GÖTEBORGS UNIVERSITET
INST FÖR SVENSKA SPRÅKET

GU-ISS-2017-01

Korp 6 - Technical Report

Martin Hammarstedt, Johan Roxendal, Maria Öhrman,
Lars Borin, Markus Forsberg, Anne Schumacher

Forskningsrapporter från institutionen för svenska språket, Göteborgs universitet
Research Reports from the Department of Swedish

ISSN 1401-5919

www.svenska.gu.se/publikationer/GU-ISS

CONTENTS

1 Acknowledgements **3**

2 The Korp frontend **4**
2.1 Setting up the Korp Frontend . . . 4
2.1.1 Configuration . . . 4
2.1.2 Installing the frontend . . . 4
2.1.3 Localization . . . 4
2.1.4 Modes . . . 5
2.1.5 Corpora . . . 6
2.1.6 Customizing extended search . . . 8
2.1.7 Parallel Corpora . . . 9
2.1.8 Rendering attribute values in the statistics-view . . . 10
2.1.9 Autocompletion menu . . . 10
2.1.10 Word picture . . . 10
2.1.11 Map . . . 11
2.1.12 News widget . . . 12
2.1.13 Summary of settings . . . 12
2.2 Developing the Korp Frontend . . . 15
2.2.1 Source code . . . 15
2.2.2 Setting up the development environment . . . 15
2.2.3 Localization . . . 15
2.2.4 Map . . . 16
2.2.5 Building a distribution . . . 16

3 The Korp backend **17**
3.1 Requirements . . . 17
3.2 Installing the required software . . . 17
3.2.1 Web server (Apache) . . . 17
3.2.2 Python . . . 17
3.2.3 Subversion . . . 18
3.2.4 Corpus Workbench . . . 18
3.2.5 MariaDB or MySQL . . . 18
3.3 Installing the CGI script . . . 18
3.4 Configuring Korp . . . 19
3.5 Installing a corpus . . . 19
3.5.1 Adding some info about the corpus . . . 20
3.6 Requirements for your corpus structure . . . 20
3.7 Parallel corpora . . . 21
3.8 MySQL tables . . . 21
3.9 Relations for the Word Picture . . . 21
3.10 Lemgram index . . . 24
3.11 Corpus time span information . . . 24

4 Web API **26**
4.1 Introduction . . . 26
4.2 Queries . . . 26
4.3 Commands supported by the web service . . . 26
4.3.1 General Information . . . 27
4.3.2 Corpus Information . . . 27
4.3.3 Concordance . . . 28

4.3.4 Word Picture 30
4.3.5 Word Picture Sentences 31
4.3.6 Statistics 31
4.3.7 Lemgram Statistics 32
4.3.8 Log-Likelihood Comparison 33
4.3.9 Trend Diagram 34

1 ACKNOWLEDGEMENTS

This work and research was supported by Malin Ahlberg, Peter Ljunglöf, Olof Olsson, Dan Rosén, Roland Schäfer and Jonatan Uppström. We thank our colleagues and former colleagues who made great contributions to Korp.

2 THE KORP FRONTEND

2.1 SETTING UP THE KORP FRONTEND

This section describes how to get the Korp frontend up and running on your own machine and presents the available customization. In this step it is necessary to have a backend with at least one corpus installed. For testing purposes, Språkbankens Korp backend may be enough. It is also assumed that you have a web server available (such as Apache or Lighttpd).

Download the latest release[1] (the code is distributed under the MIT license[2].)

2.1.1 CONFIGURATION

The main configuration file of Korp is `config.js`. In this file we have configuration for where the backend is located, what features should be turned on or off etc. Corpora configuration is done in the modes files. There is more information about that later in this document.

The config-file must be started with creating a settings object:

```
var settings = {};
```

All additional configuration parameters are added to this object. For example: `settings.defaultLanguage = "en"`

Available settings will be described in feature sections and there is also a summary of all settings in section 2.1.13.

2.1.2 INSTALLING THE FRONTEND

Make sure your web server serves the Korp frontend distribution folder (with your customizations in).

2.1.3 LOCALIZATION

In `app/translations` there are several files containing translations for different parts of the application.

Files prefixed with `locale` and controls translations are hard-coded into the application and thus it should not be necessary to change these if only customization is done. The files prefixed with `corpora` however are translations of corpora attributes and values and must be replaced with data suitable for the specific set of corpora the Korp installation serves. The files are JSON structures that for each language ties a **translation key** to a particular **string** in that language. You should start with empty corpora translation files and then add the translations as you add corpora.

The translations folder also contains Python script - `check_locale_files.py` - that makes sure that each set of translation files has each translation key present in all different languages.

[1] https://spraakbanken.gu.se/eng/research/infrastructure/korp/distribution
[2] https://opensource.org/licenses/MIT

2.1.3.1 ADDING LANGUAGES

To add a new language in the frontend, for example Lithuanian, add a `corpora-lt.json` and `locale-lt.json`. `locale-lt.json` may be copied from an existing locale-file and then translated. Then add the language in `config.js`:

```
settings.languages = ["sv", "en", "lt"];
```

To make Lithuanian the default language, use:

```
settings.defaultLanguage = "lt";
```

2.1.3.1 ANGULAR.JS LOCALE

To enable full localization (dates in a datepicker for example), an extra file is necessary. Download `angular-locale_lt.js` from here:

Angular i18n[3]

Put the file in `app/translations/`.

2.1.4 MODES

Each Korp installation has a series of *Modes* in the top left corner, which are useful for presenting different faces of Korp that might have different layouts or functionality. In the Swedish version the parallel corpora have their own mode because their KWIC results don't mix particularly well with the 'normal' results.

2.1.4.1 COMMON.JS

After `config.js`, but before any mode configuration, `modes/common.js` is loaded. This may include definitions which are used in several modes s.a. a set of attributes. This helps to keep `config.js` clean.

2.1.4.2 ADDING MODES

Relevant setting fields are `settings.visibleModes` and `settings.modeConfig`. The former controls how many modes are visible in the header (the rest are hidden away in a menu). The latter looks like this:

[3] https://github.com/angular/bower-angular-i18n

```
[
  {
    localekey: "modern_texts",
    mode: "default"
  },
  {
    localekey: "parallel_texts",
    mode: "parallel"
  },
  {
    localekey: "faroese_texts",
    mode: "faroe"
  }
]
```

The `localeKey` key corresponds to a key from the localization files. The `mode` key is the mode identifier and is used to load a script file from the `modes` folder corresponding to that ID. So if you click the modeSelectors 'parallel' entry, the page refreshes and the `modes/parallel_mode.js` will be loaded.

The mode called `default` will always be loaded first. If there is no need for more than one mode, leave `settings.modeConfig` empty.

2.1.5 CORPORA

The config file contains the corpora declaration, wherein the available corpora are declared together with information about which metadata fields are searchable in them. Adding a test corpus is as simple as:

```
    settings.corpora = {};
    settings.corpora["testcorpus"] = {
        id: "testcorpus",
        title: "The Korp Test Corpus",
        description: "A test corpus for testing Korp.",
        within: {"sentence": "sentence"},
        attributes: {
            pos: {
                label: "pos",
                opts: {
                    "is": "=",
                    "is_not": "!="
                }
            }
        },
        structAttributes: {
        }
    }
```

- `id`: Short form title, should correspond to the key name of the definition.
- `title`: Long form title, for display in the corpus chooser.
- `description`: For display in the corpus chooser.

- `within`: What are the structural elements of the corpus? See `defaultWithin` in settings summary in section 2.1.13 for format and more information.
- `attributes`: each key here refers to a word attribute in Corpus Workbench. Their values are JSON structures with a few attributes of their own; they are concerned with generating the necessary interface widgets in Extended Search, display in sidebar and statistics. They are:
  - `label`: a translation key for the attributes name
  - `limitedAccess`: `boolean`, it will not be possible to select this corpus unless a user is logged in and has the correct credentials.
  - `displayType`: set to `'hidden'` to fetch attribute, but never show it in the frontend. See `hideSidebar`, `hideStatistics`, `hideExtended` and `hideCompare` for more control.
  - `translationKey`: you can declare a prefix for the translation keys of the dataset here. This is so the corpora translation file doesn't get too messy: a simple kind of namespacing.
  - `extendedTemplate`: Angular template used in conjunction with the `extendedController` to generate an interface widget for this attribute. See customizing extended search in section 2.1.6.
  - `extendedController`: Angular controller that is applied to the template. See customizing extended search in section 2.1.6.
  - `opts`: this represents the auxiliary select box where you can modify the input value. See `defaultOptions` in settings summary in section 2.1.13 for format and more information.
  - `hideSidebar`: Default `false`. Hide attribute in sidebar.
  - `hideStatistics`: Default: `false`. Should it be possible to compile statistics based on this attribute?
  - `hideExtended`: Default: `false`. Should it be possible to search using this attribute in extended?
  - `hideCompare`: Default: `false`. Should it be possible to compare searches using this attribute?
  - `type`: Possible values:
    - "set" - The attribute is formatted as "|value1|value2|". Include contains and not contains in `opts`. In the sidebar, the value will be split before formatted. When using compile / `groupby` on a "set" attribute in a statistics request, it will be added to `split`.
    - "url" - The value will be rendered as a link to the URL and possibly truncated if too long.
  - `pattern`: HTML snippet with placeholders for replacing values. Available is `key` (attribute name) and `value`. Also works for sets. Example: `'<p style="margin-left: 5px;"><%=val.toLowerCase()%></p>'`
  - `display`: How to display attribute in sidebar. Currently only supported for sets and `expandList` (see below). In the future more ways to display might be added here.
    - `expandList`: Render set as a list where the first element is visible and a button to show or hide the rest of the elements.
      - `splitValue`: Function to split up values if there are sets within the set. Example: `function(value){ return value.split(','); }`
      - `searchKey`: If `display.expandList.internalSearch` is set to `true`, links will be rendered to search for the value in Korp, using this key in the CQP-expression. Omit to use same key as attribute name.
      - `joinValues`: Interleave this string with all values on the row.
      - `stringify`: Optional override of outer `stringify`.
      - `linkAllValues`: Should the `internalSearch` be enabled for all values or only the first one in the set?
      - `internalSearch`: Alternative function to transform the attribute key and value to a CQP-expression. Example: `function(key,value){ '[' + key + '="' + val + '"]' }`
  - `internalSearch`: `boolean`. Should the value be displayed as a link to a new Korp search? Only works for sets. Searches for CQP-expression: `[<attrName> contains "<regescape(attrValue)>"]`

- `externalSearch`: Link with placeholder for replacing value. Example `https://spraakbanken.gu.se/karp/#?search=extended||and|sense|equals|<%= val %>`
- `order`: Order of attribute in the sidebar. Attributes with a lower `order`-value will be placed over attributes with a higher `order`-value.
- `stringify`: How to pretty-print attribute. Example: `function(str){ return util.lemgramToString(str, true); }`
- `isStructAttr`: `boolean`. If `true` the attribute will be treated as a structural attribute in all sense except it will be included in the `show` query parameter instead of `show_struct` for KWIC requests. Useful for structural attributes that extend to smaller portions of the text, such as name tagging.
- optional keys and values that can be utilized in the extendedTemplate / extendedController. See customizing extended search in section 2.1.6.

- `structAttributes`: refers to higher level metadata attributes. Examples include author, publishing year, URL etc. Structural attributes support the same settings as the word attributes.
- `customAttributes`: creates fields in the sidebar that have no corresponding attribute in the backend. Useful for combining two different attributes. All settings concerning sidebar format for normal attributes apply in addition to:
  - `customType`: `"struct"` / `"pos"` - decides if the attribute should be grouped under word attributes or text attributes.
  - `pattern`: Same as pattern for normal attributes, but `struct_attrs` and `pos_attrs` also available. Example: `'<p style="margin-left: 5px;"><%=struct_attrs.text_title - struct_attrs.text_description%></p>'`

2.1.6 Customizing extended search

It is possible to customize the standard input field of extended search into anything. Any key can be added to an attribute to be provided to the `extendedController` / `extendedTemplate`. Simple example:

```
var myReusableTemplate = '<div><input ng-if="inputType == 'text'" type
    ="text"><input ng-if="inputType == 'number'" type="number"></div>';

var myController = function($scope, $location) {
    // $scope.inputType is available here also
    // dependency injection of Angular services such as $location are
       possible
};

settings.corpora["testcorpus"] = {
    id: "testcorpus",
    title: "The Korp Test Corpus",
    description: "A test corpus for testing Korp.",
    attributes: {
        myAttr: {
            label: "myAttr",
            extendedTemplate: myReusableTemplate,
            extendedController: myController,
            inputType: "text"
        }
    }
};
```

However, `extendedController` is not mandatory and only shown in this example for documentation purposes.

2.1.6.1 TEMPLATE REQUISITES

In order for your template to work, it must set its value in `scope.model`, for example by using `ng-model="model"` for input-fields.

2.1.6.2 AUTOC

A directive that autocompletes word forms to lemgrams or senses using Karp. Used in the following way:

```
<autoc placeholder="placeholder" type="lemgram" model="model"
 disable-lemgram-autocomplete="disableLemgramAutocomplete"
  text-in-field="textInField">
```

Where `type` may be either `lemgram` or `sense`. `model` will be the selected lemgram / sense. `textInField` will be actual user input (user did not select anything). Placeholder will contain the pretty-printed lemgram / sense. It is also possible to make the element fall back to a "normal" text field by setting `disableLemgramAutocomplete` to `false`.

2.1.6.3 ESCAPER

`escaper` is a directive that takes the user's input and escapes any regexp characters before saving it to `scope.model`. When the model changes it automatically de-escapes any regexp characters before showing the value to the user. Input must be saved to `scope.input` for it to work. Example: `<input ng-model="input" escaper>`

2.1.7 PARALLEL CORPORA

Parallel corpora need to have its own mode. Use `modes/parallel_mode.js`, but replace the corpus definitions. Change the line `var start_lang = "swe";` to whatever language that should be the default search language.

The corpora declaration for parallel corpora is different in some important ways. Example:

```
settings.corpora["saltnld-sv"] = {
    id: "saltnld-sv",
    lang: "swe",
    linkedTo: ["saltnld-nl"],
    title: "SALT svenska-nederländska",
    context: context.defaultAligned,
    within: {
        "link": "meningspar"
    },
    attributes: {},
    structAttributes: {}
};
```

```
settings.corpora["saltnld-nl"] = {
    id: "saltnld-nl",
    lang: "nld",
    linkedTo: ["saltnld-sv"],
    title: "SALT svenska-nederländska",
    context: context.defaultAligned,
    within: {
        "link": "meningspar"
    },
    attributes: {},
    structAttributes: {},
    hide: true
};
```

The corpus configuration for parallel corpora needs to make explicit the links between the declared corpora. This is done using the `linkedTo` property. A corpus may declare any amount of links to other corpora. Also notice the `lang` property, used for building the correct language select menu. The `within` attribute should use the `"link": "meningspar"` value. Also note the `hide` attribute which prevents both subcorpora from being listed in the corpus chooser widget.

2.1.8 Rendering attribute values in the statistics-view

The appearance of the leftmost columns of hits in the stats table can be controlled by editing `app/config/statistics_config.js`. These change according to the 'compile based on' select menu and might need a different stringification method depending on the chosen attribute. Make sure the function returns valid html. A known issue is that annotations containing spaces when searching for more than one token works less than perfect.

2.1.9 Autocompletion menu

Korp features an autocompletion list for searches in the Simple Search as well as in Extended for those corpus attributes configured to use `autoc`-directive (see autoc-section in section 2.1.6.2). This is implemented using an Angular.js directive `autoc` that calls Karp's autocompletion function. Using Karp, Korp can autocomplete senses and lemgrams. To disable autocompletion in simple search use `settings.autocomplete = false`.

2.1.10 Word picture

The word picture-config object looks like this:

```
setting.wordPictureConf = {
    pos_tag: [table_def1, tabledef2...]
}
```

where `table_def` is an array of objects that describe the resulting word picture table. `table_def1` above might look like this:

```
[
    {rel: "subject", css_class: "color_blue"},
    "_",
    {rel: "object", css_class: "color_purple"},
    {rel: "adverbial", css_class: "color_purple", field_reverse: false
       }
]
```

The "_" refers to the placement of the hit in the table order. The value for `rel` refers to a key in `settings.wordpictureTagset` looking like this:

```
settings.wordpictureTagset = {
    // the actual value for the pos-tag must be given in this object
    pos_tag: "vb",

    subject: "ss",
    object: "obj",
    adverbial: "adv"
}
```

The values are the actual relations returned by the backend. The relation used is determined by `field_reverse`. If `field_reverse` is `false` (default), `dep` is used, else `head`. If you find yourself with a table full of the search word just flip the `field_reverse` switch.

`css_class` simply gives a class to the column, useful for applying background color. The last supported attribute is `alt_label`, used for when another value than the relation name should be used for the table header.

2.1.11 MAP

Korp has two versions of the map.

1. An old version where the resolution from name to location are done client-side. When map is enabled all names will be fetched for the current search result context (names occuring in matching sentences for example). To fetch the names, pos-tags are used. Which pos-tag values that should match are configurable. Then the names are looked up in `components/geokorp/dist/data/places.json` and if they occur in the file we place them on the map. `places.json` should be replaced or extended since it contains mostly Swedish places. The problem with this approach is that we get lots of errors, proper names are often mistaken for location names for example. This feature will be removed.
2. A newer version that uses annotations to get locations. The user selects rows from the statistics table and points derived from different rows will have different colors. The selected corpora must have structural attributes with location data in them. The format is `Fukuoka;JP;33.6;130.41667` - the location name, the country, latitude and longitude separated by `;`.

   Also the name of the attribute must contain `"__"` and `"geo"` to show up in the list of supported attributes.

The map is unstable and will change in upcoming releases, for example the old version of the map will be removed.

settings.enableMap - `boolean`. The old version of the map should be enabled.
settings.mapPosTag - For the old version of the map. Which pos-tag values should be used to find names. Example: `["PM", "NNP", "NNPS"]`
settings.newMapEnabled - `boolean`. The new version of the map should be enabled.
settings.mapCenter - Where the center of the map should be located when user opens map. Example:

```
settings.mapCenter = {
  lat: 62.99515845212052,
  lng: 16.69921875,
  zoom: 4
};
```

2.1.12 NEWS WIDGET

By setting a `newsDeskUrl` on settings, the news widget is enabled. The widget simply fetches a json-file from the given URL. Short example of such a file, including only one news item with its title and body in two languages and a date:

```
[
    {
        "h": {
            "en": "<p>Longer description in English</p>",
            "sv": "<p>Längre beskrivning på svenska</p>"
        },
        "t": {
            "en": "English Title",
            "sv": "Svensk Titel"
        },
        "d": "2017-03-01"
    }
]
```

2.1.13 SUMMARY OF SETTINGS

Settings are required unless specified to be optional.

autocomplete - Boolean. Enable autocomplete (see **autoc**-directive) for simple search.

languages - Array of supported interface language codes s.a. `["en", "sv"]`

defaultLanguage - The default interface language. Example: `"sv"`

downloadFormats - Available formats of KWIC-download. See supplied `config.js`.

downloadFormatParams - Settings for KWIC-download. See supplied `config.js`.

wordAttributeSelector - `"union"` / `"intersection"`. Controls attribute list in extended search. Use all selected corpora *word* attributes or only the attributes common to selected corpora.

structAttributeSelector - Same as **wordAttributeSelector**, but for structural attributes.

reduceWordAttributeSelector - Same as **wordAttributeSelector**, but for the "compile based on"-configuration in statistics. Warning: if set to `"union"`, the statistics call will fail if user selects an attribute that is not supported by a selected corpus.

reduceStructAttribute_selector - Same as **reduceWordAttributeSelector**, but for structural attributes.

newsDeskUrl - See **News widget**. Optional.

wordpictureTagset - See **Word picture**

wordPictureConf - See **Word picture**

visibleModes - See **Adding modes**

modeConfig - See **Adding modes**

primaryColor - Background color in corpus chooser, CSS color. Example: `"rgb(221, 233, 255)"`

primaryLight - Background color of settings area, CSS color. Example: `"rgb(221, 233, 255)"`

defaultOverviewContext - The default context for KWIC-view. Use a context that is supported by the majority of corpora in the mode (URLs will be shorter). E.g.: `"1 sentence"`. For corpora that do not support this context an additional parameter will be sent to the backend based on the `context`-setting in the corpus.

defaultReadingContext - Same as **defaultOverviewContext**, but for the context-view. Use a context larger than the **defaultOverviewContext**.

defaultWithin - An object containing the structural elements of a corpus. Default within is used unless a corpus overrides the setting using `within`. Example:

```
settings.defaultWithin = {
    "sentence": "sentence",
    "paragraph": "paragraph"
};
```

In simple search, we will search within the default context and supply extra information for the corpora that do not support the default context.

In extended search, the default `within` will be used unless the user specifies something else. In that case the user's choice will be used for all corpora that support it and for corpora that do not support it, a supported `within` will be used.

cqpPrio - An array of attributes to order and-clauses in CQP-expressions by. Order the array by how specific an attribute is in increasing order. `word` will probably be the most specific attribute and should be placed last, while POS-tags will be near the beginning. A well ordered list will speed up queries significantly.

defaultOptions - Object containing the default operators for extended search. May be overridden for each attribute by setting `opts` on the attribute-configuration. The object keys are translation keys and values are the frontend's internal representation of CQP. Example:

```
settings.defaultOptions = {
    "is": "=",
    "is_not": "!=",
    "starts_with": "^=",
    "contains": "_=",
    "ends_with": "&=",
    "matches": "*=",
    "matches_not": "!*=",
}
```

Explanation of internal format:

	Internal representation	CQP	Note
is	[key = “val”]	[key = “val”]	
is not	[key != “val”]	[key != “val”]	
starts with	[key ^= “val”]	[key = “val.*“]	
contains	[key _= “val”]	[key = “.*val.*“]	
ends with	[key &= “val”]	[key = “val.*“]	
matches	[key *= “val”]	[key = “val”]	Used with `escaper`-directive, regexp
matches not	[key !*= “val”]	[key != “val”]	special characters will not be escaped.

cgiScript - URL to Korp CGI-script

downloadCgiScript - URL to Korp download CGI-script

wordpicture - Boolean. Enable word picture.

statisticsCaseInsensitiveDefault - Boolean. Selects case-insensitive for “compile based on” by default.

inputCaseInsensitiveDefault - Boolean. Selects case-insensitive for simple search by default.

corpora - See **Corpora**

corpusListing - After specifying all corpora in a modes-file use: `settings.corpusListing = new CorpusListing(settings.corpora);` to enable the configuration. For parallel corpora use: `settings.corpusListing = new ParallelCorpusListing(settings.corpora, parseLocationLangs());`

corporafolders - Create a directory-structure in corpus chooser. Example:

```
settings.corporafolders.foldername = {
    title: "A folder",
    contents: ["corpus1", "corpus2"],
    description: "Optional description"
};

settings.corporafolders.foldername.subfolder = {
    title: "A sub folder",
    contents: ["corpus3", "corpus4"]
}
```

preselectedCorpora - An array of corpus (internal) names or folder names. Given corpora and corpora in folders will be selected on load. To select only a subfolder write `folder.subfolder`.

enableMap - See **Map**.

mapPosTag - See **Map**.

newMapEnabled - See **Map**.

mapCenter - See **Map**.

2.2 Developing the Korp Frontend

Here is where we present details on how to install development dependencies for the Korp frontend and how to build and distribute the frontend code.

2.2.1 Source code

The source code is available on Github[4].

2.2.2 Setting up the development environment

The Korp frontend uses a plethora of technologies and has a corresponding amount of dependencies. Luckily, a set of package managers do all the heavy lifting and so you should be up and running in no time. Simply follow these steps:

- Install node, preferably through a package manager[5] or download an installer[6].
- Using npm (the node package manager, which comes bundled with node), install global dependencies: `npm install -g bower grunt-cli`.
- Your machine probably has ruby installed already. Otherwise, install it[7]. Run `gem install sass`.
- Fetch the latest Korp source release.
- `cd` to the Korp folder you just checked out and run `npm install` in order to fetch the local dependencies. This includes libs for compiling the CoffeeScript files, building, running a dev server, as well as the required client side javascript libs utilized directly by Korp.

You are now ready to start the dev server, do so by running `grunt serve`. In you browser, open `http://localhost:9000` to launch Korp. Now, as you edit the Korp code, CoffeeScript and Sass files are automatically compiled as required, additionally causing the browser window to be reloaded to reflect the new changes.

2.2.3 Localization

Korp does runtime DOM manipulation when the user changes language. Using an Angular filter to specify which translation key looks like this:

```
<div>{{'my_key' | loc}}</div>
```

[Deprecation warning] Before the Angular approach we used the `rel` attribute, like so (but you shouldn't any more): `<span rel="localize[translation_key]">...</span>`

[4] https://github.com/spraakbanken/korp-frontend/
[5] https://nodejs.org/en/download/package-manager/
[6] https://nodejs.org/download/
[7] http://www.ruby-lang.org/en/downloads/

2.2.4 MAP

Modify the map with configuration, `scripts/map_controllers.coffee` or the Geokorp-component located in `components/geokorp`. Geokorp wraps Leaflet[8] and adds further functionality such as integration with Angular, marker clustering, marker styling and information when selecting a marker.

2.2.5 BUILDING A DISTRIBUTION

Building a distribution is as simple as running the command `grunt`. A `dist` folder is created. These are the files to use for production deployment. The build system performs concatenation and minimization of JavaScript and CSS source files, giving the resulting code a lighter footprint. Note that you don't actually have to use the build system (as long as you compile the CoffeeScript-files and Sass-files by some other means).

[8] http://leafletjs.com/

3 THE KORP BACKEND

This document describes how to get the Korp backend up and running on your own machine.

Start by downloading the Korp backend[9]. The code is distributed under the MIT license[10].

3.1 REQUIREMENTS

To use the basic features of the Korp backend you need the following:

- A web server (e.g. Apache[11])
- Python 2.6[12] or newer (but not 3.x)
- MySQL-python[13]
- Corpus Workbench[14] (CWB) 3.2 or newer
- Subversion[15]

To use the additional features such as the Word Picture you also need the following:

- MariaDB[16] or MySQL[17]

3.2 INSTALLING THE REQUIRED SOFTWARE

The following information assumes that you are running Ubuntu, but will most likely work for any Linux-based OS.

3.2.1 WEB SERVER (APACHE)

Please follow the instructions on installing Apache[18].

3.2.2 PYTHON

Python should already be installed by default on your system.

The required MySQL library for Python can be installed by running the following command in a terminal:

```
sudo apt-get install python-mysqldb
```

[9] https://spraakbanken.gu.se/eng/research/infrastructure/korp/distribution
[10] https://opensource.org/licenses/MIT
[11] http://httpd.apache.org/
[12] http://python.org/
[13] http://pypi.python.org/pypi/MySQL-python
[14] http://cwb.sourceforge.net/beta.php
[15] http://subversion.apache.org/
[16] https://mariadb.org/
[17] http://www.mysql.com/
[18] https://www.maketecheasier.com/install-and-configure-apache-in-ubuntu

3.2.3 SUBVERSION

Subversion is needed to download the source code for Corpus Workbench. To install, run the following command in a terminal:

```
sudo apt-get install subversion
```

3.2.4 CORPUS WORKBENCH

You will need the latest version of CWB for unicode support. Install by following these steps:

Check out the latest version of the source code from subversion by running the following command in a terminal:

```
svn co https://cwb.svn.sourceforge.net/svnroot/cwb/cwb/trunk cwb
```

Navigate to the new cwb directory and run the following command:

```
sudo ./install-scripts/cwb-install-ubuntu
```

CWB is now installed, and by default you will find it under `/usr/local/cwb-X.X.X/bin` (where `X.X.X` is the version number). Confirm that the installation was successful by typing:

```
/usr/local/cwb-X.X.X/bin/cqp -v
```

CWB needs two directories for storing the corpora. One for the data, and one for the corpus registry. You may create these directories wherever you want, but from here on we will assume that you have created the following two:

```
/corpora/data
/corpora/registry
```

If you're not running Ubuntu or if you run into any problems, refer to the INSTALL text file in the cwb dir for further instructions.

3.2.5 MARIADB OR MYSQL

Both MariaDB and MySQL can usually be installed using your Linux distribution's package manager.

3.3 INSTALLING THE CGI SCRIPT

Once you're done installing the required software, you are ready to install korp.cgi as a CGI script on your web server. This page[19] might be of help if you don't know how. Remember to give the file the right permissions by running the command `chmod 755 korp.cgi` if needed.

[19] https://docs.python.org/2/howto/webservers.html#setting-up-cgi-on-your-own-server

Be sure to put the included “concurrent” directory and the korp_config.py file in the same directory as korp.cgi.

3.4 Configuring Korp

First you might need to change the path to the Python executable in the first line of korp.cgi. To figure out the path of your local Python installation, you can run the following command in a terminal:

```
which python
```

Except for that line, you normally don't need to change anything else in korp.cgi.

The rest of the configuration is done by editing the file korp_config.py, where you at least have to set the following variables:

- `CQP_EXECUTABLE`
  The absolute path to the CQP binary. By default `/usr/local/cwb-X.X.X/bin/cqp`
- `CWB_SCAN_EXECUTABLE`
  The absolute path to the cwb-scan-corpus binary. By default `/usr/local/cwb-X.X.X/cwb-scan-corpus`
- `CWB_REGISTRY`
  The absolute path to the CWB registry files. This is the `/corpora/registry` folder you created before.

If you are using the database functionality, you also need to set the following:

- `DBNAME`
  The name of the MySQL database where the corpus data will be stored.
- `DBUSER & DBPASSWORD`
  Username and password for accessing the database.

Try visiting the korp.cgi URL in a web browser and see if it works. With no corpora installed you will only get some version info from the script.

3.5 Installing a corpus

Included with the Korp backend you should find a small test corpus, in a file named `testcorpus.vrt`. This contains a small excerpt from the Swedish Wikipedia article about Språkbanken. The VRT format is a format used as input for Corpus Workbench to create a binary corpus file. To import the VRT file, run the following commands in the directory containing the VRT file (change the paths to reflect the paths on your system):

```
mkdir /corpora/data/testcorpus
/usr/local/cwb-3.4.7/bin/cwb-encode -s -p - -d /corpora/data/
   testcorpus -R /corpora/registry/testcorpus -x -c utf8 -f testcorpus
   .vrt -P word -P pos -S sentence:0+n -S text:0+title+date+datefrom+
   dateto+timefrom+timeto
/usr/local/cwb-X.X.X/bin/cwb-makeall -V -r /corpora/registry
   TESTCORPUS
```

For more information about encoding corpora, see this tutorial[20].

Once you have a corpus installed and korp.cgi set up, you are ready to use the script for querying corpora. To test it out, try visiting the following URL (after changing it to match your server of course):

```
http://[location_on_your_server]/korp.cgi?command=query&cqp=[word=%22
    korpusar%22]&corpus=TESTCORPUS&start=0&end=0&defaultcontext=1%20
    sentence&indent=2
```

You should get a JSON structure containing one matching sentence from the test corpus.

This guide assumes that you are going to use the Korp Frontend to communicate with korp.cgi, and therefore no further information will be given about the Korp API here.

3.5.1 Adding some info about the corpus

For Korp to show the number of sentences and the date when a corpus was last updated, you have to manually add this information to a file. Create a file called ".info" in the directory containing the CWB corpus data:

```
/corpora/data/testcorpus/.info
```

In this file, add the following lines (editing the values to match your material), and be sure to end the file with a blank line:

```
Sentences: 12345
Updated: 2017-04-28
FirstDate: 2001-01-16 00:00:00
LastDate: 2001-01-30 23:59:59
```

Once this file is in place, korp.cgi will be able to display this information.

3.6 Requirements for your corpus structure

To use the basic concordance features of Korp there are no particular requirements regarding the markup of your corpora. Everything should however be in UTF-8.

To use the **Word Picture** functionality your corpus must follow the following format:

- The structural annotation marking sentences must be named "sentence".
- Every sentence annotation must have an attribute named "id" with a unique (within the corpus) value.

To use the **Trend Diagram** functionality, your corpus needs to be annotated with date information using the following four structural attributes: `text:datefrom`, `text:timefrom`, `text:dateto`, `text:timeto`. The date format should be *YYYYMMDD*, and the time format *hhmmss*. You need to use the full date-time resolution, even if your material only has information about years, meaning that a material dated 2006 would have the following values:

- `text:datefrom`: 20060101
- `text:timefrom`: 000000
- `text:dateto`: 20061231

[20] http://cwb.sourceforge.net/files/CWB_Encoding_Tutorial.pdf

- `text:timeto`: 235959

3.7 PARALLEL CORPORA

To encode parallel corpora, i.e. two corpora that are linked on some structural level, you need a link attribute in both corpora. Every link should have a unique ID attribute. A link with a particular ID will be linked to the link with the same ID in the other corpus.

In the following example, you have two corpora, an English and a Swedish one: `testcorpus_en` and `testcorpus_sv`. You also have a structural link attribute `<link id="123">` linking parts of the corpora. First you run the following commands:

```
/usr/local/cwb-X.X.X/bin/cwb-align -v -o testcorpus_en.align -V
   link_id testcorpus_en testcorpus_sv link
/usr/local/cwb-X.X.X/bin/cwb-align -v -o testcorpus_sv.align -V
   link_id testcorpus_sv testcorpus_en link

/usr/local/cwb-X.X.X/bin/cwb-align-encode -v -d /corpora/data/
   testcorpus_en/ testcorpus_en.align
/usr/local/cwb-X.X.X/bin/cwb-align-encode -v -d /corpora/data/
   testcorpus_sv/ testcorpus_sv.align
```

Secondly, you need to manually edit the corpus registry files. At the end of the file `/corpora/registry/testcorpus_en`, add the following line:

```
ALIGNED testcorpus_sv
```

And at the end of `/corpora/registry/testcorpus_sv`, add the following line:

```
ALIGNED testcorpus_en
```

3.8 MYSQL TABLES

This section describes the database tables needed to use the Word Picture and Lemgram index functions.

3.9 RELATIONS FOR THE WORD PICTURE

If you want to use the Word Picture feature in Korp, you need to have the word picture data in your database. The data consists of head-relation-dependent triplets and frequencies. For every corpus, you need five database tables. The table structures are as follows:

```
Table name: relations_CORPUSNAME
Charset:    UTF-8

Columns:
    id              int                 A unique ID (within this
       table)
    head            int                 Reference to an ID in the
       strings table (below). The head word in the relation
    rel             enum(...)           The syntactic relation
    dep             int                 Reference to an ID in the
       strings table (below). The dependent in the relation
    freq            int                 Frequency of the triplet (
       head, rel, dep)
    bfhead          bool                True if head is a base
       form (or lemgram)
    bfdep           bool                True if dep  is a base
       form (or lemgram)
    wfhead          bool                True if head is a word
       form
    wfdep           bool                True if dep is a word form

Indexes:
    (head, wfhead, dep, rel, freq, id)
    (dep, wfdep, head, rel, freq, id)
    (head, dep, bfhead, bfdep, rel, freq, id)
    (dep, head, bfhead, bfdep, rel, freq, id)
```

```
Table name: relations_CORPUSNAME_strings
Charset:    UTF-8

Columns:
    id              int                 A unique ID (within this
       table)
    string          varchar(100)        The head or dependent
       string
    stringextra     varchar(32)         Optional preposition for
       the dependent
    pos             varchar(5)          Part-of-speech for the
       head or dependent

Indexes:
    (string, id, pos, stringextra)
    (id, string, pos, stringextra)
```

```
Table name: relations_CORPUSNAME_rel
Charset:    UTF-8

Columns:
    rel             enum(...)            The syntactic relation
    freq            int                  Frequency of the relation

Indexes:
    (rel, freq)
```

```
Table name: relations_CORPUSNAME_head_rel
Charset:    UTF-8

Columns:
    head            int                  Reference to an ID in the
       strings table. The head word in the relation
    rel             enum(...)            The syntactic relation
    freq            int                  Frequency of the pair (
       head, rel)

Indexes:
    (head, rel, freq)
```

```
Table name: relations_CORPUSNAME_dep_rel
Charset:    UTF-8

Columns:
    dep             int                  Reference to an ID in the
       strings table. The dependent in the relation
    rel             enum(...)            The syntactic relation
    freq            int                  Frequency of the pair (rel
       , dep)

Indexes:
    (dep, rel, freq)
```

```
Table name: relations_CORPUSNAME_sentences
Charset:    UTF-8

Columns:
    id              int                 An ID from
       relations_CORPUSNAME
    sentence        varchar(64)         A sentence ID (see the
       section about corpus structure above)
    start           int                 The position of the first
       word of the relation in the sentence
    end             int                 The position of the last
       word of the relation in the sentence

Indexes:
    id
```

Unless you customize the word picture in the Korp frontend, you should use the SUC2 tagset[21] for part of speech, and syntactic relations from MAMBA[22].

The `sentences` table contains sentence IDs for sentences containing the relations, with start and end values to point out exactly where in the sentences the relations occur (1 being the first word of the sentence).

3.10 LEMGRAM INDEX

The lemgram index is an index of every lemgram in every corpus, along with the number of occurrences. This is used by the frontend to grey out auto-completion suggestions which would not give any results in the selected corpora. The lemgram index consists of a single MySQL table, with the following layout:

```
Table name: lemgram_index
Charset:    UTF-8

Columns:
    lemgram      varchar(64)        The lemgram
    freq         int                Number of occurrences
    freq_prefix  int                Number of occurrences as a
       prefix
    freq_suffix  int                Number of occurrences as a
       suffix
    corpus       varchar(64)        The corpus name

Indexes:
    (lemgram, corpus, freq, freq_prefix, freq_suffix)
```

3.11 CORPUS TIME SPAN INFORMATION

Korp can show a diagram in the corpus selector showing the distribution of material over time. For this to work, or if you want to use the Trend Diagram feature, you need to add token-per-time-span data to the

[21] http://spraakbanken.gu.se/parole/Docs/sucnyckel.suc2
[22] http://stp.ling.uu.se/~nivre/swedish_treebank/dep.html

database. A sample SQL file for the test corpus is included in the download, and can be imported to MySQL using the following command:

```
cat timespan.sql | mysql your_database_name
```

If you manually want to create the required tables, use the following table layout:

```
    Table name: timedata
    Charset:    UTF-8

    Columns:
        corpus    varchar(64)        The corpus name
        datefrom  datetime           Full from-date and time
        dateto    datetime           Full to-date and time
        tokens    int                Number of tokens between from-
           date and (including) to-date

    Indexes:
        (corpus, datefrom, dateto)
```

```
    Table name: timedata_date
    Charset:    UTF-8

    Columns:
        corpus    varchar(64)        The corpus name
        datefrom  date               From-date (only date part)
        dateto    date               To-date (only date part)
        tokens    int                Number of tokens between from-
           date and (including) to-date

    Indexes:
        (corpus, datefrom, dateto)
```

4 WEB API

4.1 INTRODUCTION

Korp's web service is used to query Språkbanken's corpora, and is accessible at the following address:

```
https://spraakbanken.gu.se/ws/korp
```

4.2 QUERIES

Queries to the web service are done using HTTP GET requests:

```
https://spraakbanken.gu.se/ws/korp?command=...&corpus=...&...
```

The service responds with a JSON object.

4.3 COMMANDS SUPPORTED BY THE WEB SERVICE

Below is a list of the commands supported by the web service, and what arguments are accepted. The arguments for each command is presented as a bulleted list. The following example

- `a = ...`
- `[opt] q = "XF"`
- `[multi] x = ...`

means that *q* is optional, *x* may take multiple values separated by comma, and that the final URL query string becomes:

```
?a=...&q=XF&x=...,...
```

The following arguments can be used together with any of the commands:

- `[opt] encoding = <Character encoding used when communicating with Corpus Workbench. Default is UTF-8.>`
- `[opt] indent = <Indent the JSON response to make it human readable. By default a compact un-indented JSON is used.>`
- `[opt] callback = <A string which will surround the returned JSON object, sometimes necessary with AJAX requests.>`
- `[opt] cache = <Set to 'false' to bypass cache.>`

In case of an error (e.g. non-existing command or corpus, or syntax errors) the server responds with the following:

```
{
  "ERROR": {
    "type": <Error type>,
    "value": <Error message>
  }
}
```

For every request a time value will always be included in the JSON object, indicating the execution time in seconds:

```
{
  "time": <Seconds>
}
```

4.3.1 General Information

Get information about available corpora, which corpora are protected, and which version of CQP that is used.

- `command = "info"`

Returns

```
{
  "cqp-version": <CQP version>,
  "corpora": [<List of corpora on the server>],
  "protected_corpora": [<List of which of the above corpora that
     are password protected>]
}
```

Example

`?command=info`[23]

4.3.2 Corpus Information

Fetch information about one or more corpora.

- `command = "info"`
- `[multi] corpus = <Corpus name>`

Returns

[23] https://spraakbanken.gu.se/ws/korp?command=info&indent=4

```
{
  "corpora": {
    <Corpus name>: {
      "attrs": {
        "p": [<List of positional attributes>],
        "s": [<List of structural attributes>],
        "a": [<List of align attributes, for linked corpora>]
      },
      "info": {
        "Charset": <Character encoding of the corpus>,
        "FirstDate": <Date and time of the oldest dated text in
           the corpus>,
        "LastDate": <Date and time of the newest dated text in the
            corpus>,
        "Size": <Number of tokens in the corpus>,
        "Sentences": <Number of sentences in the corpus>,
        "Updated": <Date when the corpus was last updated>
      }
    }
  "total_size": <Total number of tokens in the above corpora>,
  "total_sentences": <Total number of sentences in the above
     corpora>
  }
}
```

Example

`?command=info&corpus=ROMI,PAROLE`[24]

4.3.3 CONCORDANCE

Do a concordance search in one or more corpora.

- command = "query"
- [multi] corpus = <Corpus name>
- cqp = <CQP query>
- start = <First result to return; used for pagination>
- end = <Last result to return; used for pagination>
- [opt] defaultcontext = <Context to show>
- [opt, multi] context = <Context to show for specific corpora, overriding the default. Specified using the format 'corpus:context'>
- [opt, multi] show = <Positional attributes to show>
- [opt, multi] show_struct = <Structural attributes to show>
- [opt] cut = <Limit total number of hits to this number>
- [opt] defaultwithin = <Prevent search from crossing boundaries of the given structural attribute>
- [opt, multi] within = <As above, but for specific corpora, overriding the default. Specified using the format 'corpus:attribute'>

[24] https://spraakbanken.gu.se/ws/korp?indent=4&command=info&corpus=ROMI,PAROLE

- [opt] sort = <Sort the results within each corpus by: search word ('keyword'), left ('left')or right context ('right'), random ('random'), or a given positional attribute>
- [opt] random_seed = <Numerical value for reproducible random order (used together with sort=random above)>
- [opt] incremental = <Return results incrementally when set to 'true' and more than one corpus is specified>

The positional attribute "word" will always be shown, even if omitted.

Returns

```
{
  "hits": <Total number of hits>,
  "corpus_hits": {
    <Corpus>: <Number of hits>,
    ...
  }
  "kwic": [
    {
      "match": {
        "start": <Start position of the match within the context>,
        "end": <End position of the match within the context>,
        "position": <Global corpus position of the match>,
      },
      "tokens: [
        {
          "word": <Word form>,
          "pos": <Part of speech annotation>,
          "baseform": <Baseform annotation>,
          ...
        },
        <Second token of the row>,
        ...
      ],
      <If aligned corpora>
      "aligned": {
        <Aligned corpus>: [<List of tokens>], ...
      }
    },
    <Second concordance row>,
    ...
  ]
}
```

Example showing the 10 first sentences/links matching the CQP query "och" [] [pos="NN"]

- Query SUC3 and show part of speech and baseform:

  `?command=query&corpus=SUC3&start=0&end=9&defaultcontext=1+sentence&cqp="och"+[]+[pos="NN"]&show=msd,lemma`[25]

[25] https://spraakbanken.gu.se/ws/korp?indent=4&command=query&corpus=SUC3&start=0&end=9&defaultcontext=1+sentence&cqp=%22och%22+%5B%5D+%5Bpos=%22NN%22%5D&show=msd,lemma

- Query SALTNLD-SV and show part of speech + the linked Dutch sentence:

  `?command=query&corpus=SALTNLD-SV&start=0&end=9&context=1+link&cqp="och "+[]+[pos="NN"]&show=saltnld-nl`[26]

4.3.4 WORD PICTURE

Get typical dependency relations for a given lemgram or word.

- `command = "relations"`
- `[multi] corpus = <Corpus name>`
- `word = <Word or lemgram to look up>`
- `[opt] type = <Search type: 'word' (default)or 'lemgram'>`
- `[opt] min = <Cut-off frequency>`
- `[opt] max = <Max number of results, 0 = unlimited>`
- `[opt] incremental = <Incrementally return progress updates when the calculation for each corpus is finished>`

Returns

```
{
  "relations": [
    {
      "corpus": [
        <List of corpora containing hits>
      ],
      "dep": <Dependent lemgram or word>,
      "depextra": <Dependent prefix>,
      "deppos": <Dependent part of speech>,
      "freq": <Number of occurrences>,
      "head": <Head lemgram or word>,
      "headpos": <Head part of speech>,
      "mi": <Lexicographer's mutual information score>,
      "rel": <Relation>,
      "source": [
        <List of IDs, for getting the source sentences>
      ]
    },
    ...
  ]
}
```

Example for getting dependency relations for the lemgram ge..vb.1

`?command=relations&word=ge..vb.1&type=lemgram&corpus=ROMI`[27]

[26] https://spraakbanken.gu.se/ws/korp?indent=4&command=query&corpus=SALTNLD-SV&start=0&end=9&defaultcontext=1+link&cqp=%22och%22+%5B%5D+%5Bpos=%22NN%22%5D&show=saltnld-nl

[27] https://spraakbanken.gu.se/ws/korp?indent=4&command=relations&word=ge..vb.1&type=lemgram&corpus=ROMI

4.3.5 Word Picture Sentences

Given the source ID for a relation (from a Word Picture query), return the sentences in which this relation occurs.

- `command = "relations_sentences"`
- `[multi] source = <List of source IDs (from a Word Picture query)>`
- `[opt] start = <Start row>`
- `[opt] end = <End row>`

Returns

Returns a structure identical to a regular concordance query.

Example requesting all sentences containing the relation (ge..vb.1, SS, instruktion..nn.1)

`?command=relations_sentences&source=ROMI:332579`[28]

4.3.6 Statistics

Given a CQP query, calculates the frequency for one or more attributes. Both absolute and relative frequency are calculated.

- `command = "count"`
- `cqp = <CQP query>`
- `[multi] groupby = <Positional and/or structural attributes by which the hits should be grouped>`
- `[multi] corpus = <Corpus name>`
- `[opt] defaultwithin = <Prevent search from crossing boundaries of the given structural attribute>`
- `[opt, multi] within = <As above, but for specific corpora, overriding the default. Specified using the format 'corpus:attribute'>`
- `[opt, multi] ignore_case = <Attributes for which the case will be ignored>`
- `[opt] start = <Start row; used for pagination>`
- `[opt] end = <End row; used for pagination>`
- `[opt] incremental = <Incrementally return progress updates when the calculation for each corpus is finished>`

For instances when you want to calculate statistics for *every* token in one or several corpora, the `count_all` command should be used instead, since it is several times faster than using `count`. This command takes the same arguments as `count`, except for *cqp* which is not used.

If you want to base your statistics on one single token in a multi token query, prefix that token with an @: *[pos = "JJ"] @[pos = "NN"]*.

Returns

[28] https://spraakbanken.gu.se/ws/korp?indent=4&command=relations_sentences&source=ROMI:332579

```
{
  "corpora": {
    <Corpus>: {
      "absolute": {
        <attribute1/attribute2/attribute3/...>: <Absolute
           frequency>,
        ...
      },
      "relative": {
        <attribute1/attribute2/attribute3/...>: <Relative
           frequency>,
        ...
      },
      "sums": {
        "absolute": <Absolute sum>,
        "relative": <Relative sum>
      }
    },
  },
  "total": {
    "absolute": {
      <attribute1/attribute2/attribute3/...>: <Absolute frequency
         >,
      ...
    },
    "relative": {
      <attribute1/attribute2/attribute3/...>: <Relative frequency
         >,
      ...
    },
    "sums": {
      "absolute": <Absolute sum>,
      "relative": <Relative sum>
    }
  }
  "count": <Total number of different values>
}
```

Example returning frequencies for the different word forms of the lemgram `ge..vb.1`

```
?command=count&corpus=ROMI&cqp=[lex+contains+"ge..vb.1"]&groupby=word&
ignore_case=word
```[29]

4.3.7 LEMGRAM STATISTICS

Return the number of occurrences of one or more lemgrams in one or more corpora.

- `command = "lemgram_count"`
- `[multi] lemgram = <Lemgram to look up>`

[29] https://spraakbanken.gu.se/ws/korp?indent=4&command=count&corpus=ROMI&cqp=%5Blex+contains+%22ge..vb.1%22%5D&groupby=word&ignore_case=word

- [opt, multi] corpus = <Corpus name (if omitted: all corpora)>
- [opt] count = <Occurrences as regular lemgrams ('lemgram'), as prefix ('prefix'), or as suffix ('suffix')>
- [opt] incremental = <Incrementally return progress updates when the calculation for each corpus is finished>

Returns

```
{
  "<lemgram>": <Number of occurrences>,
  ...
}
```

Example returning the number of occurrences of the lemgrams `ge..vb.1` and `ta..vb.1` in a single corpus

`?command=lemgram_count&lemgram=ge..vb.1,ta..vb.1&corpus=ROMI`[30]

4.3.8 Log-Likelihood Comparison

Compare the results of two different searches by using log-likelihood.

- command = "loglike"
- set1_cqp = <CQP query 1>
- set2_cqp = <CQP query 2>
- [multi] groupby = <Positional and/or structural attributes by which the hits should be grouped>
- [multi] set1_corpus = <Corpus name 1>
- [multi] set2_corpus = <Corpus name 2>
- [opt] max = <Max number of hits>
- [opt] incremental = <Incrementally return progress updates when the calculation for each corpus is finished>

Returns

```
{
  "average": <Average for log-likelihood>,
  "loglike": {
    "<Value>": <Log-likelihood value>,
    ...
  },
  "set1": {
    "<Value>": <Absolute frequency>,
    ...
  },
  "set2": {
    "<Value>": <Absolute frequency>,
    ...
  }
}
```

[30] https://spraakbanken.gu.se/ws/korp?indent=4&command=lemgram_count&lemgram=ge..vb.1,ta..vb.1&corpus=ROMI

A positive log-likelihood value indicates a relative increase in set2 compared to set1, while a negative value indicates a relative decrease.

Example comparing the nouns of two different corpora

```
?command=loglike&set1_cqp=[pos="NN"]&set2_cqp=[pos="NN"]&groupby=word&max
=10&set1_corpus=ROMI&set2_corpus=GP2012
```
[31]

4.3.9 Trend Diagram

Show the change in frequency of one or more expressions over time.

- `command = "count_time"`
- `cqp = <CQP query>`
- `[opt] subcqp# = <Where # is a number. Sub-queries to the query above. Any number of numbered subcqp-arguments can be used.>`
- `[multi] corpus = <Corpus name>`
- `[opt] granularity = <Time resolution. y = year (default), m = month, d = day>`
- `[opt] incremental = <Incrementally return progress updates when the calculation for each corpus is finished>`

If `subcqp` is omitted, the result will only contain frequency information for the head CQP query in `cqp`. If one or more sub-queries are specified using `subcqp1`, `subcqp2` and so on, the result will contain frequency information for these as well.

The result is returned both per corpus, and a total.

Returns

```
{
  "corpora": {
    "<Corpus>": [
      {
        "relative": {
          "<Date>": <Relative frequency>,
          "<Date>": <Relative frequency>,
          ...
        },
        "sums": {
          "relative": <Sum, relative frequency>,
          "absolute": <Sum, absolute frequency>
        },
        "absolute": {
          "<Date>": <Absolute frequency>,
          "<Date>": <Absolute frequency>,
          ...
        }
      },
```

[31] https://spraakbanken.gu.se/ws/korp?command=loglike&set1_cqp=%5Bpos=%22NN%22%5D&set2_cqp=%5Bpos=%22NN%22%5D&groupby=word&max=10&set1_corpus=ROMI&set2_corpus=GP2012

```
            {
              "cqp": "<Sub-CQP query>",
              "relative": {
                "<Date>": <Relative frequency>,
                "<Date>": <Relative frequency>,
                ...
              },
              "sums": {
                "relative": <Sum, relative frequency>,
                "absolute": <Sum, absolute frequency>
              },
              "absolute": {
                "<Date>": <Absolute frequency>,
                "<Date>": <Absolute frequency>,
                ...
              }
            },
            <More structures like the one above, one per sub-query>
          ],
          ...
        },
        "combined": [
          {
            "relative": {
              "<Date>": <Relative frequency>,
              "<Date>": <Relative frequency>,
              ...
            },
            "sums": {
              "relative": <Sum, relative frequency>,
              "absolute": <Sum, absolute frequency>
            },
            "absolute": {
              "<Date>": <Absolute frequency>,
              "<Date>": <Absolute frequency>,
              ...
            }
          },
          {
            "cqp": "<Sub-CQP query>",
            "relative": {
              "<Date>": <Relative frequency>,
              "<Date>": <Relative frequency>,
              ...
            },
            "sums": {
              "relative": <Sum, relative frequency>,
              "absolute": <Sum, absolute frequency>
            },
```

```
            "absolute": {
              "<Date>": <Absolute frequency>,
              "<Date>": <Absolute frequency>,
              ...
            }
          },
          <More structures like the one above, one per sub-query>
        ]
      }
```

Example showing how the use of "tsunami" and "flodvåg" has changed over time in Göteborgs-Posten

```
?command=count_time&cqp=[((lex+contains+"tsunami..nn.1"+|+lex+contains+"
flodvåg..nn.1"))]&corpus=GP2001,GP2002,GP2003,GP2004,GP2005,GP2006,GP2007
,GP2008,GP2009,GP2010,GP2011,GP2012&subcqp0=[lex+=+'|tsunami..nn.1|']&
subcqp1=[lex+=+'|flodvåg..nn.1|']
```
[32]

[32] https://spraakbanken.gu.se/ws/korp?command=count_time&cqp=%5B((lex+contains+%22tsunami%5C.%5C.nn%5C.1%22+%7C+lex+contains+%22flodv%C3%A5g%5C.%5C.nn%5C.1%22))%5D&corpus=GP2001%2CGP2002%2CGP2003%2CGP2004%2CGP2005%2CGP2006%2CGP2007%2CGP2008%2CGP2009%2CGP2010%2CGP2011%2CGP2012&subcqp0=%5Blex+%3D+'%5C%7Ctsunami%5C.%5C.nn%5C.1%5C%7C'%5D&subcqp1=%5Blex+%3D+'%5C%7Cflodv%C3%A5g%5C.%5C.nn%5C.1%5C%7C'%5D

GU-ISS, Forskningsrapporter från Institutionen för svenska språket, är en oregelbundet utkommande serie, som i enkel form möjliggör spridning av institutionens skriftliga produktion. Det främsta syftet med serien är att fungera som en kanal för preliminära texter som kan bearbetas vidare för en slutgiltig publicering. Varje enskild författare ansvarar för sitt bidrag.

GU-ISS, Research reports from the Department of Swedish, is an irregular report series intended as a rapid preliminary publication forum for research results which may later be published in fuller form elsewhere. The sole responsibility for the content and form of each text rests with its author.

Forskningsrapporter från institutionen för svenska språket, Göteborgs universitet
Research Reports from the Department of Swedish

ISSN 1401-5919

www.svenska.gu.se/publikationer/GU-ISS