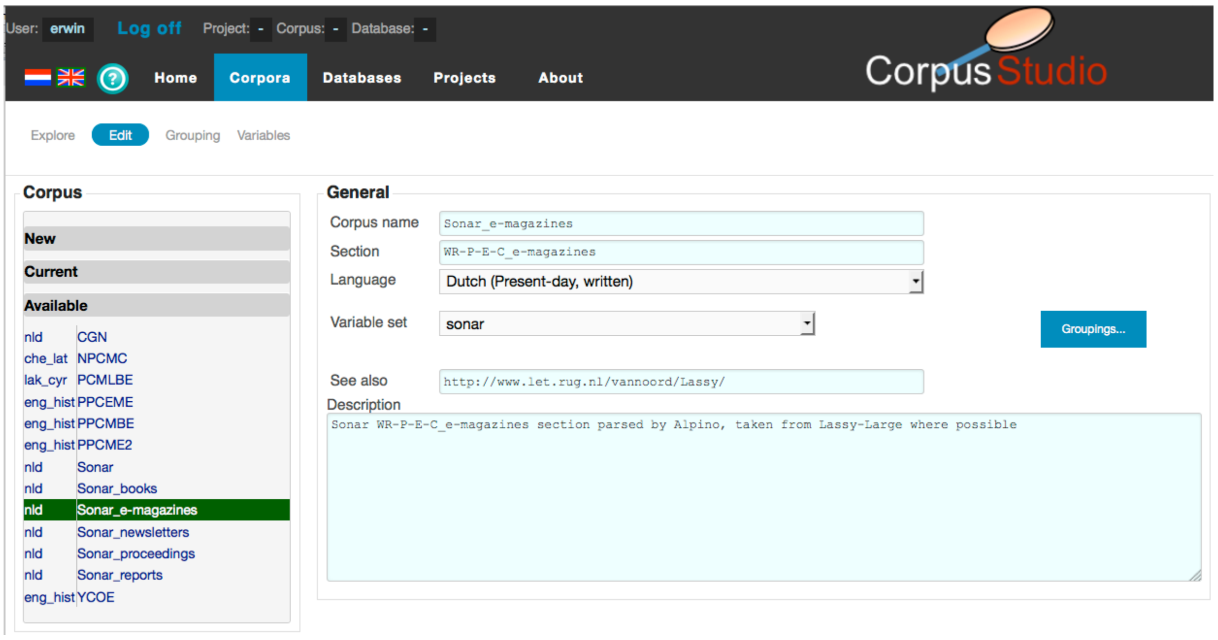
Collection	Language	Type	Notes
Sonar	Written Dutch	folia	A selection of sections from Sonar500 that have been parsed by Alpino (taken from Lassy-large where available). The following sections are included: WR-P-P-B_books WR-P-E-C_e-magazines WR-P-P-J_proceedings WR-P-P-K_reports WR-P-P-D_newsletters
CGN	Spoken Dutch	psdx, folia	The “Corpus Gesproken Nederlands” (Corpus of spoken Dutch), converted from tiger (negra) to psdx and folia xml.
NPCMC	Chechen (NE Caucasian)	psdx, folia	The Nijmegen Parsed Corpus of Modern Chechen. This was created in psdx, but a folia conversion is available.
PCMLBE	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.

The corpora listed above are available to everyone. Individual users may have access to more corpora, depending on the settings an these user’s rights.

Navigating to the “Corpora” section of the web application brings one to a set of sub menu’s: “Explore”, “Edit”, “Grouping” and “Variables”. These menu’s are currently under construction and are aimed to contain valid functionality in future releases.

4.2 The “Corpora/Edit” page

The **Edit** page lists all the corpora available to the current user, providing the details for every corpus in the ‘General’ form part.



nld	CGN
che_lat	NPCMC
lak_cyr	PCMLBE
eng_hist	PPCEME
eng_hist	PPCMBE
eng_hist	PPCME2
nld	Sonar
nld	Sonar_books
nld	Sonar_e-magazines
nld	Sonar_newsletters
nld	Sonar_proceedings
nld	Sonar_reports
eng_hist	YCOE

Corpus name	Sonar_e-magazines
Section	WR-P-E-C_e-magazines
Language	Dutch (Present-day, written)
Variable set	sonar
See also	http://www.let.rug.nl/vannoord/Lassy/
Description	Sonar WR-P-E-C_e-magazines section parsed by Alpino, taken from Lassy-Large where possible

Figure 2 The corpus information editor

The priviledged user that has administrative rights can edit the information for each corpus, add the information for a new corpus and delete the information of a corpus whose support is discontinued.

The corpus information consists of the following elements:

`Corpus name`	Name (label) for this corpus or corpus-selection
`Section`	Name of the directory (on the server) where this corpus (or selection) resides
`Language`	Choose one of the languages from the options provided
`Variable set`	Select one of the variable-sets as defined in the **Variables** menu
`See also`	A URL pointing to more information about this corpus
`Description`	Any information in prose about this corpus (or selection)

The **Corpora** >> **Edit** menu is an important point for any user, since it provides the starting point for the creation of or editing of a ‘grouping’.

First select a corpus, and then press the Groupings button to arrive at the right selection of groupings in the **Corpora** >> **Grouping** page.

4.3 The “Corpora/Grouping” page

The **Grouping** page is planned to become available in future editions of CorpusStudio. This page allows for the creation and the editing of ‘groupings’ (or ‘divisions’): collections of group-specifications into which the results from each search file should be divided. A grouping matches each file with a group label according to the text file’s meta data information.

4.4 The “Corpora/Variables” page

The **Variables** page is planned to become available in future editions of CorpusStudio. This page allows for the creation and the editing of meta data ‘variables’.

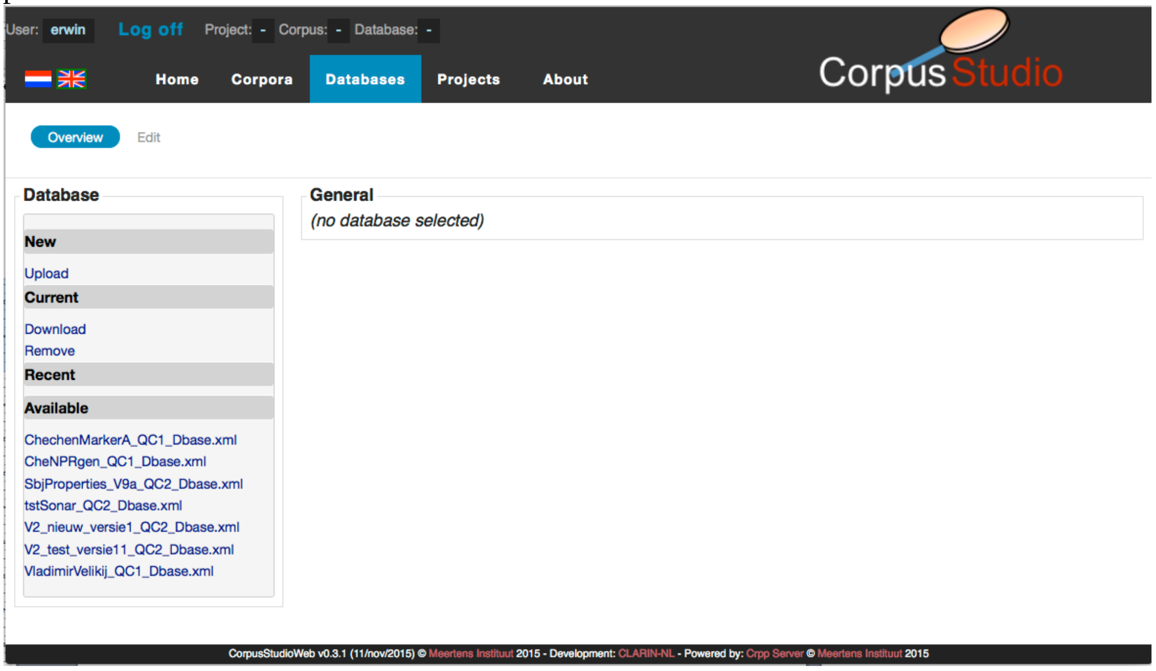
5 Databases

The databases menu serves to inspect, download and upload the ‘Result databases’ available to the user. Result databases can optionally be generated as a result of a query line in a CRP. The databases can also serve as input to a CRP.

The *xml* format of the result databases used in the web application is similar to the one used for the Windows version of CorpusStudio, which means that it should be possible to exchange databases between these versions of the program. It should also be possible to load and edit a corpus Result database in the Windows program ‘Cesax’.

5.1 The “Database/Overview” page

The ‘Overview’ page facilitates inspection of the databases, downloading and uploading them. This functionality will become available in future releases of the CorpusStudio web application.



(no database selected)

ChechenMarkerA_QC1_Dbase.xml
CheNPRgen_QC1_Dbase.xml
SbjProperties_V9a_QC2_Dbase.xml
tstSonar_QC2_Dbase.xml
V2_nieuw_versie1_QC2_Dbase.xml
V2_test_versie11_QC2_Dbase.xml
VladimirVelikij_QC1_Dbase.xml

CorpusStudioWeb v0.3.1 (11/nov/2015) © Meertens Instituut 2015 - Development: CLARIN-NL - Powered by: Crpp Server © Meertens Instituut 2015

Figure 3 Result database overview

5.2 The “Database/Edit” page

There is no ‘Edit’ page in the current version of the web application. Future releases should implement the possibility to view and edit the individual entries of a Result database.

6 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) **Features**. Define and order result database features.
7) **Execute**. BUTTON: execute the queries (available from **Project**).
8) **Results**. View the results of the queries.

Selecting the “Projects” menu leads the user to the “Project” page (step 1), where a project is to 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 (step 7).[1] The page “Results” (step 8) 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.

6.1 The “Projects/Project” page

The “Project” page that comes up under the “Projects” main menu selection serves two purposes: (a) identification of the CRP to work with, and (b) editing of the CRP’s “General” information.

6.1.1 CRP identification

The “Project” page initially comes up with the screen shown in Figure 4.

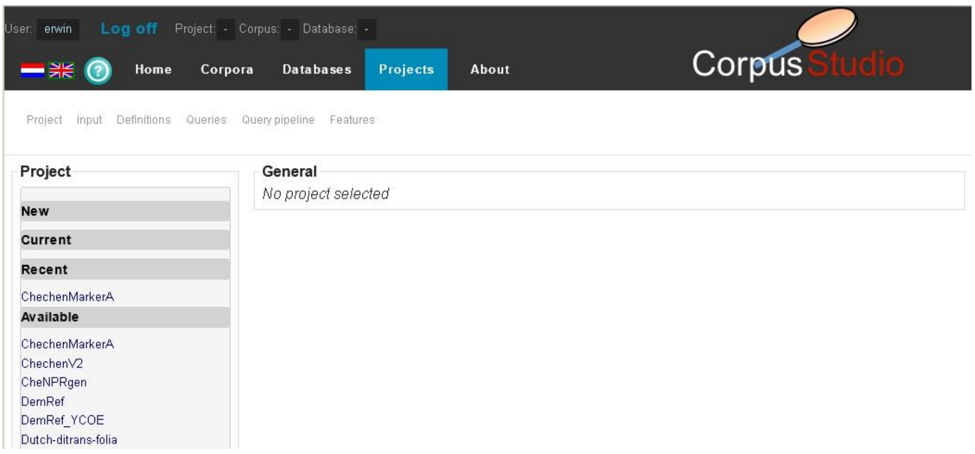


Figure 4 Initial state of the "Project" page

[1] This button is only available in the “Project” page.

The CRP can be chosen from the list on the left side, from the “Available” section. Clicking an existing project will load it and make its information available for inspection and editing.

The project selector contains more options. It consists of four sections, the first two of which have sub sections:

New.	Methods to get a new CRP on the server:	
	Upload.	Upload a CRP to the server
	New project (manual).	Create a new project manually
	Project wizard.	Create a new project with the wizard (see X)
Current.	Actions on the currently selected CRP:	
	Download.	Download the selected CRP from the server
	Remove.	Remove the selected CRP from the server
Recent.	The CRP that has been ‘executed’ last.	
Available.	Select one of the CRPs that have been uploaded to the server.	

When the **New** section is selected, three options to get a new CRP onto the server become available: Upload, New project (manual) and Project wizard. The Project creation wizard is discussed in section X.

As far as the **Current** section’s actions are concerned: please note that both the Download and Remove commands pertain to the CRP that is located **on the server**. So ‘removing’ a CRP within CorpusStudio has no effect whatsoever on a copy of the CRP as it exists on a user’s computer. However, should a CRP have been created on the server, and should the creator not have made a backup of the CRP on his own computer, then ‘removing’ the CRP from the server means that it is lost completely.

The **Recent** section contains just one CRP; this is the CRP that has been executed most recently.

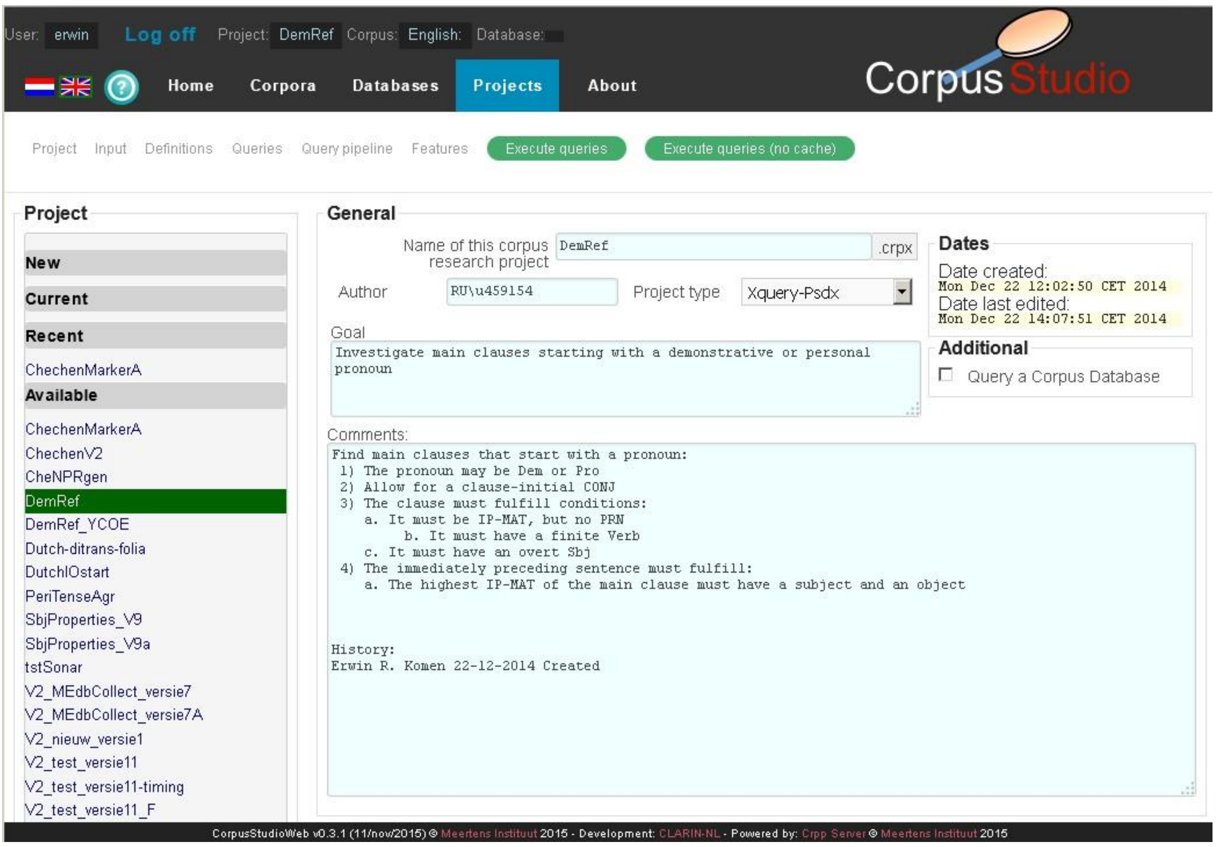
The selection of a CRP that is in the **Available** section makes it the current CRP, which is marked in two ways:

(a) the name of the current CRP appears in the top bar under “corpus”, and

(b) the “General” information of the CRP appears.

6.1.2 The CRP’s “General” information

Figure 5 is an example of the “General” information for one particular CRP.



Project

Recent: ChechenMarkerA

Available: ChechenMarkerA, ChechenV2, CheNPRgen, DemRef, DemRef_YCOE, Dutch-ditrans-folia, DutchIOstart, PeriTenseAgr, SbjProperties_V9, SbjProperties_V9a, tstSonar, V2_MEdbCollect_versie7, V2_MEdbCollect_versie7A, V2_nieuw_versie1, V2_test_versie11, V2_test_versie11-timing, V2_test_versie11_F

General

Name of this corpus research project: DemRef .crpx

Author: RU\u459154

Project type: Xquery-Psdx

Goal:

```
Investigate main clauses starting with a demonstrative or personal
pronoun
```

Comments:

```
Find main clauses that start with a pronoun:
 1) The pronoun may be Dem or Pro
 2) Allow for a clause-initial CONJ
 3) The clause must fulfill conditions:
    a. It must be IP-MAT, but no PRN
         b. It must have a finite Verb
    c. It must have an overt Sbj
 4) The immediately preceding sentence must fulfill:
    a. The highest IP-MAT of the main clause must have a subject and an object



History:
Erwin R. Komen 22-12-2014 Created
```

Dates

Date created: Mon Dec 22 12:02:50 CET 2014

Date last edited: Mon Dec 22 14:07:51 CET 2014

Additional

CorpusStudioWeb v0.3.1 (11/nov/2015) © Meertens Instituut 2015 - Development: CLARIN-NL - Powered by: Crpp Server © Meertens Instituut 2015

Figure 5 The "General" information available to a project

The "General" information of a project is crucial for future reference. It consists of elements that can be edited (these are in the light-blue boxes) and some that the program creates itself. The editable elements are:

`Name`	A succinct non-spaced name identifiying this project
`Author`	Name of the main author
`Project type`	One of the available project types (e.g. Xquery-Psdx, Folia-xml)
`Goal`	What is the goal this project strives to get at?
`Comments`	Any additional information helpful to understand later what this project does

The CRP's general information also contains two items that are non-editable: the creation date of the CRP and the date it has last been changed.

Note: Any **changes** (edits) made by the user in the editable elements (`Name`, `Author`, `Project type`, `Goal`, `Comments`) become effective only after the user has pressed the "Save" button that is to appear once a user starts making changes. There is no automatic saving in the current version.

6.1.3 Test version

The test version of the Corpus Studio web application can be used by logging in as "**guest**" (password "crpstudio"). This account can be accessed by any CLARIN user, but it is only one user at a time that can be logged into it.

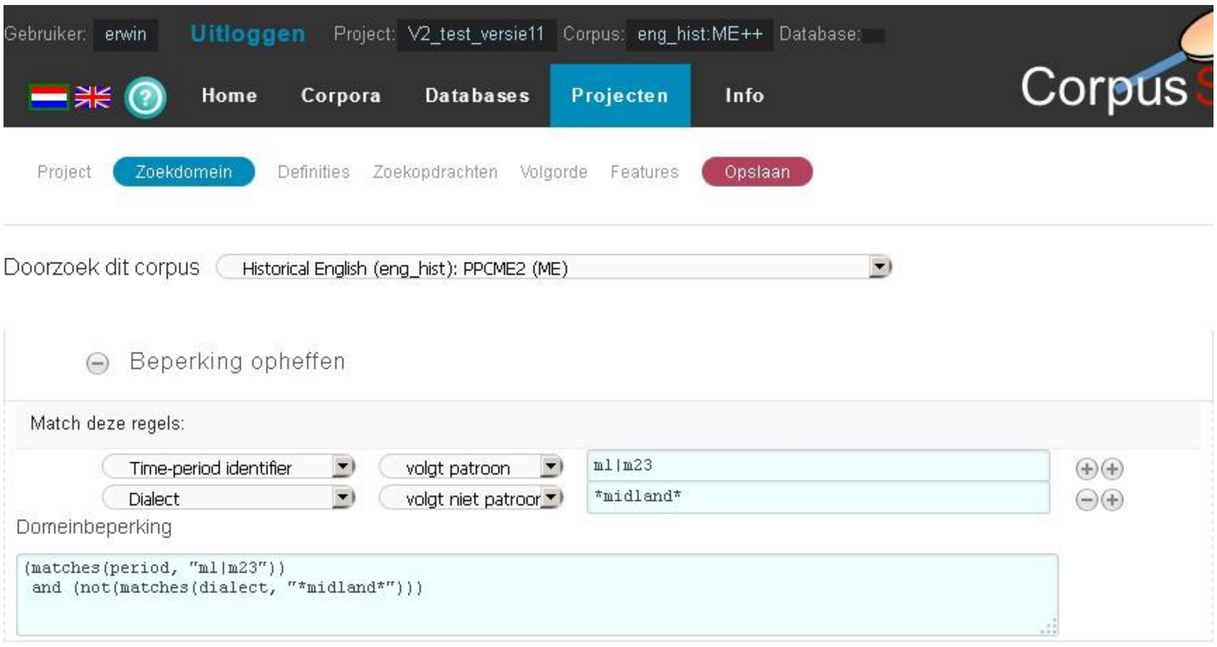
6.2 The “Projects/Input” page

The CorpusStudio program is meant to facilitate researchers in writing and executing queries that take syntactically parsed texts as input. The **Input** page is the place to specify what exactly should serve as input to these queries. There are two possibilities to fine-tune the input:

1) **Corpus selection**. Select a corpus from the selector (combobox) "Corpus to search".
2) **Domain restriction**. Restrict the texts within the selected corpus.

The corpus selection is straight-forward: one can select one of the pre-installed text corpora from the selector. The text corpora have already been divided into main parts where this is possible with the given corpora.

The restriction of texts within a selected corpus can be done on the basis of the meta data supplied in each text.



Doorzoek dit corpus: Historical English (eng_hist): PPCME2 (ME)

Time-period identifier	volgt patroon	m1|m23
Dialect	volgt niet patroon	*midland*

```
(matches(period, "m1|m23"))
 and (not(matches(dialect, "*midland*")))
```

Figure 6 De input selector

The example in Figure 6 shows the project called "V2_test_versie11" as having the "ME" (Middle English) part of the historical English corpora as its main input. A further restriction on the texts that are to be taken as input is defined in the Rules section:

1) **Time period**. The time-period needs to be "m1" or "m23".
2) **Dialect**. The dialect may not contain the word "midland".

Input domain restrictions can be released by clicking "Reset restrictions" (Dutch: Beperking opheffen). Setting domain restrictions can be started by clicking "Restrict input" (Dutch: Domein beperken).

The available meta data such as genre, author and so forth differ from one corpus to the other. This is why the contents of the rule-selectors is changed as soon as a different corpus is selected. Changing the corpus selection also resets all the input restriction rules.

The meta data information of the texts can be compared with the values supplied by the user in the text boxes at the right hand side. This is an overview of the available comparison operators (the middle selectors):

1) `is`	- The meta data in the text equals the value supplied.
2) `is not`	- The meta data differs from the value supplied.
3) `smaller than`	- The meta data is smaller than the value supplied.
4) `smaller than or equal to`	- The meta data is smaller than or equal to the value.
5) `larger than`	- The meta data is larger than the value supplied.
6) `larger than or equal to`	- The meta data is larger than or equal to the value.
7) `matches pattern`	- The meta data matches the pattern supplied.
8) `doesn't match pattern`	- The meta data does not match the pattern.

The 'matching' operator allows using the following special symbols:

`*`	Zero or more characters
`?`	One character of any value
`[a-z]`	One character in the range 'a' until 'z'
`|`	The logical 'or' operator

The rules supplied by the user are internally converted to a piece of Xquery code, which is executed before a text is queried with the main queries. Only a positive evaluation of the input restriction code results in the execution of the main queries.

The set of input matching rules as well as the Xquery code derived from it are part of the *xml* code of the Corpus Research Project. Downloading the project allows one to have a look at the code that is being used.

6.3 The "Projects/Definitions" page

The queries that can be used to search through the texts that have been selected in the "Input" page are made up of four components:

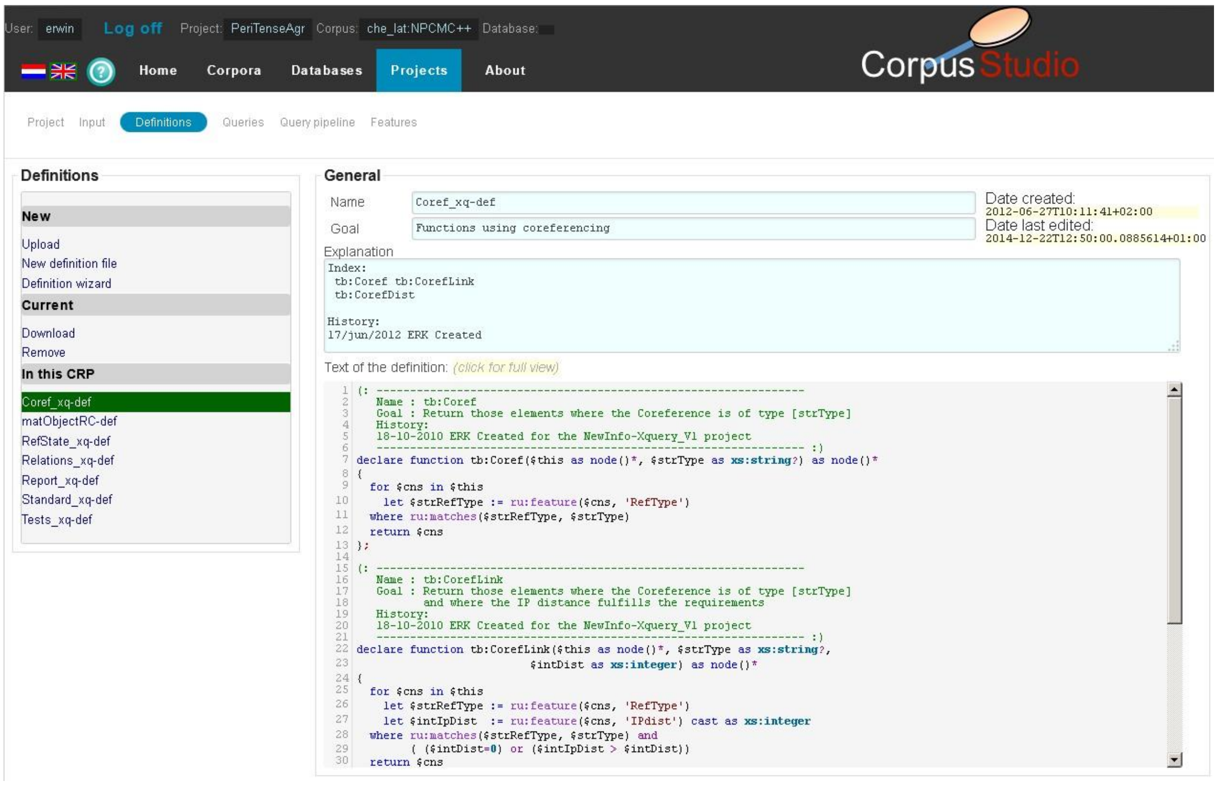
1) **Definitions**	- Variables and functions needed for the queries
2) **Queries**	- The queries themselves
3) **Pipeline**	- The order and hierarchy in which queries need to be executed
4) **Features**	- The (optional) features that need to be calculated for a result database

This section treats the **Definitions**. One or more definition 'files' may be used within one Corpus Research Project. The definition files contain declarations of global variables and definitions of function—all in the Xquery language (Boag et al. 2010; W3C 2010; Yao & Bouma 2010).

The CorpusStudio program combines *all* definition files and *one* query file to form the queries that are actually calculated in the pipeline.

There is no need to include a declaration of the namespaces "`ru:`", "`tb:`" and "`functx:`"—these namespaces are inserted at the top of the combined query, should they not have been defined by the user.[2]

[2] The "`ru:`" namespace is used for CorpusStudio-internal function, the "`tb:`" namespace is intended to be used for the user's own functions, and the "`functx:`" namespace is there to allow one-to-one implementation of any function available on http://www.xqueryfunctions.com/



Name: Coref_xq-def
Goal: Functions using coreferencing
Date created: 2012-06-27T10:11:41+02:00
Date last edited: 2014-12-22T12:50:00.0885614+01:00

Explanation:

```
Index:
 tb:Coref tb:CorefLink
 tb:CorefDist

History:
17/jun/2012 ERK Created
```

Text of the definition: (click for full view)

```
(: ---------------------------------------------------------------
   Name : tb:Coref
   Goal : Return those elements where the Coreference is of type [strType]
   History:
   18-10-2010 ERK Created for the NewInfo-Xquery_V1 project
   --------------------------------------------------------------- :)
declare function tb:Coref($this as node()*, $strType as xs:string?) as node()*
{
  for $cns in $this
    let $strRefType := ru:feature($cns, 'RefType')
  where ru:matches($strRefType, $strType)
  return $cns
};

(: ---------------------------------------------------------------
   Name : tb:CorefLink
   Goal : Return those elements where the Coreference is of type [strType]
          and where the IP distance fulfills the requirements
   History:
   18-10-2010 ERK Created for the NewInfo-Xquery_V1 project
   --------------------------------------------------------------- :)
declare function tb:CorefLink($this as node()*, $strType as xs:string?,
                              $intDist as xs:integer) as node()*
{
  for $cns in $this
    let $strRefType := ru:feature($cns, 'RefType')
    let $intIpDist  := ru:feature($cns, 'IPdist') cast as xs:integer
  where ru:matches($strRefType, $strType) and
        ( ($intDist=0) or ($intIpDist > $intDist))
  return $cns
```

Figure 7 The definition editor

The definition editor contains a list of 'definition' files on the left side. These are the definition files that are included in the currently selected CRP. The contents of a definition file becomes available for editing when it is selected in the list.

There are a number of actions that can be performed on or with definition files. Each of these is treated in the following sub sections.

6.3.1 Editing a definition file

A definition file consists of the following components, each of which can be edited:

1) `Name` - The name of the definition 'file'.
   This may not contain spaces and it must be unique within this CRP.
2) `Goal` - Main purpose of this definition file.
3) `Creation` - The time and date when the definition file was created
4) `Changed` - The time and date when the changes were saved last
5) `Comments` - Details of the ‘definition’ in prose
6) `Text` - The Xquery code of the definition

As soon as one of the components in the list is edited, a red Save_ button appears. Changes will only be saved to the CRP file after pressing this button. The user may know that changes have been saved by the fact that the “save” button disappears and by the fact that the “Date last edited” on the main project’s page is adapted.[3]

Should more room for editing the *text* of the definition be needed, then the line above the editor that says “(click for full view)” may be pressed. The Xquery code then takes up the whole “General” area. Return to ‘normal’ view is possible by clicking “(click to return to normal view)”.

[3] Future releases will also change the “Date last edited” on the “Definitions” page.

6.3.2 *Creating a new definition file*

Select **New** >> **New definition file** to create a new definition file from scratch. The user is requested to provide basic information for the new definition file:

1)	`Name`	- The name of the definition 'file'. This may not contain spaces and it must be unique within this CRP.
2)	`Goal`	- Main purpose of this definition file.
3)	`Explanation`	- Details of the 'definition' in prose

Press "Create" to effectuate the creation of a new definition file or "Cancel" to return to the Definition page without creating a new file.

The observant user may have noticed the option **New** >> **Definition wizard**. This option is meant to become available in future releases of the web application.

6.3.3 *Importing a definition file*

Select **New** >> **Upload** to import a definition file from a local computer into the project. This option allows for a number of exchange possibilities:

- **Windows**. Import definition files that have been created using the Windows version of CorpusStudio.
- **Own**. Import one's own definition files that have previously been exported to one's local computer.
- **Exchange**. Import definition files that have been passed on to the user by a colleague.

6.3.4 *Removing a definition file*

Select **Current** >> **Remove** to delete a definition file from the project. The file is completely deleted and this operation can not be undone in any way.

Note: **no confirmation** is asked for. Files are deleted immediately.

6.3.5 *Exporting a definition file*

Definition files may be exported (that is, downloaded) to one's own local computer. Select **Current** >> **Export** to start the export process.

The definition file will be prepared by the web application and put in a place from where it can be downloaded. The space above **New** will receive a clickable link from where the user can download the created file. The file has an "`.xq`" extension. It is a plain text file (in "utf-8" encoding) that can be opened with a text editor like NotePad. Just clicking the download link opens the definition file in a new tab page of the browser.[4]

Exporting definition files may serve different purposes:

- **Exchange**. Export a definition file to share it with a colleague.
- **Collection**. Export a definition file to the user's own (local) computer in order to have a selection of definition files that can be used in other projects.
- **Windows**. Export a definition file to use it in the Windows version of CorpusStudio.

[4] This may depend on the browser being used and its settings.

6.3.6 *Coding a definition file*

The actual coding of definition files and queries is beyond the CorpusStudio manual, but there are a few matters worth mentioning here.

The definition files are meant to contain only the following two elements:

1) **Declarations** - Declarations of variables available to other definitions and queries
2) **Functions** - Definitions of functions in the tb: or functx: namespace

As for **declarations**, valid declarations of variables look like this:

```
declare variable $_finiteIP as xs:string := "IP-MAT*|IP-SUB*";
```

A variable declaration consists of the following components:

- **Preamble**. A declaration starts with the fixed words '`declare variable`'.
- **Name**. The name of the global variable. This name must start with a `$` sign.
- **Type**. The type of the variable needs to be specified, and must be preceded by the keyword "as". Possible types include the following:

`xs:string`	One or more words combined into a string
`xs:integer`	Boolean (either true or false)
`xs:integer`	Whole numbers—either negative or positive (e.g. -20, 4, 1000)
`xs:decimal`	Like integer
`xs:double`	A number containing a floating point (e.g. 3.14)
`element()`	A node
`node()`	A node
`item()`	A node or atomic value

- **Assignment**. The assignment operator `:=` can be paraphrased as 'becomes'
- **Value**. The value assigned to the variable. This must be between common quotation marks. <u>Note</u>: avoid copying definitions from Microsoft Word to a definition file, since this introduces 'curly' quotation marks, and such marks are *not recognized* by the Xquery processor.
- **Semicolon**. Finish the variable definition with a semicolon. Otherwise an error will be reported.

As for **user-defined functions**, here is an example:

```
(: --------------------------------------------------------------
   Name : tb:IsStarred
   Goal : Check whether the given <eTree> node contains an <eLeaf>
             child of the type [Star]
   History:
   13-09-2010 ERK    Created
   -------------------------------------------------------------- :)
declare function tb:IsStarred($this as node()?) as xs:boolean
{
  (: Just check ... :)
  let $ok := $this[child::eLeaf/@Type = 'Star']
  (: Return true if the collection is not empty :)
  return
    if (empty($ok))
      then false()
      else true()
} ;
```

Key elements of a user-defined Xquery function are the following:

- **Preamble**. A declaration starts with the fixed words '`declare function`'.

- **Name**. The name of the user function. This name must start with a `tb:` namespace prefix, and it may not contain spaces.
- **Arguments**. One or more arguments can be specified between brackets of the function definition. An argument consists of: (a) variable name, (b) type definition. The name of the argument here *does not have to be equal* to the name of the function's argument where it is called from. It is, in fact, quite common practice to use a different name here in order to avoid confusion. Note: the type definition may be followed by one of the symbols '`?`', '`*`', '`+`', which indicate respectively: optional argument, argument may appear zero or more times, argument may appear one or more times.
- **Type**. The type of the value returned. The example returns a boolean value.
- **Definition**. The body of the definition is to be surrounded by curly brackets {…}. The contents of a user function may be:
    - A simple statement, such as "`empty($this/child::eTree)`"
      Note: **no** return keyword can be used if the function consists of one statement.
    - A 'let-return' sequence, such as in the example function
    - A 'for-let-where-return' sequence
- **Semicolon**. The definition of the user function must be concluded by a semicolon.

There are a number of useful web sites that discuss writing Xquery user functions:

- W3c. The well-known "w3c" school gives some basic facts on Xquery user functions.
- Stylus. Dr. Michael Kay, one of the key programmers of the Saxon Xquery interface used in CorpusStudio, provides a tutorial on Xquery user functions.

6.4 The "Projects/Queries" page

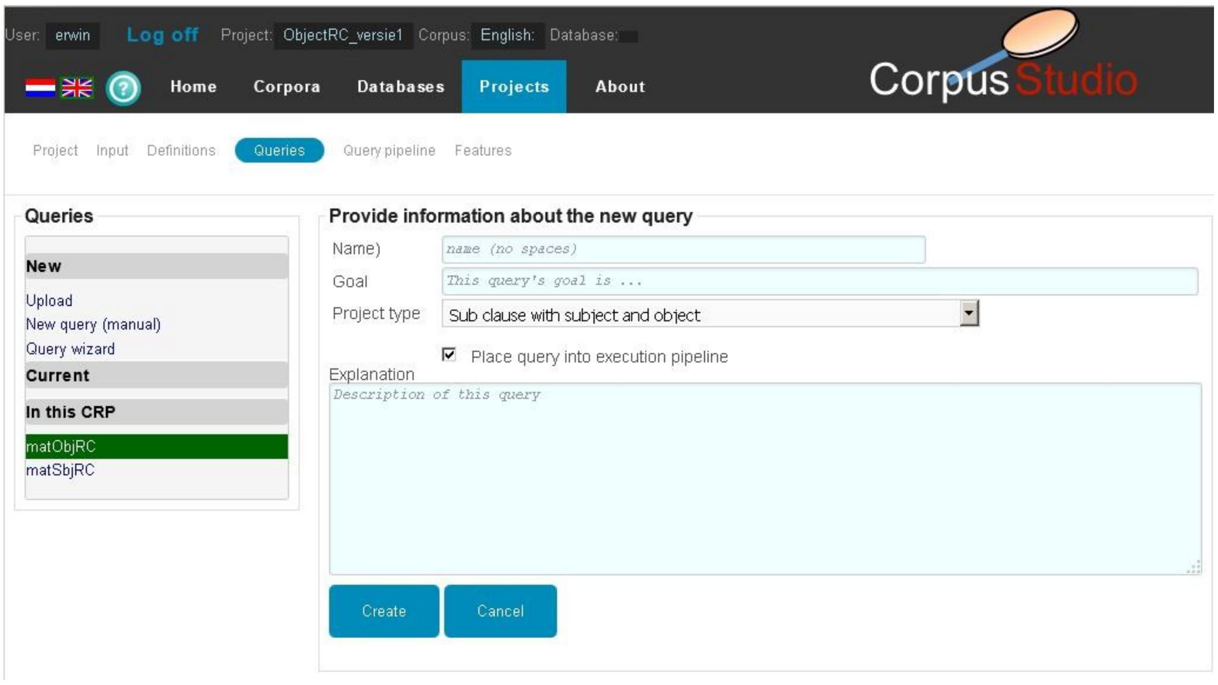
The "Queries" page facilitates creating, importing, exporting and editing of search specifications coded in the Xquery language. The options available for each query are similar to the options available for definition files:

- **New** >> **Upload**. Upload a query from the user's local computer to the web application.
- **New** >> **New query (manual)**. Create a new query manually (discussed below).
- **New** >> **Query wizard**. This is for future releases.
- **Current** >> **Download**. Create a copy of the query as an '`.xq`' file and present an URL to download it from the web application to the local computer.
- **Current** >> **Remove**. Delete the currently selected query from the project. Note: deletion takes effect immediately and knows no 'undo'.

This section on the Queries focuses on the two main issues where queries differ from definitions: (a) Query creation and (b) the code for queries meant to run in CorpusStudio. The remaining functionality is similar to that offered for Definitions, so the user is referred to that section for details.

6.4.1 *Query creation*

Queries can be created using the **New**/**NewQuery**(manual). This brings up the form shown in Figure 8.


User: erwin Log off Project: ObjectRC_versie1 Corpus: English: Database:

Provide information about the new query

Name)	name (no spaces)
Goal	This query's goal is ...
Project type	Sub clause with subject and object
☑	Place query into execution pipeline
Explanation	Description of this query

Figure 8 The NewQuery form

The options available for a new query differ slightly from those that are required for a new definition. They are:

1)	`Name`	- The name of the query. This may not contain spaces and it must be unique within this CRP.
2)	`Goal`	- Main purpose of this query (succinct but meaningful).
3)	`Project type`	- The basic kind of query.
4)	`Pipeline`	- Put this query in the execution pipeline or not (yet)?
5)	`Explanation`	- Details of the query in prose. Important for later reference.

Although the New Query form is not a wizard, it does allow the user to specify a basic '**project type**' for the query to be formed. The number and kind of basic queries may vary from release to release of CorpusStudio. Choosing a basic type here will result in the automatic creation of a query body that is 'tuned' to the environment: it takes into account what kind of text files are being processed (e.g. Psdx, FoLiA), and it takes into account the kind of part-of-speech tags and constituent labels used in the corpus that has been selected in 'Input'.

Important: first specify the "Input" and only then create a new query.

If the CRP's input is changed *after* a new query has been created, then the query may no longer be valid. A change in input corpus may mean that different part-of-speech tags are being used, and a change in input text type (e.g. from FoLiA to Psdx or the reverse) definitely means that the *xml* tags to be looked for are different.

The **pipeline** checkbox is by default 'checked', so that any manually created new query is automatically put in the query execution pipeline.

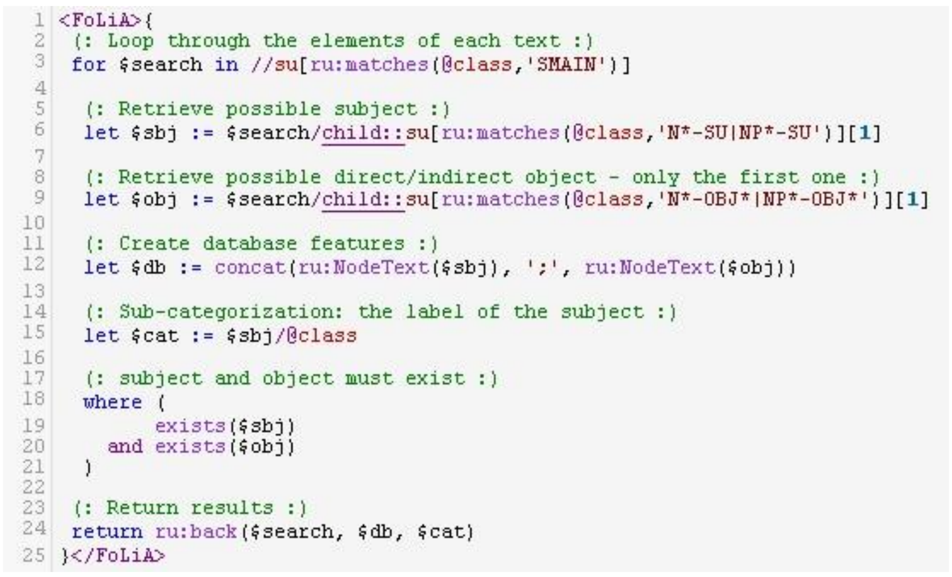
Important: only queries that are part of the query 'pipeline' will actually be executed!!

6.4.2 Coding queries for CorpusStudio

The elements of a query are the following:

- **Opening**. The query must start with an *xml* opening tag followed by an opening curly bracket. The actual opening tag that is being used does not really matter because of the way CorpusStudio processes the queries.[5] It is common practice to use the tag name `<FoLiA>` where FoLiA files are processed and to use `<TEI>` for Psdx files.
- **Body**. The body of the query normally consists of a for-let-where-return sequence, where the 'return' part should call the CorpusStudio function `ru:back()`.[6]
- **Closure**. The query must be closed by a closing curly bracket and an *xml* end-tag that matches the opening tag. The difference between an opening and a closing tag is the presence of a forward slash '/' in the latter.

The form and layout of the Xquery code that is used for queries can best be explained by looking at an example:



```
 1 <FoLiA>{
 2  (: Loop through the elements of each text :)
 3  for $search in //su[ru:matches(@class,'SMAIN')]
 4
 5   (: Retrieve possible subject :)
 6   let $sbj := $search/child::su[ru:matches(@class,'N*-SU|NP*-SU')][1]
 7
 8   (: Retrieve possible direct/indirect object - only the first one :)
 9   let $obj := $search/child::su[ru:matches(@class,'N*-OBJ*|NP*-OBJ*')][1]
10
11   (: Create database features :)
12   let $db := concat(ru:NodeText($sbj), ';', ru:NodeText($obj))
13
14   (: Sub-categorization: the label of the subject :)
15   let $cat := $sbj/@class
16
17   (: subject and object must exist :)
18   where (
19          exists($sbj)
20      and exists($obj)
21   )
22
23  (: Return results :)
24  return ru:back($search, $db, $cat)
25 }</FoLiA>
```

Figure 9 Example query

The lines 1 and 25 provide the obligatory opening and closure of the query. Line 3 starts the for-block and specifies that the query considers `<su>` nodes (the 'su' nodes in FoLiA are the 'syntactic units'; they are equivalent to the 'eTree' nodes, embedded trees, in Psdx). Only those nodes that have a `@class` attribute 'SMAIN' (main clauses in the Sonar corpus) are regarded further. The for specifier in line 3 says that it will loop through the whole text and that every time the conditions are met, the variable `$search` will contain the 'SMAIN' node.

Lines 6, 9, 12 and 15 create local variables `$sbj`, `$obj`, `$db` and `$cat` and assign values to them. The assignments in lines 6 and 9 build on the `$search` variable. The `$obj` in line 9, for instance, is defined as being the first `<su>` node that is a child from `$search`, whose `@class`

[5] CorpusStudio does not use the *xml* code that is produced by the Xquery code. It makes use of the arguments passed to the '`ru:back()`' function instead. This means that CorpusStudio is not a general-purpose Xquery processing platform; it is fine-tuned and optimalized towards using Xquery's search and variable-creation mechanisms for the benefit of linguistic research.

[6] The 'where' part can be skipped, provided the 'for' part restricts the input in the way required by the user. What is usually done, however, is dividing the restriction between a more general 'for' part and a 'where' part that fine-tunes the restrictions. This gives code that is more readable and that lasts longer.

attribute matches either `N*-OBJ*` or `NP*-OBJ*`.[7] The built-in function `ru:matches()` is explained in the appendix, in section 8.3.8.

Note that the restriction blocks `[]` may be put one-after-the-other, and that they are considered left-to-right. If one were to reverse the order of the blocks, resulting in `$search/child::su[1][ru:matches(@class, 'N*-OBJ*|NP*-OBJ*')]`, then the meaning would change. It would be: take the first `<su>` node that is a child from `$search`, and check if this node has a `@class` attribute matches either `N*-OBJ*` or `NP*-OBJ*`. If there is a match, then the node is assigned to $obj; if there is no match, then `$obj` will be empty.[8]

Line 12 makes use of a standard Xquery function `concat()`. There exists a range of standard functions the user can use in coding. These functions are listed and explained at w3c. This line also uses the function `ru:NodeText` that can be used to get a plain text representation of all the words that are part of a constituent. What line 12 does is create a string that consists of the text of the subject and the text of the object, separated by a semicolon. This kind of format is suitable to pass on as second argument in the `ru:back()` function, and then build a result feature database from it.

Line 15 assigns the `@class` attribute of the subject variable `$sbj` to a variable `$cat`. This variable is passed on as the third argument of `ru:back()`, where it serves as a '**sub-categorization**' label. Result numbers will be given not only for the grand total of hits, but they will be sub-divided over those that have the same value for `$cat`. This feature is illustrated in the numerical results that appear when the project is run with the "WR-P-P-K_reports" section of Sonar500 as input (for result-displaying see section 6.8):

QC	Result Label	Category	Number
2	matSbjObj	(all together)	5676
2	matSbjObj	N-SU	328
2	matSbjObj	NP-SU	5348

The first line with numerical results gives the number (5676) of hits for all categories together, while the second and third line divide this over the sub categorization tags that have been used: the POS-tag of the subject. The texts have the `-SU` label attached either to single nouns or to whole NPs. The reason this query doesn't find pronominal subjects is the fact that the selection in lines 6 and 19 does not allow for them. Changing the second argument of `ru:matches()` in line 6 to "`*-SU`" leads to a larger number of sub categories:

QC	Result Label	Category	Number
2	matSbjObj	(all together)	10282
2	matSbjObj	ADJ-SU	65
2	matSbjObj	BW-SU	14
2	matSbjObj	CONJ-SU	459
2	matSbjObj	CP-SU	63
2	matSbjObj	LID-SU	19
2	matSbjObj	MWU-SU	342
2	matSbjObj	N-SU	346
2	matSbjObj	NP-SU	5771
2	matSbjObj	OTI-SU	7
2	matSbjObj	SPEC-SU	1094
2	matSbjObj	TI-SU	6
2	matSbjObj	TW-SU	7
2	matSbjObj	VG-SU	23
2	matSbjObj	VNW-SU	2018
2	matSbjObj	WHREL-SU	12
2	matSbjObj	WHSUB-SU	9
2	matSbjObj	WW-SU	27

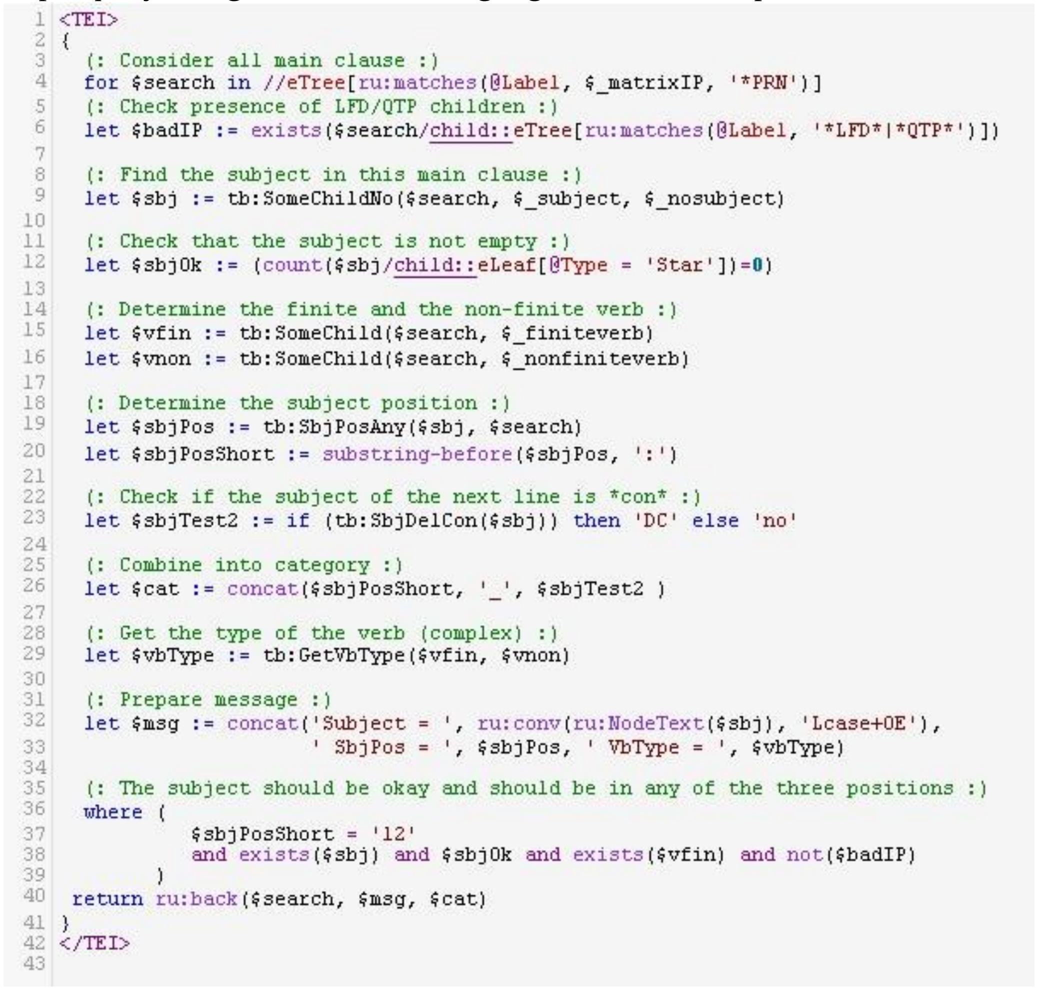
[7] The observant reader may notice that the first definition implies the second one, so that `N*-OBJ*` would be sufficient.

[8] Checking for empty nodes is possible through the functions `empty($obj)` or `not(exists($obj))`.

The where condition in lines 18-21 says that the nodes found for the subject and object *must* exist. If they do, the function continues; if they don't the function is not executed any further, and processing continues with the next `<su>` node that has a 'SMAIN' `@class` attribute.

The for-block ends with a '`return`' statement, and this statement is obligatory within the context of CorpusStudio. That is to say, any results in terms of counts, result database features and sub categorization, will only become available if this function `ru:back()` is called in a return statement with the correct arguments.

The example query in Figure 10 serves to highlight a few more aspects.



```
 1 <TEI>
 2 {
 3   (: Consider all main clause :)
 4   for $search in //eTree[ru:matches(@Label, $_matrixIP, '*PRN')]
 5   (: Check presence of LFD/QTP children :)
 6   let $badIP := exists($search/child::eTree[ru:matches(@Label, '*LFD*|*QTP*')])
 7
 8   (: Find the subject in this main clause :)
 9   let $sbj := tb:SomeChildNo($search, $_subject, $_nosubject)
10
11   (: Check that the subject is not empty :)
12   let $sbjOk := (count($sbj/child::eLeaf[@Type = 'Star'])=0)
13
14   (: Determine the finite and the non-finite verb :)
15   let $vfin := tb:SomeChild($search, $_finiteverb)
16   let $vnon := tb:SomeChild($search, $_nonfiniteverb)
17
18   (: Determine the subject position :)
19   let $sbjPos := tb:SbjPosAny($sbj, $search)
20   let $sbjPosShort := substring-before($sbjPos, ':')
21
22   (: Check if the subject of the next line is *con* :)
23   let $sbjTest2 := if (tb:SbjDelCon($sbj)) then 'DC' else 'no'
24
25   (: Combine into category :)
26   let $cat := concat($sbjPosShort, '_', $sbjTest2 )
27
28   (: Get the type of the verb (complex) :)
29   let $vbType := tb:GetVbType($vfin, $vnon)
30
31   (: Prepare message :)
32   let $msg := concat('Subject = ', ru:conv(ru:NodeText($sbj), 'Lcase+OE'),
33                      ' SbjPos = ', $sbjPos, ' VbType = ', $vbType)
34
35   (: The subject should be okay and should be in any of the three positions :)
36   where (
37              $sbjPosShort = '12'
38              and exists($sbj) and $sbjOk and exists($vfin) and not($badIP)
39         )
40  return ru:back($search, $msg, $cat)
41 }
42 </TEI>
43
```

Figure 10 Query that exemplifies calling user-functions

This second example query is meant to search through Psdx texts that are part of the historical English corpora, such as the YCOE (Taylor et al. 2003). Hence its reference to *xml* tags `<eTree>` (syntactic constituents), `<eLeaf>` (end-nodes) and its reference to the attributes `@Label` (pos-tag) and `@Type` (one of four possible end-node types).

The query introduces a few new facets that deserve mentioning:

1) **Xquery functions** - Line 12 makes use of the Xquery funcion `count()`, which counts the number of nodes it receives as argument, and line 20 makes use of one of Xquery string functions, '`substring-before()`'.
2) **CorpusStudio functions** - Line32 uses one more built-in function: `ru:conv()`. This converts a string using a particular format.

3) **User functions** - Lines 9, 15-16, 19, 23 and 29 call functions that have been defined in definition files by the user. These functions can be recognized by their '`tb:`' namespace prefixes.

The order of execution of a query written in Xquery is data-dependent. The processor first looks at the return statement, and sees which elements are needed for that: `$search`, `$msg` and `$cat`. It then regards each of these elements, finding out how their values can be determined and so forth. Note that as a consequence of this method, line 6 is only actually executed if lines 9, 12 and 15 provide positive results: see line 38 for this.

Note that line 32 now prepares a string variable called `$msg`, but this variable only serves as a message that accompanies each of the results found. More about when and how the second variable of `ru:back()` serves as a feature database specifier in section 6.6.

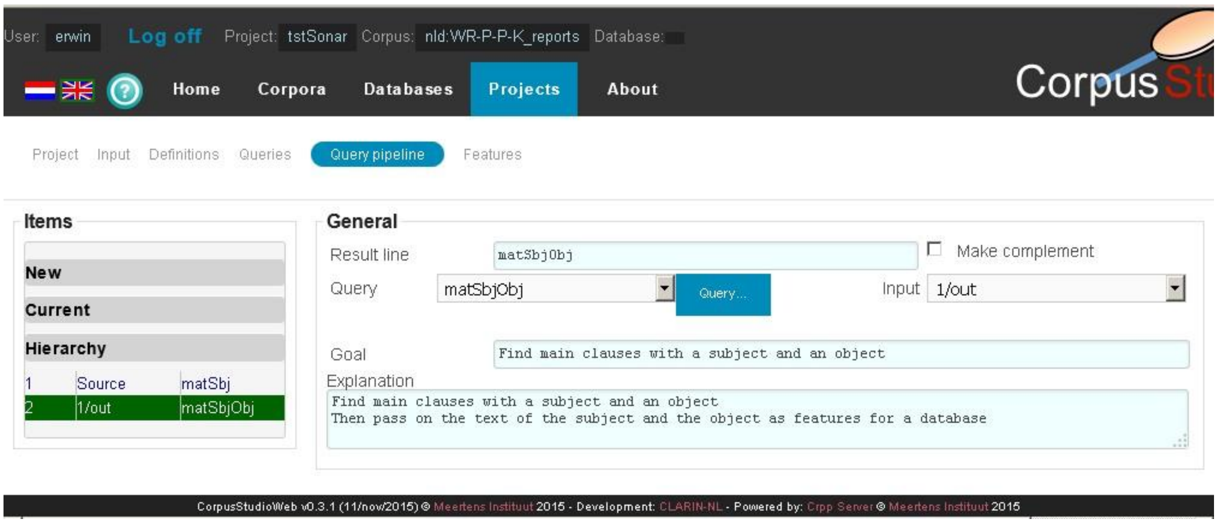
6.5 The "Projects/Pipeline" page

The queries that are part of the CRP all appear in the list in the Query page. These queries can *potentially* be executed in the context of this project. Whether and how they are, in fact, executed depends on where and how they appear in the 'Pipeline' page of the CRP.

The options available for each element of the query execution pipeline are:

- **New** >> **New Line**. Add a new query line to the pipeline.
- **Current** >> **Remove**. Delete the currently selected line from the pipeline.

Note that it is *not possible* to remove a line from the query execution pipeline that serves as input for another line.



1	Source	matSbj
2	1/out	matSbjObj

Result line: matSbjObj
Query: matSbjObj
Input: 1/out
Goal: Find main clauses with a subject and an object
Explanation:
Find main clauses with a subject and an object
Then pass on the text of the subject and the object as features for a database

Figure 11 The pipeline page (constructor editor)

The example of the pipeline shown in Figure 11 contains two lines. The first line has 'Source' as input, as can be learned from the 'Hierarchy' section in the Items selector on the left hand side. This means that the line with query 'matSbj' takes *all* the sentences in the text corpus as its input.

The second line, however, only takes the sentences that are passed through by the first line as input. Only those sentences that have a main clause with a subject, according to the query 'matSbj', serve as input to the second query called 'matSbjObj'. This is what is meant by the "Input" specification '1/out'.

This kind of input restriction might lead to a small, data-dependent, reduction in execution time. Its purpose may be found more on the conceptual level—a researcher may want to explicitly fine-tune a number of queries on top of one another.

The constructor editor (or 'pipeline page') allows for a number of adjustments:

1) `Result line` - The label that is used to identify the results for this line. This label needs to be unique within the constructor editor (the pipeline page). When **New** >> **NewLine** is used, the name of the query is offered as result line label. This is usually okay, unless a query serves is called upon in more than one line.
2) `Make complement` - Check this box if the 'complement' of the current line should serve as input to another line. This complement contains all *sentences* (not just constituents!!) that do not pass the query.
3) `Query` - Select one of the queries as specified in the Query editor.
4) `Input` - Choose the part of the data that should serve as input to this query line: *all* data ('Source'), the output of a line (n/out) or the complement of a preceding line (n/cmp).
5) `Goal` - The goal of this query line. The goal of the query is copied here initially, but this can be adjusted.
6) `Explanation` - Details on this line. The text is initially copied from the query.

The button "Query" within the 'General' area allows immediate jumping to the selected query for quick inspection.

Creating a query on the Query page through New/NewQuery normaly places this query in the execution pipeline. The user should nevertheless verify the Pipeline page to make sure that the 'Input' specification of each line is set correctly.

6.6 The "Projects/Features" page

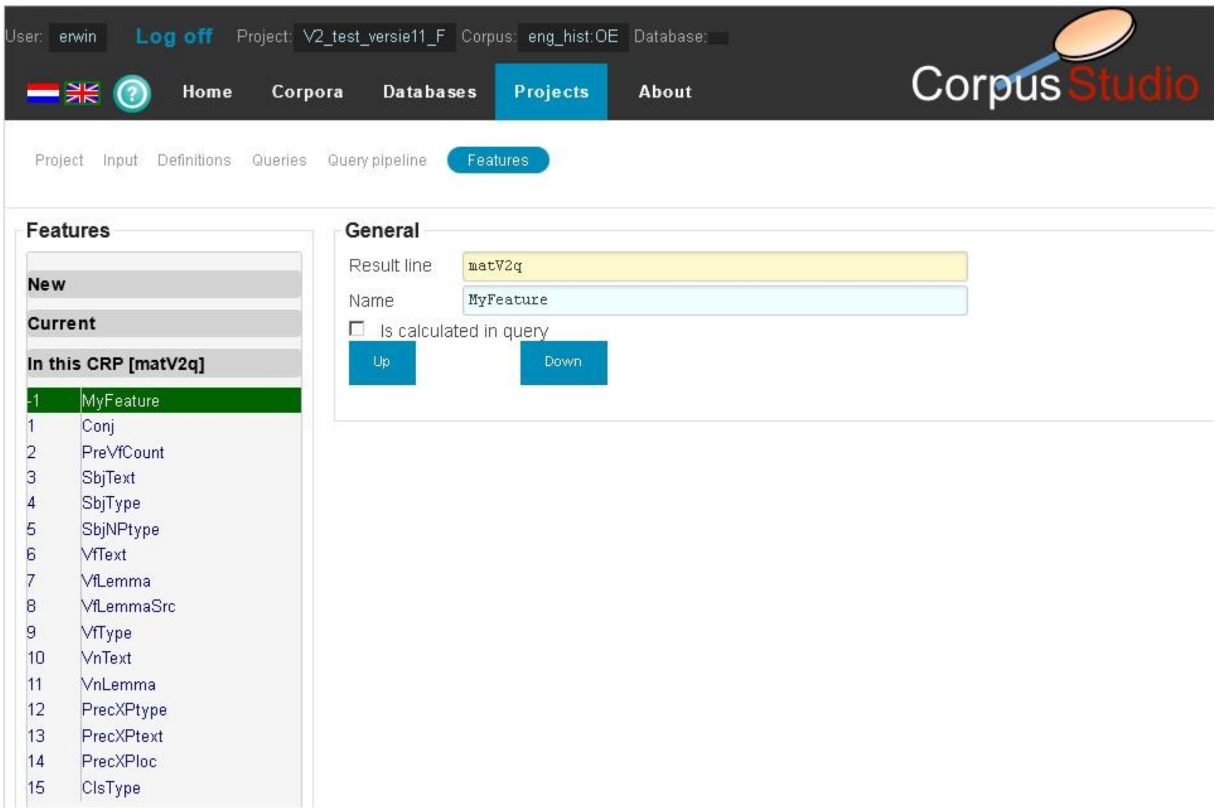
One of the beauties of CorpusStudio is that every query can be made to produce a 'Results database'. Such a database contains user-specified fields per 'hit'.

Telling CorpusStudio to make such a result database involves two steps:

1) Provide a **semi-colon separated list** of values as second argument to `ru:back()`
2) Specify the **names of all features** in the Projects/Features page

CorpusStudio is blissfully unaware of what the second argument of `ru:back()` is going to be used for, basically assuming that it is just a comment that needs to accompany the resulting hits that the user will take a look at. It is only when a list of feature names is provided in the Projects/Features page for a particular query execution line that the web application will attempt to produce a Result database.

The 'Features' page, then, is dependant upon the element in the query execution pipeline that is currently selected. Once a line in the Query Pipeline has been selected, one can create or edit the feature name specifications for it in the Feature editor.



Features

New

Current

In this CRP [matV2q]

-1	MyFeature
1	Conj
2	PreVfCount
3	SbjText
4	SbjType
5	SbjNPtype
6	VfText
7	VfLemma
8	VfLemmaSrc
9	VfType
10	VnText
11	VnLemma
12	PrecXPtype
13	PrecXPtext
14	PrecXPloc
15	ClsType

General

Result line: matV2q

Name: MyFeature

☐ Is calculated in query

Figure 12 Feature editor

Features can be divided into two groups: (1) those that are calculated within a query, and (2) those that are *not* calculated within a feature.

The first group, the ‘pre-calculated’ features, must be part of the ‘semi-colon-separated-string’ that is passed as second argument to `ru:back()`. Their position within this string must match exactly with the *number* they have in the Feature list. All features in the second group receive an ordering number -1 to set them apart from the features in the first group.

Each feature has the following characteristics:

1) `Result line` - This is the label of the Pipeline’s query line to which the features here belong. This label has a yellow background, which means that it cannot be changed. The idea is that one first selects a result line in the Pipeline, and then continues to edit the features for that line in the Feature editor page.
2) `Name` - The name of the feature. This name can be chosen completely independent of the names used in the Xquery code that calculated the feature values.
3) `Is calculated in query` - Check this box to indicate that the feature is being calculated within the query.
4) `Feature number` - The feature number is assigned automatically. Features that are not calculated within a query receive the "-1", while the other features get consecutive numbers. The order of the features (and their numbering) can be changed by selecting a feature and using the buttons ‘Up’ and ‘Down’ to change their position in the list.

6.7 Query execution

Execution of the queries that have been placed in the Pipeline (the constructor editor) means that the web application takes each text from the corpus that has been specified in the Input page, and line-by-line executes the combination of the Xquery definitions and each query on them.

There are two buttons that can be used to execute the queries. These buttons appear only when the 'Project' page, the main page for CRP editing, is selected. The button '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.

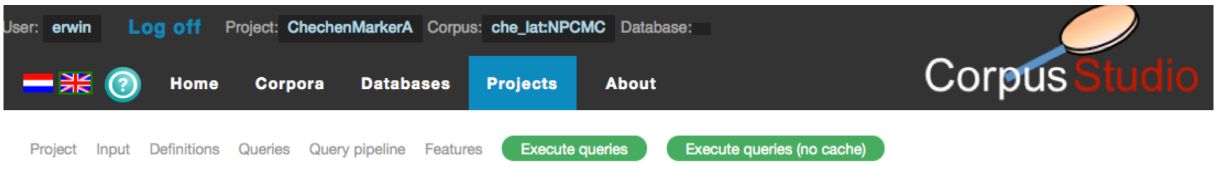


Figure 13 Two query execution buttons

The button labelled "Excute queries (no cache)" executes the queries afresh, without making use of any previously stored data. Use this button if there is reason to believe something went wrong on your computer or on the server.

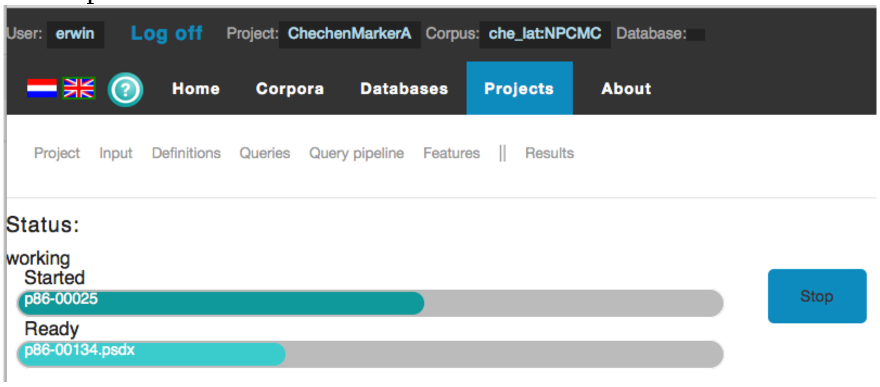
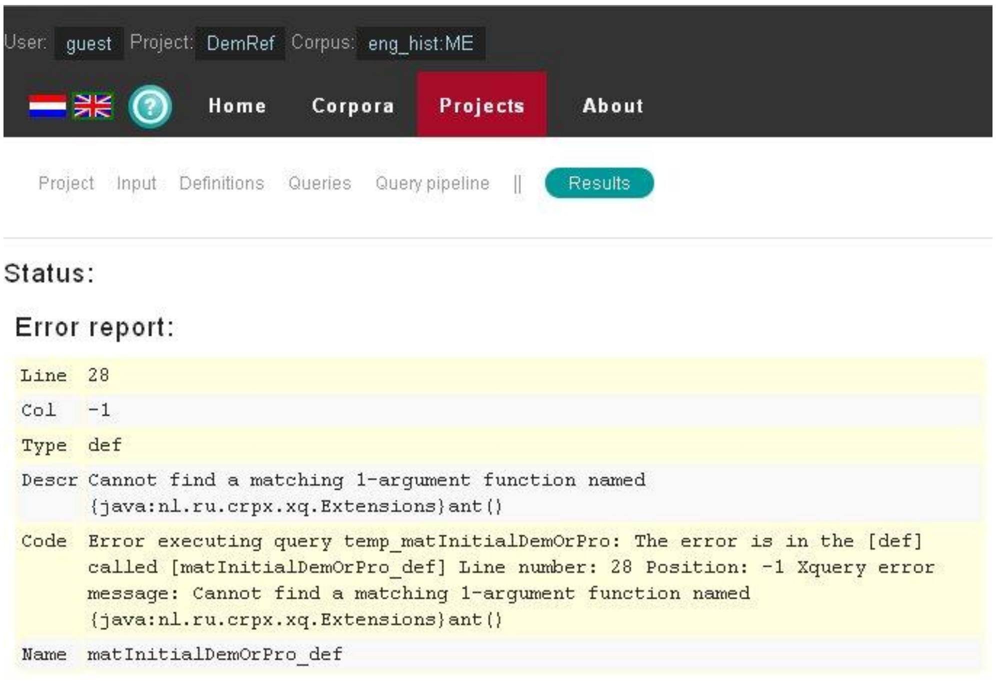


Figure 14 Progress bars while executing a project

The normal reaction to pressing the **Execute** button is to reveal 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. Depending on the project and the number of processes on the server, it may take some time before the progress bars become visible.

The space below 'Status' may give additional information on the status of the project that is being executed.

There are at least two kinds of possible errors in the Xquery code. Syntax errors or making use of internal functions that do not exist will show up before the actual execution of the project takes place. An error message like the one in Figure 15 will appear.



Status:

Error report:

Line	28
Col	-1
Type	def
Descr	Cannot find a matching 1-argument function named {java:nl.ru.crpx.xq.Extensions}ant()
Code	Error executing query temp_matInitialDemOrPro: The error is in the [def] called [matInitialDemOrPro_def] Line number: 28 Position: -1 Xquery error message: Cannot find a matching 1-argument function named {java:nl.ru.crpx.xq.Extensions}ant()
Name	matInitialDemOrPro_def

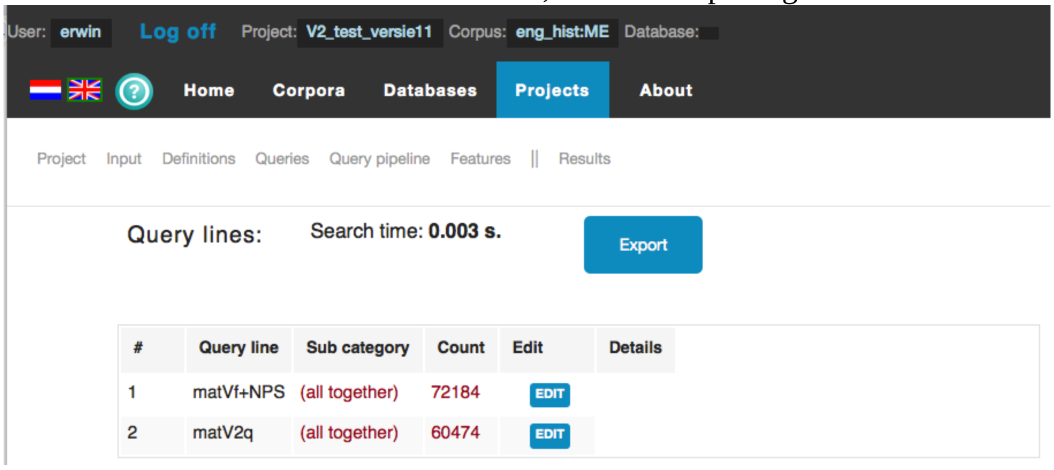
Figure 15 An error in the Xquery code

The error report identifies the name of the query or definition (see “Name”, last row) where the error has occured, 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.

6.8 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 for example Figure 13.



Query lines: Search time: **0.003 s.**

#	Query line	Sub category	Count	Edit	Details
1	matVf+NPS	(all together)	72184	EDIT	
2	matV2q	(all together)	60474	EDIT	

Figure 16 Concise overview of the results

The overview in Figure 13 shows the number of hits for each of the seven query lines, such as ‘60474’ for line #2.

6.8.1 The results overview

The overview table serves as a platform for further processing. The options are:

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:

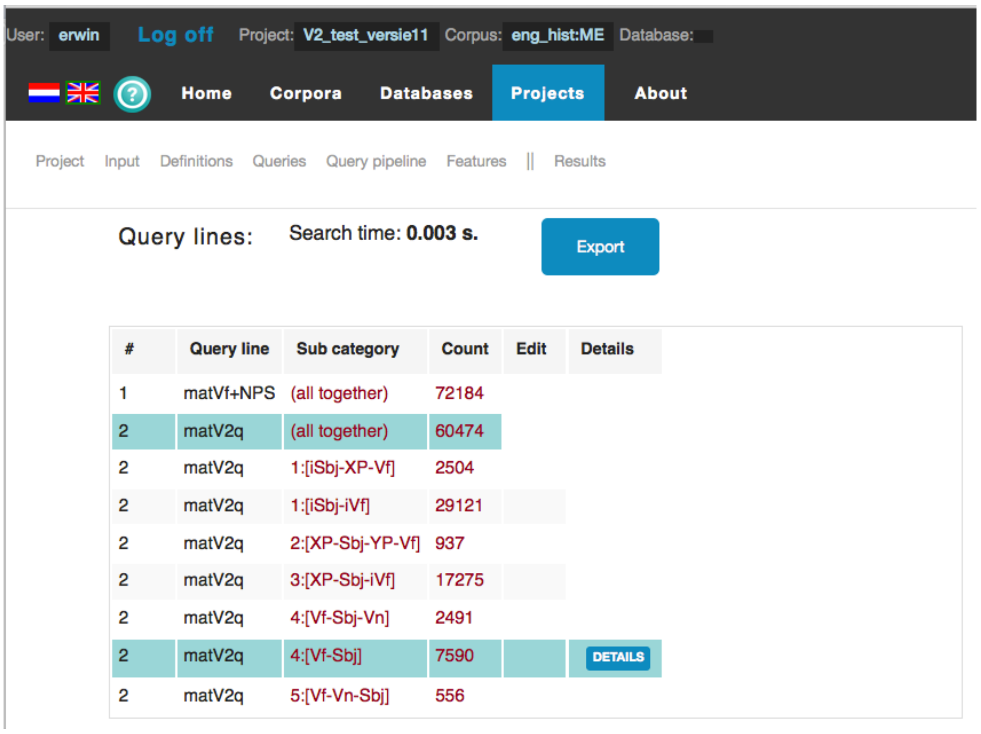
```
date:             2015-07-31 13:45:03

overview table

QC   Result Label Category          Number
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
2    matV2q       5:[Vf-Vn-Sbj]     556
```

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

Pressing the redded 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 section 8.3.1 on this feature of `ru:back()` and section 6.4.2 for an example), 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.



Query lines: Search time: **0.003 s.**

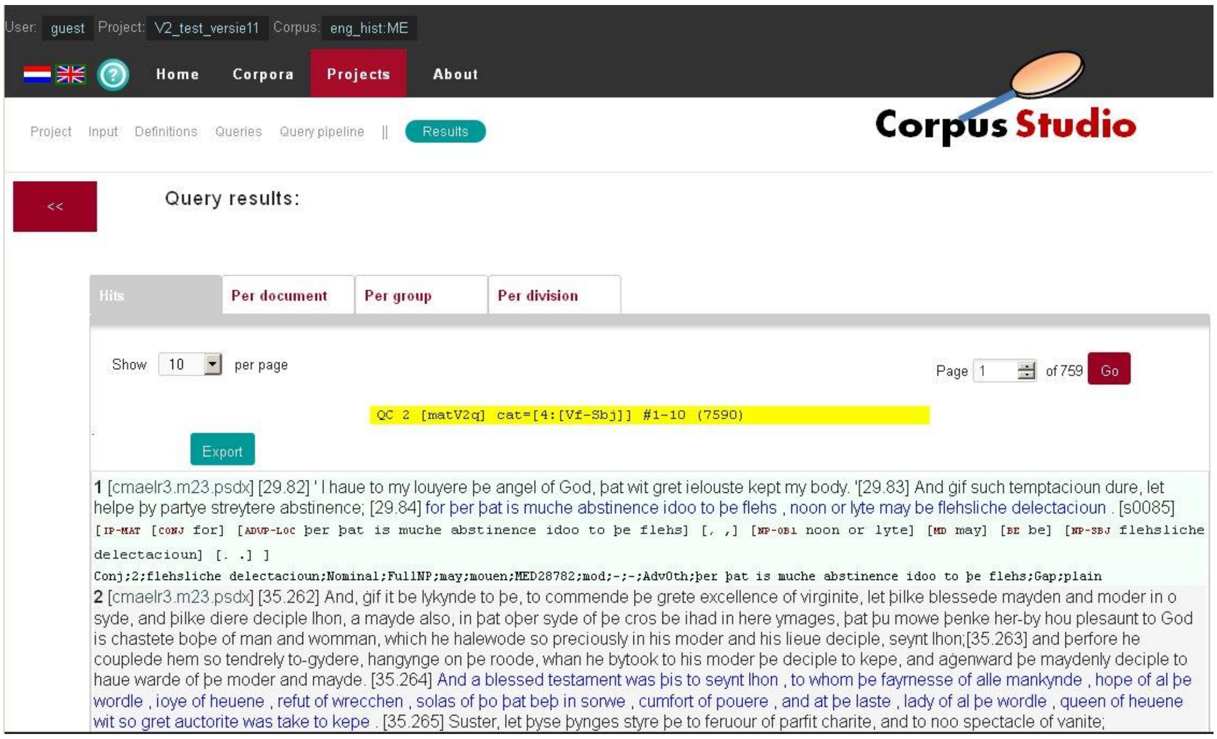
#	Query line	Sub category	Count	Edit	Details
1	matVf+NPS	(all together)	72184		
2	matV2q	(all together)	60474		
2	matV2q	1:[iSbj-XP-Vf]	2504		
2	matV2q	1:[iSbj-iVf]	29121		
2	matV2q	2:[XP-Sbj-YP-Vf]	937		
2	matV2q	3:[XP-Sbj-iVf]	17275		
2	matV2q	4:[Vf-Sbj-Vn]	2491		
2	matV2q	4:[Vf-Sbj]	7590		DETAILS
2	matV2q	5:[Vf-Vn-Sbj]	556		

Figure 17 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.

6.8.2 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.[9] The "hit-by-hit", as illustrated in Figure 18, is the default one.



```
QC 2 [matV2q] cat=[4:[Vf-Sbj]] #1-10 (7590)
```

1 [cmaelr3.m23.psdx] [29.82] ' I haue to my louyere þe angel of God, þat wit gret ielouste kept my body. '[29.83] And ġif such temptacioun dure, let helpe þy partye streytere abstinence; [29.84] for þer þat is muche abstinence idoo to þe flehs , noon or lyte may be flehsliche delectacioun . [s0085]

```
[IP-MAT [CONJ for] [ADVP-LOC þer þat is muche abstinence idoo to þe flehs] [, ,] [NP-OB1 noon or lyte] [MD may] [BE be] [NP-SBJ flehsliche delectacioun] [. .] ]
Conj;2;flehsliche delectacioun;Nominal;FullNP;may;mouen;MED28782;mod;-;-;AdvOth;þer þat is muche abstinence idoo to þe flehs;Gap;plain
```

2 [cmaelr3.m23.psdx] [35.262] And, ġif it be lykynde to þe, to commende þe grete excellence of virginite, let þilke blessede mayden and moder in o syde, and þilke diere deciple Ihon, a mayde also, in þat oþer syde of þe cros be ihad in here ymages, þat þu mowe þenke her-by hou plesaunt to God is chastete boþe of man and womman, which he halewode so preciously in his moder and his lieue deciple, seynt Ihon;[35.263] and þerfore he couplede hem so tendrely to-gydere, hangynge on þe roode, whan he bytook to his moder þe deciple to kepe, and aġenward þe maydenly deciple to haue warde of þe moder and mayde. [35.264] And a blessed testament was þis to seynt Ihon , to whom þe faymesse of alle mankynde , hope of al þe wordle , ioye of heuene , refut of wrecchen , solas of þo þat beþ in sorwe , cumfort of pouere , and at þe laste , lady of al þe wordle , queen of heuene wit so gret auctorite was take to kepe . [35.265] Suster, let þyse þynges styre þe to feruour of parfit charite, and to noo spectacle of vanite;

Figure 18 Result details per hit

The detailed results information offers the following possibilities:

1) **Back**. The << button returns to the general result table overview (see 6.8.1).
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* Go" combination allows one to select the page to be shown.

Note: it is possible to get *all* the hits on one 'page', but this process may take some time, depending on the amount of hits there exist.

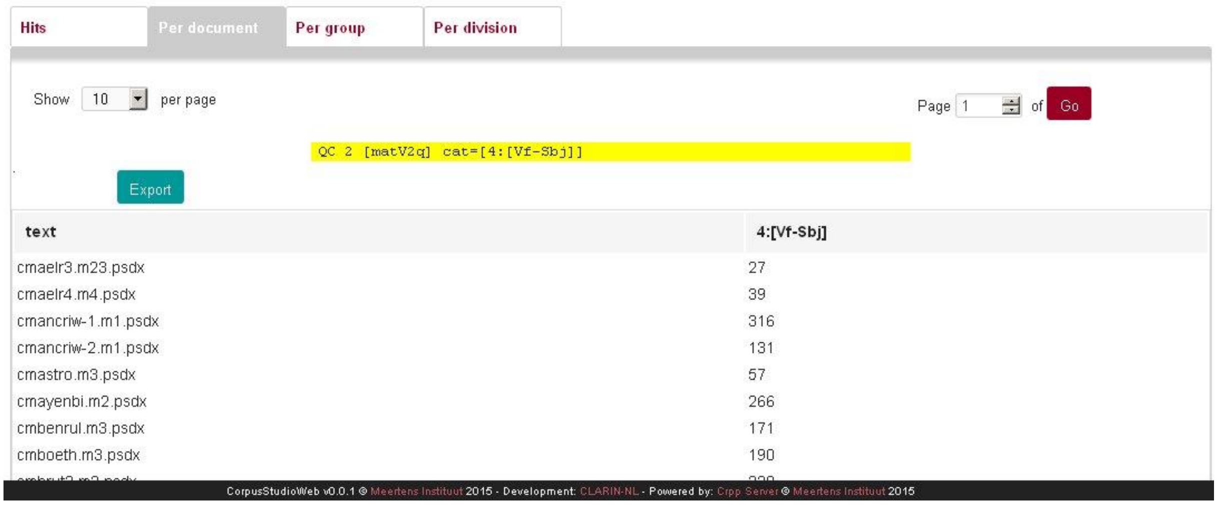
The **Export** button causes a tab-separated text file to be made that contains the results shown in the currently selected page. Since this export file may be useful for further research in other programs, it is important to know the format of the text file. Each line in the tab-separated text file contains the following eight fields:

[9] The 'hits-per-group' and 'hits-per-division' options are planned to be available in future releases. Anything that is found on these tab pages now is experimental and subject to change.

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

6.8.3 *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. Only those documents that have at least one 'hit' in them are shown in this overview. When one category has been selected in the Results overview page, the frequencies are shown per document for that category (Figure 19), but when the query line has been selected as a whole, the frequencies for all the categories are shown in a table (Figure 20).

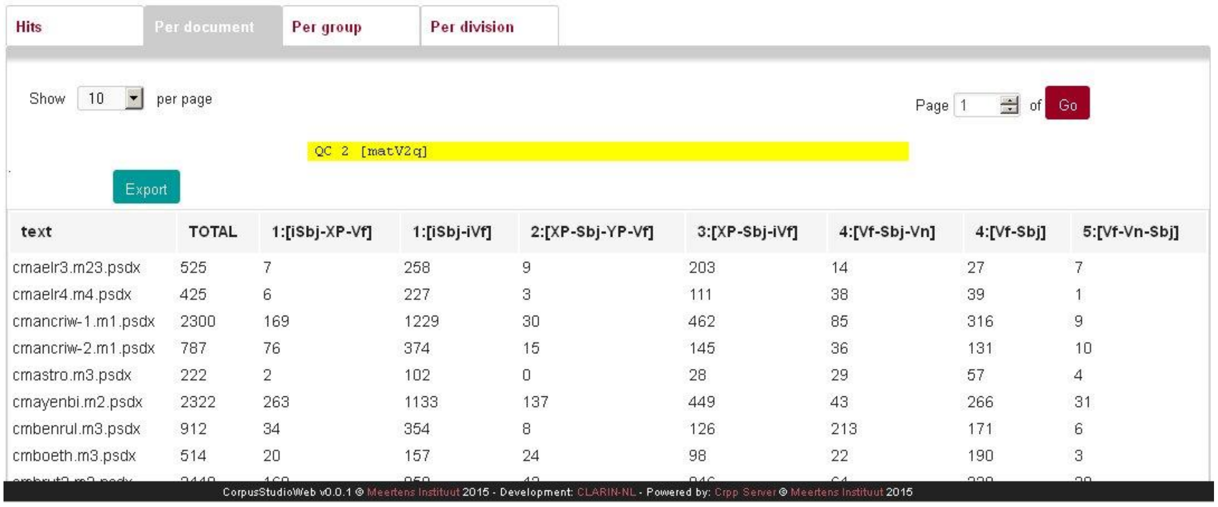

Hits
Per document
Per group
Per division
Show 10 per page
Page 1 of Go
Export
CorpusStudioWeb v0.0.1 © Meertens Instituut 2015 - Development: CLARIN-NL - Powered by: Crpp Server © Meertens Instituut 2015

QC 2 [matV2q] cat=[4:[Vf-Sbj]]

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

Figure 19 Result details per document: one category selected

.


Hits
Per document
Per group
Per division
Show 10 per page
Page 1 of Go
Export
CorpusStudioWeb v0.0.1 © Meertens Instituut 2015 - Development: CLARIN-NL - Powered by: Crpp Server © Meertens Instituut 2015

QC 2 [matV2q]

text	TOTAL	1:[iSbj-XP-Vf]	1:[iSbj-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
cmancriw-1.m1.psdx	2300	169	1229	30	462	85	316	9
cmancriw-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
cmbenrul.m3.psdx	912	34	354	8	126	213	171	6
cmboeth.m3.psdx	514	20	157	24	98	22	190	3

Figure 20 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). These lines have been made 'red' in the example above, but in the text file they do not have a distinguishing color.

Subsequent lines contain the information per document:

1) the total number of hits for the document for this query line, and

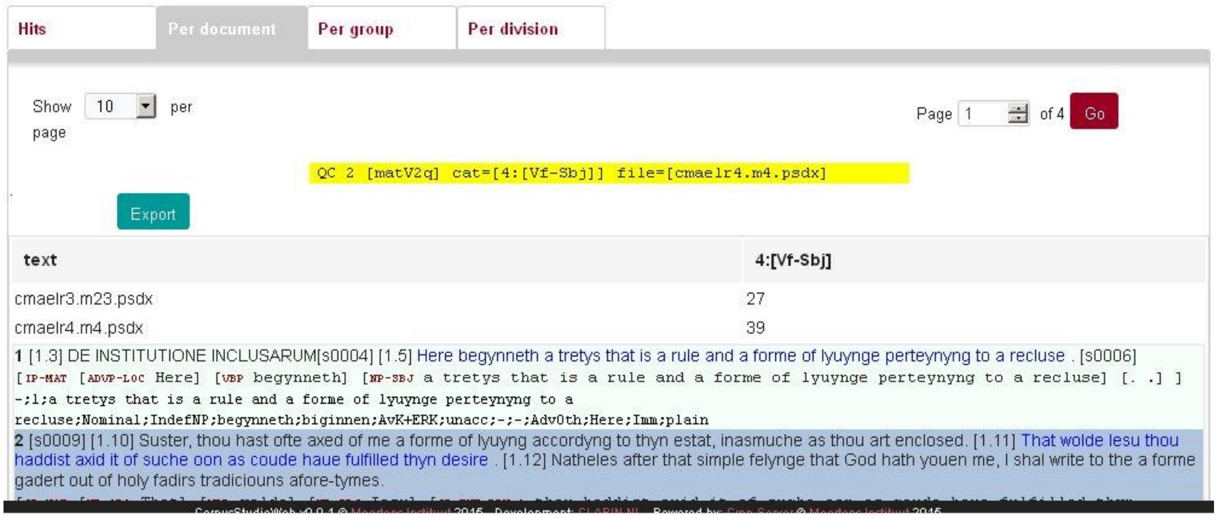
2) the number of hits taken up by the currently selected sub category.

This allows for the calculation of relative frequencies. Future releases will have more quantitative information of the type that is already available in the Windows edition of CorpusStudio:

1) the number of sentences per document (that is: the number of `<forest>` tags in Psdx texts or the number of `<s>` tags in FoLiA texts)

2) the number of words per document

The output that has been produced by the query can be inspected for each individual document Selection of, for example, "`cmaelr4.m4.psdx`" in Figure 19 shows all the hits for query line 2 ("`matV2q`"), category "`4:[Vf-Sbj]`" and file "`cmaelr4.m4.psdx`", as in Figure 21.



QC 2 [matV2q] cat=[4:[Vf-Sbj]] file=[cmaelr4.m4.psdx]

text	4:[Vf-Sbj]
cmaelr3.m23.psdx	27
cmaelr4.m4.psdx	39

1 [1.3] DE INSTITUTIONE INCLUSARUM[s0004] [1.5] Here begynneth a tretys that is a rule and a forme of lyuynge perteynyng to a recluse . [s0006]

```
[IP-MAT [ADVP-LOC Here] [VBP begynneth] [NP-SBJ a tretys that is a rule and a forme of lyuynge perteynyng to a recluse] [. .] ]
-;1;a tretys that is a rule and a forme of lyuynge perteynyng to a
recluse;Nominal;IndefNP;begynneth;biginnen;AvK+ERK;unacc;-;-;AdvOth;Here;Imm;plain
```

2 [s0009] [1.10] Suster, thou hast ofte axed of me a forme of lyuyng accordyng to thyn estat, inasmuche as thou art enclosed. [1.11] That wolde Iesu thou haddist axid it of suche oon as coude haue fulfilled thyn desire . [1.12] Natheles after that simple felynge that God hath youen me, I shal write to the a forme gadert out of holy fadirs tradiciouns afore-tymes.

Figure 21 Result details for one document

6.8.4 *Result details: "per group"*

The option to view results per 'group' is not yet available, but is due to be implemented as soon as time permits.

The idea of dividing results in groups works like this:

1) The CorpusStudio program comes with a number of pre-defined 'groupings': collections of groups.
2) The user can define its own groupings for a particular corpus by (a) selecting that corpus in **Corpora** >> **Edit** and then pressing the **Grouping** button
3) Within the context of one 'grouping', each text is assigned to a particular group label on the basis of the text's meta data (e.g. genre, publication year, author)
4) Groups (and groupings) can be defined *after* a project has been executed

When the user wants to look at the results for a particular 'grouping', this grouping needs to be selected in the combobox on the **Results** >> **By group** page. What appears there is a table that is divided into rows and columns. There is one column for each 'group' that is part of the grouping (each group will show, even if it does not contain hits). And there's one row for each sub category in the results, if the current results have been divided into sub categories through the `ru:back()` function.

Each of the cells in the group table, then, represents the number of hits for a particular group-category combination. Clicking a cell brings up its contents: the files that have hits for this group/category combination, and the number of hits in them. Clicking one of the files shows the results for that file.

6.8.5 *Result details: "per division"*

The option to view results per 'division' is not yet available. This option will either be merged with the 'result per group' view (since that view already allows choosing a division, a grouping), or a new way to look at all defined groupings in one go will be created.

Note: Users are invited to actively participate in the development of this option.

7 Bibliography

Boag, Scott, Chamberlin, Don, Fernández, Mary F., Florescu, Daniela, Robie, Jonathan & Siméon, Jérôme. 2010. *XQuery 1.0: An XML Query Language (Second Edition)*: W3C Recommendation, <http://www.w3.org/XML/Query/#specs >.

Taylor, Ann, Warner, Athony, Pintzuk, Susan & Beths, Frank. 2003. *The York-Toronto-Helsinki Parsed Corpus of Old English Prose*, <http://www-users.york.ac.uk/~lang22/YCOE/YcoeHome.htm >.

W3C. 2010. *XML query (Xquery)*, <http://www.w3.org/XML/Query/#specs >.

Yao, Xuchen & Bouma, Gosse. 2010. “Mining Discourse Treebanks with XQuery”. Paper presented at *Ninth international workshop on treebanks and linguistic theories (TLT9)*, Tartu, Estonia.

8 Appendix

This appendix contains vital information for those who would like to run CorpusStudio on a different server for their own corpora (see 8.1). The second part of the appendix (section 8.2) contains samples of user-defined Xuery functions and the third part of the appendix (section 8.3) contains information about each of the `ru:` namespace Xquery functions that come with the current web application version of CorpusStudio.

8.1 Adapt CorpusStudio for use with other corpora

Adapting the CorpusStudio web application for one's own corpora is possible, provided the application is run from one's own Linux server.

8.1.1 Requirements

- **Server**. Linux server. The program has been developed under CentOS, but it should be possible to use other Linux versions.
- **Space**. The server should have enough space to hold one's corpora as well as CorpusStudio and the engine it runs on (Java + Tomcat). The program uses space on the hard drive under the /etc/project directory as well as under its deployment directory.
- **Memory**. The amount of RAM memory needed is roughly 1 Gbyte per processor plus perhaps 20 Gbyte to start with. The amount can be determined experimentally by creating a CRP that searches through the maximum number of input files (it's the number of files that counts, not so much their size), and then watching the amount of 'free memory left' in the cataline log file. E.g: `sudo tail -n 400 /usr/share/tomcat/logs/catalina.out | grep "Free m"`. Note: changes in the amount of memory that is available do *not* automatically become 'known' to Tomcat. The amount of memory that should be used by Tomcat must be specified by adding or adapting the JAVA_OPTS line in the tomcat.conf file. The CentOS server CLARIN's application has been built on has this file in the `/usr/share/tomcat/conf/` directory.
- **Processor**. The speed at which the web application will search through texts increases as more processing cores are available. But note that each processing core also increases the amount of RAM memory needed. Something like 8-20 cores should get a nice speed.
- **Software**. Java (version 1.7 or higher) and tomcat (version 7 or higher) should be installed on the server.

8.1.2 Installation

The installation of the CorpusStudio web application is Linux-system (and version) dependent. A virtual machine that has a CentOS system containing <u>no</u> installation of Java would require the following commands to prepare the machine for CorpusStudio:

```
# Arrange for sudo access
# - Add a user account for the person in charge (e.g "crphome")
# - Log in as root: su
# - Edit the /etc/sudoers file:
#   add a line for 'crphome' that has the same priorities as 'root'
# Changes to the principal's user account: get a directory 'webapps'
cd ~
mkdir webapps
#
# Update to latest installations
sudo yum update
```

```
# Install epel-release
sudo yum install -y epel-release
# Manual tweaking of epel-release:
sudo wget http://dl.iuscommunity.org/pub/ius/stable/RedHat/6/x86_64/ius-release-
     1.0-13.ius.el6.noarch.rpm
sudo rpm -Uvh ius-release*.rpm
# Install Java
sudo yum install -y java
# Install tomcat
sudo yum install -y tomcat
# Check the correct paths for Java, and create (or adapt) the file .bash_profile in
     the directory /home/crpstudio:
     # .bash_profile
     # Get the aliases and functions
     if [ -f ~/.bashrc ]; then
             . ~/.bashrc
     fi
     # User specific environment and startup programs
     PATH=$PATH:$HOME/bin:/usr/share/alpino/bin
     JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.45-28.b13.el6_6.x86_64/jre
     JRE_HOME=$JAVA_HOME
     PATH=$JAVA_HOME/bin:$PATH:$HOME/bin
     CATALINA_HOME=/usr/share/tomcat
     # Make the variables available
     export JAVA_HOME
     export JRE_HOME
     export PATH
# Adapt the file "ajp.conf" in directory /etc/httpd/conf.d so that it contains the
     lines with "crpp" and "crpstudio":
     ProxyRequests Off
     <Proxy *>
             Order deny,allow
             Deny from none
             Allow from localhost
     </Proxy>
     ProxyPass           /crpp   ajp://localhost:8009/crpp
     ProxyPassReverse    /crpp   ajp://localhost:8009/crpp
     ProxyPass           /crpstudio    ajp://localhost:8009/crpstudio
     ProxyPassReverse    /crpstudio    ajp://localhost:8009/crpstudio
# Go to directory /usr/share/tomcat/conf/Catalina
cd /usr/share/tomcat/conf/Catalina
# Create a file crpp.xml
     <?xml version='1.0' encoding='utf-8'?>
     <Context docBase="/home/crphome/webapps/CrppS" path="/crpp" reloadable="true"
          />
# Create a file crpstudio.xml
     <?xml version='1.0' encoding='utf-8'?>
     <Context antiJARLocking="true" docBase="/home/crphome/webapps/CrpStudio"
          path="/crpstudio" reloadable="true" />
# Adapt the existing file server.xml in directory /usr/share/tomcat/conf, adding a
     'localhost2' section after the existing 'localhost' one (depending on the main
     user's home directory; this makes sure the *.war files in the user's webapps
     directory are unpacked properly):
      <Host name="localhost2"  appBase="/home/crphome/webapps"
            unpackWARs="true" autoDeploy="true">
        <!-- SingleSignOn valve, share authentication between web applications
             Documentation at: /docs/config/valve.html -->
        <!--
        <Valve className="org.apache.catalina.authenticator.SingleSignOn" />
        -->
        <!-- Access log processes all example.
             Documentation at: /docs/config/valve.html
             Note: The pattern used is equivalent to using pattern="common" -->
        <Valve className="org.apache.catalina.valves.AccessLogValve"
     directory="logs"
               prefix="nederlab_access_log." suffix=".txt"
               pattern="%h %l %u %t &quot;%r&quot; %s %b" />
      </Host>
```

```
# Adapt the lines for JAVA_HOME and JAVA_OPTS in the file tomcat.conf that resides
     in directory /usr/share/tomcat/conf (this says Java uses 45 Gbytes maximum and
     1Gbytes minimum):
     # Where your java installation lives
     JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.45-28.b13.el6_6.x86_64/jre"
     # You can pass some parameters to java here if you wish to
     JAVA_OPTS="-Xmx45G -Xms1G"
# add user crphome to the 'tomcat' group
sudo usermod -a -G tomcat crphome
#
# log off and log in again as crphome to let the changes become effective
#
# Create a directory structure that at least contains:
# /etc/corpora      - this must contain file crp-info.json
# /etc/project      - this must contain a directory for each user
#                     Crpstudio will create a settings.json file
#                       inside each user's directory
# /etc/crpstudio    - this must contain a directory for each user
#                     it must also contain crpstudio-settings.json
#
# Make sure the directories created are owned by tomcat:tomcat
#
# Make sure the crp-info.json file is adapted to one's use and is validated
```

8.1.3 *File crp-info.json*

The file 'crp-info.json' must reside in the server's /etc/corpora directory. Here is a sample of its possible contents. It contains a definition for one corpus (the Lak language) and all its associated information.

```
{ "corpora": [
    {"lng": "lak_cyr", "name": "Lak (lbe), Cyrillic orthography", "eth": "lbe",
     "metavar": "caucasian",
     "parts": [
        {"name": "PCMLBE", "dir": "PCMLBE", "descr": "Parsed Corpus of Modern Lak",
                "metavar": "caucasian", "psdx": "", "folia": "",
                "url": "http://erwinkomen.ruhosting.nl/lbe/crp/"}

     ]}
  ],
  "metavar": [
    {"name": "caucasian",
     "variables": [
      {"name": "author",     "descr": "Author",                      "type": "txt",
               "loc": "header", "value": "descendant::titleStmt/@author"},
      {"name": "date",       "descr": "Estimated manuscript date",     "type": "int",
               "loc": "header", "value": "descendant::creation/@manuscript"},
      {"name": "editor",     "descr": "Editor",                      "type": "txt",
               "loc": "header", "value": "descendant::titleStmt/@editor"},
      {"name": "genre",      "descr": "Type of text",                "type": "txt",
               "loc": "header", "value": "descendant::creation/@genre"},
      {"name": "subtype",    "descr": "Time/genre/translated combi",   "type": "txt",
               "loc": "header", "value": "descendant::creation/@subtype"},
      {"name": "title",      "descr": "Title of document",           "type": "txt",
               "loc": "header", "value": "descendant::titleStmt/@title"},
      {"name": "translated","descr": "Is this translated?",          "type": "txt",
               "loc": "header", "value": "if (ends-
               with(descendant::creation/@subtype, 't')) then 'true' else 'false'"}
     ],
     "tagset": [
        {"title": "clsAny", "def": "IP*"},
        {"title": "clsMain","def": "IP-MAT*"},
        {"title": "clsSub", "def": "IP-SUB*"},
        {"title": "clsInf", "def": "IP-INF*"},
        {"title": "npSbj",  "def": "NP-SBJ*"},
        {"title": "npObj",  "def": "NP-OB*"},
        {"title": "npAny",  "def": "NP|NP-*"},
        {"title": "ppAny",  "def": "PP|PP-*"},
        {"title": "vbAny",  "def": "V*|AX*"},
        {"title": "vbFin",  "def": "VB[PD]*|AX*"}
     ],
     "groupings": [
       {"name": "subtype", "descr": "Combination of time and translated",
                "value": "$subtype"},
       {"name": "titleAlphabet", "descr": "First letter of title",
                "value": "if ($title = '') then '(n.t.)' else substring($title, 0,
                1)"},
       {"name": "authorAlphabet", "descr": "First letter of author surname",
                "value": "if ($author = '') then '(n.a.)' else substring($author, 0,
                1)"},
       {"name": "date", "descr": "Year of manuscript",
                "value": "$date"}
     ]
    }
  ]
}
```

The essential sections from the crp-info.json file:

- **corpora**. This defines key ingredients used by CorpusStudio:
  - **lng** the language code. This must coincide with a directory name: `/etc/corpora/lak_cyr`
  - **metavar** the name of the set of variables used for input restriction and grouping
  - **parts** one or more parts of the corpus, as identifiable by the sub directory (the attribute ‘`dir`’) within the corpus directory. Here: `/etc/corpora/lak_cyr/PCMLBE`
- **metavar**. This provides common variable names and their Xquery definitions.
  - **name** The name of this section. It must correspond with the value of the ‘`metavar`’ attribute in the ‘`corpora`’ section.
  - **variables** A list of all the variables that can be used in the definition of input restrictions as well as groupings. It is the variable names defined here that occur in the selection box for the input restriction definition of the Projects/Input page of CorpusStudio. Only the variables that are defined here may occur later in the ‘`groupings`’ section.
  - **tagset** This is, in the current version of CorpusStudio, a fixed list of names and the labels (POS or class tags) that are to be associated with them. The labels are generally defined by a ‘def’ specifiction. Where such a specification is not enough, a ‘`fs`’ specification may specify the name and the value of a feature. The ‘`vbFin`’ definition for ‘sonar’ files, for instance, is defined as:
    `{"title": "vbFin",     "def": "WW*", "fs": [{"alp/wvorm": "pv"}]}`
  - **groupings** The idea of result-groupings is in the process of being coded in CorpusStudio. The indicated section is a sample of the standard groupings every ‘`metavar`’ section must provide. The ‘`value`’ part of each grouping is an Xquery expression that results into a string label that identifies the ‘result group’ one particular file belongs to.

8.1.4 File crpstudio-settings.json

The directory /etc/crpstudio must contain a file named crpstudio-settings.json with the user/password information for the users that are allowed to make use of the version of CorpusStudio being used.

```
{ "users": [
    {"name": "erwin",   "password": "xx"},
    {"name": "erkomen", "password": "xx"},
    {"name": "guest",   "password": "xx"}
  ]
}
```

8.1.5 *File crpp-settings.json*

The /crpp 'machine' is the query-crunching motor serving the CorpusStudio web application. Its parameters are specified by the file crpp-settings.json that should be located in the main user's webapps directory (that would be `/home/crphome/webapps` in the running example). The format and contents of this settings file have been borrowed from the BlackLab code.[10] Here is a sample of its contents.

```
{ // Corpus Research Project Processor config file
  // ===============================================================
  // A list of IPs that will run in debug mode.
  "debugModeIps": [
    "127.0.0.1",       // IPv4 localhost
    "0:0:0:0:0:0:0:1" // IPv6 localhost
  ],

  // Specify the maximum number of "XqF" jobs allowed
  "maxparjobs": 18,

  // List of important directories
  "projectBase": "/etc/project",
  "corpusBase": "/etc/corpora",

  // A list of possible Engines, with the default engine identified
  "pinfo": {
    "ProjectType": "Xquery-psdx",
    "Xquery-psdx": {
      // Description
      "Descr": "Xquery with XML output",
      // Extension of query files
      "Qext": ".xq",
      // Extension of definition files
      "Dext": ".xq",
      // Extension of source files
      "SrcExt": ".psdx",
      // Start and end of comments
      "ComBeg": "(:", "ComEnd": ":)",
      // The 'engine' that is being used
      "Engine": "Xquery",
      // Default definition and period file locations
      "DefaultDefFile": "",
      "DefaultPerFile": "",
      // Text of default query
      "DefaultQuery": [
        "<TEI>",
        "{",
        " for $search in //eTree[ru:matches(@Label, $_matrixIP)]",
        "",
        "   (: Use your own 'let' definitions here :)",
        "   let $sbj := tb:SomeChildNo($search, $_subject, $_nosubject)",
        "",
        "   (: Define your 'where' definition here :)",
        "   where ( exists($sbj)",
        "         )",
        " return ru:back($search)",
        "}",
        "</TEI>"]
    },
    "FoLiA-xml": {
      // Description
      "Descr": "Xquery with XML output",
      // Extension of query files
      "Qext": ".xq",
      // Extension of definition files
      "Dext": ".xq",
```

[10] Blacklab is a program written by INL. See: https://github.com/INL/BlackLab.

```
      // Extension of source files
      "SrcExt": ".folia.xml",
      // Start and end of comments
      "ComBeg": "(:", "ComEnd": ":)",
      // The 'engine' that is being used
      "Engine": "Xquery",
      // Default definition and period file locations
      "DefaultDefFile": "",
      "DefaultPerFile": "",
      // Text of default query
      "DefaultQuery":  [
        "<FoLiA>",
        "{",
        "   (: Look for all main clauses within this sentence :)",
        "  for $search in //su[ru:matches(@cat, 'smain')]",
        "",
        "   (: Get the subject of this particular main clause :)",
        "  let $sbj := $search/child::su[@rel='su']",
        "",
        "   (: Do not allow empty (elided) subjects :)",
        "  where ( ",
        "           exists($sbj) and (count($sbj/child::su)>0)",
        "        )",
        "   (: Return the main clause :) ",
        "  return ru:back($search)",
        "}",
        "</FoLiA>"]
    }
  },
  // The location and parameters for each language-corpus index
  // ----------------------------------------------------------------
  // (missing indices will be skipped)
  "indices": {
    "lak_cyr": {
      "dir": "/etc/corpora/lak_cyr",
      "mayViewContent": true
    }
  },
  // Settings related to tuning server load and client responsiveness
  // ----------------------------------------------------------------
  "performance": {

    // Settings for job caching.
    "cache": {
      // How many search jobs will we cache at most?
      "maxNumberOfJobs": 200,

      // After how much seconds will a search job be removed from the cache?
      "maxJobAgeSec": 3600,

      // How many MB free memory the cache should aim for while cleaning up.
      "targetFreeMemMegs": 500,

      // When there's less free memory available than targetFreeMemMegs,
      // each time a job is created and added to the cache,
      // get rid of this number of older jobs in order
      // to free up memory
      "numberOfJobsToPurgeWhenBelowTargetMem": 2
    },

    // The minimum amount of free memory required to start a new search job.
    "minFreeMemForSearchMegs": 100,

    // The maximum number of jobs a user is allowed
    // to have running at the same time.
    "maxRunningJobsPerUser": 100
    }
  }}
```

8.1.6 *Software sources*

The source for the Java/JavaScript software that needs to be compiled consists of three parts, all of which can be downloaded from https://github.com/ErwinKomen:

1) CrpxProcessor - The Xquery search 'engine'. Can run as command-line application.
2) CrppServer - The 'service' shell around CrpxProcessor. Requires the latter.
3) CrpStudio - CorpusStudio web application. Uses CrpxProcessor internally too.

The three programs have been developed under NetBeans, but it should be possible to get them running under different IDE's. The 'github' sources only provide the Java and JavaScript sources.

Each of the programs listed above contains a 'lib' directory with .jar files it makes use of (not on github):

Package	Libraries needed

CrpxProcessor	`log4j-1.2.15.jar`
	`saxon9-s9api.jar`
	`saxon9.jar`
	`saxon9-dom.jar`

CrppServer	`log4j-1.2.15.jar`
	`CrpxProcessor/dist/CrpxProcessor.jar`
	`CrpxProcessor/lib/saxon9-s9api.jar`
	`CrpxProcessor/lib/saxon9.jar`
	`CrpxProcessor/lib/saxon9-dom.jar`

CrpStudio	`log4j-1.2.15.jar`
	`velocity-tools-view-2.0.jar`
	`velocity-1.7.jar`
	`commons-collections-3.2.1.jar`
	`commons-lang-2.4.jar`
	`vtd-xml.jar`
	`CrpxProcessor/dist/CrpxProcessor.jar`

8.1.7 *Running*

The CorpusStudio web application can be started up by (re-)starting tomcat. A possible startup script could be "crpstudio.sh":

```
sudo service tomcat restart
sudo cp ~/*.empty /usr/share/tomcat/logs/catalina.out
sudo tail -n 400 /usr/share/tomcat/logs/catalina.out
```

This assumes the presence of an empty file called "catalina.empty" in the user's home directory, it assumes one is running CentOS (hence the "sudo") and one's account has sudo-rights established.

The application should be running once the Tomcat log file logs a message like "`INFO: Server startup in 9309 ms`". See `/usr/share/tomcat/logs/catalina.out`.

8.2 Useful Xquery function definitions

While one is definitely at liberty to make one's own Xquery functions, this section offers a number of useful functions that could be used for CorpusStudio.

Note that some of these functions use built-in Xquery functions which are described in section 8.3. Since they make use of the CorpusStudio in-built facilities, such user-defined Xquery functions will obviously not work outside of CorpusStudio.

8.2.1 *Functions for Psdx projects*

A number of Xquery functions have been created for projects that take texts in the Psdx format as input. The table that is to appear here (in due course) will provide an overview of these functions, divided over the definition files in which they can be found.

Section	User-function	Description
`Standard_xq-def`	*(declaration of common variables)*	

`Coref_xq-def`	`tb:Coref`
	`tb:CorefLink`
	`tb:CorefDist`
		Find nodes of a particular 'coref' type
		Nodes of a 'coref' with a specified ref distance
		Get the antecedent distance of a node

`RefState_xq-def`	`tb:RefStateA`
	`tb:RefStateLUA`
	`tb:RefStateAct`
	`tb:RefStateP`
		Calculate referential states according to different protocols, including *estimation* of the referential state

`Relations_xq-def`	`tb:SomeChild`
	`tb:AllStarred`
	`tb:IsStarred`
	`tb:SpecialChild`
	`tb:AllChildren`
	`tb:FirstChild`
	`tb:SomeCHildNo`
	`tb:SomeDescendant`
	`tb:AnyFirstChild`
	`tb:Contains`
	`tb:ChildLabel`
	`tb:HasChild`
	`tb:HasDescendant`
	`tb:HasDescOrSelf`
	`tb:HasOnlyChild`	

`Report_xq-def`	`tb:Labelled`
	`tb:Sentence`
	`tb:GrRole`
	`tb:Phrases`
	`tb:Syntax`
	`tb:ChainText`
	`tb:English`
	`tb:HasEnglish`
	`tb:ConstType`	

`Tests_xq-def`	`tb:IsMain`
	`tb:GetCfType`
	`tb:IsInFinite`
	`tb:IsExpl`
	`tb:IsAnyExpl`
	`tb:HasLocContr`
	`tb:HasFocAdv`
	`tb:HasAnchor`
	`tb:HasLeaf`
	`tb:HasFeature`
	`tb:IsCata`	

Future releases of CorpusStudio are intended to allow making use of these functions by incorporating these files into one's project through the Projects / Definitions / New / **Definition Wizard** option.

8.2.2 *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
};
```

8.3 Built-in functions for Xquery-psdx projects

Several Xquery functions have been built into CorpusStudio that serve to facilitate working with FoLiA-xml and Xquery-psdx projects. The functions are described in this section. All functions are part of the `ru:` namespace.

8.3.1 Function ru: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 obligatory one for the CorpusStudio web application. It should be placed at the end of the main `for … let … where … return` (FLOWR) loop, as part of the `return` statement. The function returns a node that is supplied with additional attributes, such as the identifier of the ndArg1 node.[11]

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 could, in principle, 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/@* }
};
```

The built-in `ru:back()` function does a bit more when it comes to the second argument strMsg. When strMsg contains a semi-colon-separated list of values, these may internally be interpreted as the feature values for an *xml*-coded "Result Database" that is to be produced as a result of the query.

[11] The node that is returned is traditionally the `<forest>` node, as used in the Psdx format. The exact name of the node has become unimportant, however, since the *xml* output produced by ru:back() is not used to build a new *xml* file. The attributes returned in the node that is produced by ru:back() *are* used for different purposes:
1) The first argument ndArg1 is used as the basis from which a syntactic outline of the result is built
2) The second argument strMsg is positioned as a message with its 'hit', and it may be used as feature set for Result database creation
3) The third argument strCat is used to divide the results over the categories it provides.

8.3.2 *Function ru: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 conversino 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. (Case conversion can also be accomplished by using String functions that are part of the Xquery version-1-or higher package.)
A few combinations are allowed too: ‘Lcase+OE’ and ‘Ucase+OE’.

8.3.3 *Function ru:docroot*

Definition:	**ru:docroot**()	
Types:	output	node()
Status:	*superseded by* `ru:header`	

Description: Return the highest node in the current document. This is the `<TEI>` root in .psdx files, and the `<FoLiA>` root in .folia.xml files. This function is included for backward compatibility but it should not be used in the Corpus Studio web application. The Windows version of CorpusStudio works file-by-file, so that it makes sense to have a pointer to the document as a whole. But the web implementation works part-by-part (where one part is a sentence or the header). Access the the information in the header is provided by the `ru:header`() function.

8.3.4 *Function ru: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.

8.3.5 *Function ru:header*

Definition:	**ru:header**()	
Types:	output	node()

Description: Return the 'header' node of the current document. This is the `<teiHeader>` node in .psdx files, while it is the `<metadata>` node in .folia.xml files. This header may contain metadata information that is of interest to the user. Some corpora have their metadata in separate .imdi or .cmdi files. This information can be accessed using the `ru:mdi`() function.

8.3.6 *Function ru: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. (The current version of Corpusstudio web does not yet have the user interface to handle lexicons that have been made.) 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.

8.3.7 Function ru: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.	

8.3.8 Function ru: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`.	

8.3.9 Function ru:message

Definition:	**ru:message**(strText)	
Types:	strText	xs:string
	output	xs:boolean

Description:	The output of this function is always *true*. When the function is evaluated (which happens when it occurs in a boolean expression), a message box is displayed with the message strText, and the user can press "OK".

8.3.10 Function ru:mdi

Definition:	**ru:mdi**()	
Types:	output	node()
Status:	*implemented*	
Description:	Return the root node of the metadata stored in the .imdi or .cmdi file associated with the current document. The files in the CGN (corpus of spoken Dutch) have their metadata in .imdi files, while the .cmdi files typically (but not necessarily) co-occur with folia files. If no .imdi or .cmdi file is available the function returns an empty node.	

8.3.11 Function ru: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** (when it will be implemented)		