The Open Corpus Workbench (CWB)

CQPweb System Administrator's Manual

— CQPweb Version 3.3.5 and above —

Andrew Hardie
http://cwb.sourceforge.net/

February 2021

Contents

1 Installing CQPweb

This chapter contains only the minimum amount of information you need to get CQPweb up and running; it touches on many aspects of the system, but does not go into full detail. Further information can be found in other sections of this manual.

1.1 What you will need

You'll need a machine with a Unix-style operating system. CQPweb has been installed successfully, to our knowledge, on Mac OS X; Sun OS; Debian; Ubuntu; SUSE; and Fedora.

Windows compatibility (for Windows 10) is planned but not yet achieved. For the moment, there are two possible ways to run CQPweb on Windows:

With Cygwin. Install Cygwin, then install CQPweb, and all its dependencies, on top of Cygwin.

With the Windows Subsystem for Linux. Enable the WSL, then install a Linux distribution such as Debian or Ubuntu via the Microsoft Store app. Then install CQPweb in that distribution.

Other software you need to have installed:

- Apache or some other web server
- MySQL (v5.0 at minimum, preferably v8.0 or higher) or MariaDB (version 10.0 or higher). Other SQL DB systems, like MS-SQL, SQlite, or Postgresql, don't work with CQPweb.
- PHP (version 7.3+).
- Corpus Workbench (see 1.5 for details)
- *Optionally*, Perl and the CWB-Perl modules (see 1.6 for details)
- R
- *Optionally*, standard Unix command-line tools accessible on the shell path: **awk**, **sort** and **head**; either GNU versions, or versions compatible with them. See 2.3.11 for how to enable their use if they are present and you wish to use them.

A word of warning: Installing CQPweb on a shared server where you do not have full control over the setup can sometimes be problematic. For instance, as will be explained, there are certain options (especially in PHP and MySQL/MariaDB) which need to be set in certain ways for CQPweb to work properly. If you don't have control over these settings, then it will be very difficult to get things working properly. For instance, MySQL/MariaDB servers can be configured to block some of the permissions that you will need to have - so if you can't reconfigure those permissions you will not get very far.

1.2 Your web browser

CQPweb uses modern web technologies. It may not work properly if accessed by means of an out-of-date browser.

In particular, the browsers that are supported (in the sense that we'll always do our best to make sure the CQPweb interface works as intended in these browsers) are the following:

- Google Chrome, or the related *Chromium* browser.
- Mozilla Firefox.
- Microsoft Edge.
- Apple Safari.

Of course, browsers that share the rendering engine of one or other of the above will also be OK.

As of first quarter 2019 (CQPweb v3.2.32 and above) we no longer support Internet Explorer. We now ignore any errors and issues reported from users of Internet Explorer.

You will need to use a suitable browser to access the administrative functions of CQPweb. Likewise, the users of your server will need to access the site and perform their queries, etc., using an approved browser.

1.3 Hardware requirements

It is difficult to generalise about what hardware you will need. Even a (relatively) low-powered computer should be able to run CQPweb with small-to-medium sized corpora; a modern desktop system should also be fine with pretty large corpora - as should most laptops as long as they have a big enough hard drive (see below). As a general rule, having less-than-ideal hardware will cause things to run slowly, rather than not run at all.

One potential bottleneck is working memory for big database operations. For very intensive operation, the SQL DB (MySQL/MariaDB) requires lots of memory; if it can't get enough RAM, it will use hard disk space as temporary storage instead; if it uses up all available hard disk space, it will fall over mid-operation, possibly with a very uninformative error message (e.g. saying that it can't read from or write to a particular temporary file). These "big database operations" tend to be aspects of the corpus setup procedure - especially frequency table creation - rather than anything that a user who is not an administrator can set in motion.

⟪*XREF to the CACHE chapter – section on disk space and advising on partitioning etc*⟫ **TODO**

How much disk space precisely the SQL DB might need for some of CQPweb's big setup procedures is difficult to say for certain - it depends, apart from anything else, on how far the SQL daemon can get just with RAM. However, as a rule of thumb, you should aim to run CQPweb's SQL database on a disk or partition which has free space equal to a multiple of the raw-text size of the corpus you are working with. (This will also be more than enough for cache space and CWB indexes of your corpus, if the cache and data directories are on the same disk or partition.)

For example, on one of the CQPweb development machines - a relatively modern server with plenty of RAM - the SQL DB needed approximately 4 GB of temporary disk space for the process of building frequency tables for a corpus whose raw text took up 1 GB. So if you are regularly dealing with corpora of that size (on the order of 100,000,000 words), it is a good idea to have, say, ten times as much space available as your raw text takes up.

1.4 Installing the webscripts

Download CQPweb by going to the CWB website (`http://cwb.sourceforge.net/download.php#gui`) and doing one of the following:

- Get a release of CQPweb as a compressed download file, then decompress it. This will get you a stable version of the program, but one that might be quite old. If your system does not have Subversion installed on it, then this is the only option.

- Use Subversion to *export* a copy of the program from our code repository. You can get either the *trunk* version (cutting-edge version, may contain bugs) or one of the older *branches* (may lack recently added features but should have fewer bugs), as follows:
  - `svn export http://svn.code.sf.net/p/cwb/code/gui/cqpweb/trunk CQPweb`
  - `svn export http://svn.code.sf.net/p/cwb/code/gui/cqpweb/branches/X.Y CQPweb`
  - (for a branch, replace "X.Y" with the version number of the branch you want; the CWB website will indicate what branches are available/recommended).
- Use Subversion to *check out* a copy of the program. The difference between checking out and exporting is that a checked-out copy can be automatically updated if the version in the repository changes. This makes updating the system easy. This is, therefore, recommended. Again, you can check out either the trunk or a branch.
  - `svn co http://svn.code.sf.net/p/cwb/code/gui/cqpweb/trunk CQPweb`
  - `svn co http://svn.code.sf.net/p/cwb/code/gui/cqpweb/branches/X.Y CQPweb`

When you first create it, the base CQPweb directory will contain several subdirectories, as follows:

adm Web-directory for the admin interface.

bin This directory contains scripts that can be run offline using command-line access to the machine that CQPweb runs on. See chapter 5.

css Web-directory for stylesheets and other files related to the appearance of CQPweb.

doc Web-directory for manual files.

exe Web-directory for the corpus-query interface.

jsc Web-directory for client-side JavaScript code.

lib This directory contains the actual CQPweb code. CQPweb never runs from this directory, but all the directories where it *does* run operate by calling the code found here.

rss Web-directory for the RSS feed, if enabled (see RSS-related configuration options in section 2.3).

usr Web-directory for the user account interface.

You should *never* rearrange the internal structure of these directories. It will break CQPweb if you do so.

Once you have downloaded/exported/checked out CQPweb, move its base directory into your web server's document tree. Note that the location you choose for the web script directory will determine the web address of your CQPweb installation relative to your webserver as a whole.

You may then need to adjust the ownership/permissions of the base directory and the files within it to make sure that the user account your webserver runs under has access to it. See section 1.9 for more information on this process.

1.5 Setting up Corpus Workbench

If you do not already have CWB on your system, you will need to install it before going further.

Instructions and links for installing the core CWB system can be found at `http://cwb.sourceforge.net/download.php`.

You should install the most recent available version of CWB from the 3.4.x series. Older versions do not work as well with CQPweb. As of CQPweb version 3.3.0, version 3.4.21 of the CWB core is required. But ideally use the most recent version that you can.

When installing CWB, you have assorted options, some of which affect the location of the executables once installed. The default location for the executables under Linux is `/usr/local/bin`, but it is possible to install them elsewhere. Wherever they are, note that it will be important for CQPweb to be able to find them. By default, CQPweb assumes these executables are on the PATH variable in the environment the web server runs on. If this is not the case, then you can tell CQPweb explicitly where to find the CWB executables using the `$path_to_cwb` variable in the configuration file (see 2.3).

1.6 Setting up the Perl modules

CQPweb **does not require** the CWB-Perl interface (although older versions used to). However, you may wish to have this set of tools available. For instance, optionally, you can configure the system to use the original Perl CEQL parser (rather than CQPweb's builtin CEQL parser).

Instructions and links for installation of CWB-Perl can be found at `http://cwb.sourceforge.net/download.php#perl`.

The package mostly relevant to CQPweb is the "base" *CWB* package, which includes *CWB-CEQL*. You may wish to install other packages from the CWB-Perl API (*CWB-CL*, *CWB-Web* and *CWB-CQi*) but these are of less direct utility with CQPweb.

If you wish to use the Perl CEQL parser, you need at least version 3.0.4 of the base *CWB* package.

Once you have downloaded the packages you want to install, you can find installation instructions in each package's README file.

- If you follow the standard instructions, the Perl module files will be copied into a directory on Perl's `@INC` path, such as `/usr/local/lib/perl/{$version}/` or `/usr/lib/perl5/{$version}/`, depending on the details of how Perl is set up on your system.
- Assuming you do install to this standard location, there is a known "gotcha" to be aware of: when you update your system software to use a newer version of Perl, any modules in directories whose path includes the old version number may no longer be detected. You will need either to reinstall the modules, or else to move the modules across from the directory with the old version number to the directory with the new version number.
- If for any reason you wish to set up CQPweb to use a different Perl binary to your system's default, you need to make sure the CWB modules are accessible to *that* Perl binary. If you don't understand the previous sentences, ignore the preceding paragraph and this one!
- If you install the Perl modules into a folder that is not in the normal Perl `@INC` path (which you can do by specifying an alternative `PREFIX`), then you will need to tell CQPweb where they are. You can do this by setting the option `$perl_extra_directories` in the configuration file (see 2.3).

1.7 Setting up R

There are no special considerations to note in relation to the R statistical software.

1.8 Setting up PHP

- CQPweb version 3.3 requires at least version 7.3 of PHP and preferably version 7.4.
- CQPweb version 3.4 will require PHP version 8.

PHP's internal library of functions is divided into *extensions*. To run CQPweb, your PHP installation needs the following extensions:

- The **gd** extension is needed if you want user-account creation to be protected by CAPTCHA.
- The **intl** extension is required
- The **json** extension is required
- The **mbstring** extension is required
- The **mysqli** extension is required (see further below)
- The **phar** extension is required
- The **zlib** extension is required

CQPweb normally uses the **zlib** extension for compressing and decompressing files. Other compression methods can be provided as options if the relevant PHP extensions are available, although CQPweb will run fine without them:

- The **zip** extension
- The **bz2** extension

Depending on your operating system, the extensions listed above may be installed by default along with PHP, or you may have to install them separately. On package-managed Linux systems, for instance, it is common for extensions such as **json** and **phar** to be made available as separate packages, even though they are provided by default with PHP when it is installed outside the package management system.

This also applies to PHP's Command Line Interface (CLI) program, that is, the **php** executable which runs PHP scripts from within a terminal. In package-managed Linux, this may be absent by default so that getting it requires a specific package to be installed (e.g. on Debian/Ubuntu, **php-cli**) even though the CLI program is usually a builtin feature of PHP.

CQPweb requires this CLI program; to check whether it is installed and accessible, run the following command:

- `php -v`

If the PHP CLI is installed and accessible, this command will print the program's version number and then exit.

The CLI can be used to check whether the extensions mentioned above are installed or not. For example, to check for **mbstring**, use this command:

- `php --ri mbstring`

The output of this command is:

- If the extension is installed: a line with the name of the extension (here, *mbstring*), followed by a block of configuration information (which you can ignore).
- If the extension is *not* installed: a one line message saying so; in this case, *Extension 'mbstring' not present.*

CQPweb crucially requires PHP's **mysqli** extension in order connect to the MySQL or MariaDB database. Nearly all versions of PHP are almost certain to include **mysqli**. However, on many Linux distributions (Debian-based or Fedora-based paticularly) you may need to select and install a specific package to get **mysqli**. On other kinds of OS, including possibly on Windows, the **mysqli** extension may be present but not enabled: in which case, it can be enabled by amending the `php.ini` file (discussed in detail below).

In the unlikely event that your version of PHP does not have **mysqli** *at all*, the only fix is to recompile PHP to add it. (It would probably be easier to reinstall PHP from a source that *does* include **mysqli**.)

If you are running CQPweb on the internet (i.e. not simply on a standalone computer), then CQPweb needs to be able to send email via PHP's `mail()` function. You can check whether this is working by running the following command:

- `php -r "mail('you@somewhere.net', 'it works...','Yes it works.');"`

(obviously using your real email address). If you get an email, then PHP can indeed use the `mail()` function. If not, you need to reconfigure your system to allow PHP to send email. It's beyond the scope of this manual to explain how to do this; the PHP manual has quite a lot of information (`http://php.net/mail`) and more can be found via web search.

Certain aspects of the behaviour of PHP are controlled by its `php.ini` file (see `http://php.net/configuration.file`, `http://php.net/ini`). Your system may have several files with this name, for different environments; you need to find the one used when PHP is run via your web server, as well as the one used when PHP is run as a command-line program (i.e. CLI). On Debian systems, for instance, the `php.ini` files for the Apache web server and for the CLI are respectively:

- `/etc/php/7.3/apache2/php.ini`
- `/etc/php/7.3/cli/php.ini`

If you don't know where to look, then on a Unix-like system, `/etc/php` is usually a good place to start. On Windows, `php.ini` can normally be found in the folder in which you installed PHP.

The CLI `php.ini` does not need to be adjusted for the settings that apply only to a web environment (e.g. `post_max_size`, see below). But there are some settings (e.g. `mysqli.allow_local_infile`, see below) which need to be adjusted in both the CLI and web server environments' `php.ini` files.

- One known "gotcha" on Apple Mac OS X is that the `php.ini` file may not exist; in this case, there should be a `php.ini.default` file which you can copy into the correct directory (under the name `php.ini`) and then amend if need be.

The precise settings in the `php.ini` file(s) may be different depending on how you installed PHP (or your system administrator may have subsequently adjusted them). Most of them do not affect CQPweb.

However, four directives set limits on PHP's use of system resources, and CQPweb has been written on the assumption that these directives are set to at least moderately generous values; if your system's settings are much less than these, you may have problems. The directives in question are as follows:

- `upload_max_filesize` needs to be quite high if you want to upload corpus files for indexing over HTTP; we recommend `80M` (for files larger than that relying on HTTP is probably a bad idea anyway)
- `post_max_size` needs to be at least as high as `upload_max_filesize`
- `memory_limit` should be generous as some CQPweb operations are RAM-intensive (e.g. building subcorpus definitions in memory); a reasonable starting point would be `128M` (but if the default in your system is higher than that, keep the higher value!), while bearing in mind that certain types of corpora - for instance, those with complex XML structure - may need more RAM for common operations, e.g. `512M`.
- `max_execution_time` should be generous as well; we suggest `60`

In addition, there is another directive that has the potential to stop many CQPweb functions from working. This is `mysqli.allow_local_infile`.

- When this directive is switched off (set to `"0"`), the **mysqli** extension cannot run commands of the form `LOAD DATA LOCAL INFILE...` .
- In PHP versions 7.0 to 7.2.15, and 7.3.0 to 7.3.2, the default was `"1"`, that is, to *allow* by default.
- In PHP versions PHP 7.2.16 onwards, and 7.3.3 onwards, the default became `"0"`, that is, to *disable* by default.
- It is *possible* to run CQPweb without using `LOAD DATA LOCAL INFILE...` by means of the configuration setting `$sql_local_infile_disabled` (see 1.12.2, 2.3.3.
- But it is usually a better idea to change `mysqli.allow_local_infile` to `"1"` in `php.ini`, as other problems may arise when `LOAD DATA LOCAL INFILE ...` is not used.

Once CQPweb is up and running, the values of all PHP settings can be checked by looking at "PHP configuration" in the Admin control panel. You can also find out the location of the active `php.ini` file from this screen.

If you are running a version of PHP with the Suhosin patch (which comes by default in some Linux distributions) then there is an additional "gotcha" to look out for. This is that a limit is placed on the length of individual values in an HTTP request - 512 bytes by default. This can result in CQPweb pages failing to work if this limit is not sufficient for certain parameters. If you have Suhosin and you want to be sure of not running into this problem, you need to add the following line to `php.ini`:

- `suhosin.get.max_value_length = 8000`

However, even with this limit increase, users may have trouble with very long queries, since different browsers may impose alternative, lower limits on HTTP request values.

1.9 Setting up disk locations

As well as the location of the web scripts themselves, CQPweb needs you to allocate *four other directories* for its use. These are used for the following purposes:

1. CWB corpus index files
2. CWB registry files
3. Temporary files (including the query cache)
4. Files uploaded into the system by users

All four of these directories should be *outside your webserver's document tree* so they are not exposed to the world. Once you have created them, you should leave them exclusively to CQPweb; no one else should add, amend or delete files in any of the four directories.

You will need to enter the paths to these directories in the configuration file: see section 1.13.

If you also use CWB and CQP from the command line on the same machine that is running CQPweb, then it's worth noting that the locations you choose for the CWB index data and registry *do not* need to be the same as the ones used normally by command-line CQP. CWB has, compiled into it, a default registry path, but CQPweb *does not use that directory.*

The username of the webserver process needs to have full read-write-execute access to all four directories. The username of the *mysqld* process also needs read and write access to the third and fourth (temporary/upload) directories if you want the SQL DB to use file-access functions, as described in 1.12.4.

The easiest way to accomplish this is to give read-write-execute permissions on these folders to "all", or - if you are worried about security - to "group" (where the file is assigned to some group that both the SQL DB server's account and the web server's account belong to).

How to work out the usernames of the server programs: For *mysqld*, the username is usually `mysql`. For the Apache webserver (process name something like *httpd* or *apache2*) it is usually something like `apache` or `www` or `www-data`. To find out for certain, run the following command (in this example, for *mysqld*):

- `ps -e -o user,comm | grep mysqld`

... and the first word on the output line will be the username you want.

1.10 Extra security on disk locations

A "gotcha" can occur when creating the directories discussed above in systems that have extra security software installed. These systems limit the areas of the filesystem that server programs like Apache or *mysqld* can access - blocking access to anything outside the designated areas, *even if* the server has all the necessary filesystem permissions.

The two systems of this type that are often encountered on Linux are **AppArmor** and **SELinux**.

1.10.1 AppArmor

AppArmor is widely used in the Ubuntu and Debian flavours of Linux. In fact, it may be automatically installed and enabled on these types of system.

AppArmor works by blocking file access to programs that it thinks ought not to be manipulating a particular directory, even if the user account the program is running with has all the proper permissions. If your system is configured to apply AppArmor restrictions to your SQL DB's daemon program, that is, *mysqld*, the result will be that the daemon will *only* be able to access files in the parts of the filesystem that AppArmor has explicitly let it access. This will often *not* include the CQPweb data directories you have created! And if AppArmor blocks *mysqld* from accessing to the CQPweb directories, you will not be able to run CQPweb.

Many versions of Debian and Ubuntu will configure AppArmor so that it restricts *mysqld* by default. However, recent versions of MariaDB block this action (as noted above, MariaDB's daemon is called *mysqld* just like MySQL's daemon) so if you are using MariaDB you may find that its AppArmor configuration file contains only comment lines.

If AppArmor *does* affect your *mysqld*, the solution is to add exceptions to AppArmor's configuration file for *mysqld*. In the systems we have seen this on, the file to edit is:

- `/etc/apparmor.d/usr.sbin.mysqld`

This file contains the "profile" AppArmor applies to the MySQL/MariaDB daemon. Its filename represents the path to the executable whose filesystem access is being restricted. You will normally need to be root to edit the AppArmor configuration files (`su` or `sudo`).

Before the closing brace in this file, add a line like the following for each CQPweb directory:

- `/path/to/the/directory/in/question/** rw,`

Then restart AppArmor with one of these two commands (for systems that use **SysV init** and **systemd** respectively):

- `sudo /etc/init.d/apparmor restart`
- `sudo systemctl restart apparmor.service`

If there is no AppArmor configuration file for *mysqld*, or if there is a file but it contains nothing or only comment lines starting in `#`, then AppArmor is not enabled for your *mysqld*; if you are experiencing problems nevertheless, they have some other cause.

The alternative approach to dealing with AppArmor is simply to disable it. Only do this if you do not need the extra security it supplies. To disable the *mysqld* profile:

- `sudo aa-disable /etc/apparmor.d/usr.sbin.mysqld`

To completely disable AppArmor, use the following commands (note, these commands work if you have a **systemd**-based Linux; other kinds of OS will have other ways to manage services):

- `sudo systemctl stop apparmor`
- `sudo systemctl disable apparmor`

To uninstall AppArmor (on Debian or Ubuntu; other Linux versions may be different):

- `sudo apt remove --purge apparmor`

1.10.2 SELinux

SELinux (https://selinuxproject.org/) works differently from AppArmor but can have the same effect of blocking a program's access to files and directories that the program's username has filesystem permissions to read and/or edit. If enabled, SELinux may block Apache (and possibly also MySQL/MariaDB) from accessing/modifying your CQPweb disk locations.

Rather than editing a configuration file, telling SELinux to allow a given program access to a given folder is done via a pair of command-line utilities: `semanage fcontext` and `restorecon`. Please consult the SELinux documentation for full information. However, the following commands should work to prevent Apache from being blocked in most circumstances:

- `semanage fcontext -a -t httpd_sys_rw_content_t "/path/to/directory(/.*)?"`
- `restorecon -R -v path/to/directory`

The first of the commands above adds a "file context" for the path specified which makes the directory, and files/subdirectories, available to Apache for read and write. The second reloads the context for that path so that your added file context will take effect.

1.11 Setting up your webserver

1.11.1 Overview

There are three things that you need to make sure are configured in your webserver:

- It *must* be configured so that files with the `.php` extension are run as PHP (whether via CGI or via a module like Apache's **mod_php**). This is the default on most webservers.
- It *must* be configured so that a file named `index.html` or `index.php` is served as the default when the address of a directory is accessed (so `http://my.server.net/directory` produces the same as `http://my.server.net/directory/index.php`). This is also usually the default situation.
- It *must* allow symbolic links in URLs. CQPweb corpora are addressed via URLs of the general form `http://my.server.net/CQPweb/corpus`, but the actual entry for `corpus` within the CQPweb web-directory is implemented as a symbolic link, not an actual directory.

Other steps that are not necessary, but that might be useful, include the following:

- Block HTTP access to the `bin` and `lib` subdirectories of the CQPweb base directory.
- Turn off the use of `.htaccess` files (Apache webserver only).
- Set up your webserver to use HTTPS instead of HTTP for CQPweb.

These are discussed in further detail below.

1.11.2 Using HTTPS

HTTP is the format used by web browsers and servers to talk to one another. HTTPS is a variant of HTTP where *all this communication back and forth is encrypted.* So, while a snooper on an HTTP link can see what the browser and server are saying to one another, when the link is HTTPS, that can't happen.

It is always preferable to run CQPweb on an HTTPS server:

- Some of your data may be confidential or restricted.
- You need to protect your users' private data (in some jurisdictions this may be a legal duty).
- Users tend to reuse their account credentials (usernames and passwords), even when instructed not to. A user's CQPweb account could be considered low-importance, but it is 100% possible that some users have reused a password that they also use for internet banking or their email account or other high-importance credentials. If you let users connect to CQPweb via HTTP, their password is sent to the server unencrypted and can easily be seen and stolen. Using HTTPS protects users from this danger.

(This doesn't apply to running CQPweb on a single computer, of course. HTTP is fine for that.)

As with all else, the precise details of how you configure a web server to use HTTPS rather than HTTP vary from webserver to webserver. You will need to look at the documentation of your webserver.

《*examples – Apache.*》 **TODO**

1.11.3 Specific webservers: Apache

Since Apache is the most commonly used webserver on Unix systems, we have accumulated more experience on installing CQPweb alongside Apache than any other server. Indeed, older versions of CQPweb actually relied on Apache for username/password checks (this was changed in version 3.1). The notes in this section outline some of the most common points of Apache configuration that are important for CQPweb.

Apache configuration is a rather complex topic, and cannot be dealt with in full here (see `https://httpd.apache.org/docs/`). In particular, note that it is impossible to say here specifically what Apache configuration files you need to edit to adjust how Apache treats CQPweb, since this can differ drastically from system to system; especially if you are on Linux, a lot depends on how your distro has decided to package Apache: see `http://wiki.apache.org/httpd/DistrosDefaultLayout`.

However, here is some general advice.

Apache's behaviour is controlled by various configuration directives. These directives can be given in the main configuration file, or they can be given in **.htaccess** files that may optionally be added to each directory in Apache's document tree.

If you change an Apache configuration file, you will need to restart the webserver process for the changes to have effect. This is not necessary if you are using **.htaccess** files.

In versions of CQPweb *earlier than 3.1*, **.htaccess** files were used to control user access to corpora. More recent versions do not use or reply on **.htaccess** files at all.

The only directories to which it is necessary to control access are the `bin` and `lib` subdirectories of the base CQPweb directory. This *can* be done with **.htaccess** files, but it is better done in one of the main Apache configuration files. That way, the directives are loaded only once (when Apache starts up), and not every time the server is accessed (which is what happens when **.htaccess** files are used, meaning multiple extra disk-reads are required).

The directive which turns on the use of **.htaccess** files is:

- `AllowOverride All`

Conversely, the directive that turns it off is:

- `AllowOverride None`

In Apache's configuration file, this directive may be given globally, or within a `<Directory>` directive. You can create a separate `<Directory>` block for the directory where CQPweb lives, or you can adjust the settings for a higher-level directory, if that will not interfere with other uses of the webserver.

Similarly, you need to make sure that Apache is set to follow symbolic links in URLs (for reasons explained in 1.11). This is enabled by default, but it's advisable to declare it explicitly (because otherwise you are dependent on this setting not being affected by changes elsewhere in the Apache configuration). This *must* be done within a `<Directory>` directive, or in a **.htaccess** file; the declaration is as follows:

- `Options FollowSymlinks`

(with other `Options` values added as required).

Putting all the above together, a typical `<Directory>` block for the CQPweb directory would be:

```
<Directory /path/to/cqpweb>
  AllowOverride None
  Options FollowSymlinks
</Directory>
<Directory /path/to/cqpweb/bin>
  deny from all
</Directory>
<Directory /path/to/cqpweb/lib>
  deny from all
</Directory>
```

Note that the "/path/to/cqpweb" that you need to give is an absolute path on your filesystem (it is *not* relative to the root of the web document tree).

If you cannot change the Apache configuration file, then to achieve the above, you'd need to use **.htaccess** files (assuming they are enabled).

First create two identical files (`bin/.htaccess` and `lib/.htaccess`), each containing just the following directive:

```
deny from all
```

Then add an **.htaccess** file in the CQPweb main web directory which switches on the `FollowSymlinks` option using an `Options` declaration line:

```
Options FollowSymlinks
```

There are, of course, many more things you can do with Apache to tweak how CQPweb is accessed. For example, you can add Apache-internal password authentication. However, if you do this, the Apache authentication will be *separate from and additional to* CQPweb's own system of usernames and passwords.

Finally: there are some known "gotchas" in Apache's behaviour under certain configurations. The best approach is not worry about the following until and unless the problems as described happen in your installation!

- Sometimes, Apache will happily serve up the builtin pages (e.g. the admin area) but then give you an "Internal Server Error" when you try to access the pages created during a corpus installation. This appears to be because these files are created with 0664 permissions (group-writeable, world-readable).
  - To fix the problem for an already indexed corpus, open a terminal to the directory containing its script files (`index.php`, `concordance.php`, etc.) and run `sudo chmod 644 *.php` .
  - To prevent the problem from recurring, edit the code file `lib/admin-install.inc.php` and change all instances of `chmod()` where the mode is set to 0664 to set it to 0644 instead.
  - *This problem should not occur in current versions of CQPweb, which use symlinks per corpus instead of code files per corpus.*
- The PATH environment variable (i.e. the list of locations where Apache will look for executables, if their precise location is not specified) may present a "gotcha". The PATH as seen by scripts running under Apache is *not necessarily the same* as the PATH that is available in the login-shell environment of the username Apache runs under. This is because Apache has its own internal system for setting environment variables, using its `SetEnv` directive and related functionality. If CQPweb is having trouble finding the CWB executables, or any other external program such as R, it may be because the PATH variable as seen from within scripts running under Apache does not contain their location. This problem is easily overcome by setting the variable `$path_to_cwb` in the CQPweb configuration file (see 2.3).

1.12 Setting up the SQL database

1.12.1 Creating the database

CQPweb uses an SQL database (MySQL/MariaDB) to store most of its ancillary data - that is, everything other than the actual CWB index data.

You will probably need "root" access to the SQL daemon in order to set it up. The following instructions are based on the assumption you are accessing the SQL DB via its command-line client program, but it is also possible to use your preferred graphical interface, of course.

First, you must create a new user and a new database for CQPweb to use. The name of the user and the database can be anything you like; for the sake of the example commands in this section we will assume that they are **cqpweb_db** and **cqpweb_user** respectively.

The required SQL commands are as follows:

- `create database cqpweb_db character set utf8mb4;`
- `create user cqpweb_user identified by 'cqpweb_password';`

Naturally, instead of "cqpweb_user", "cqpweb_password", etc. use the actual username and password that you want the user to have. This username/password combination will be stored in an insecure location, so make sure that you do **not** reuse either an account name or a password that you know to be used for any other purpose on your system.

You should *never* use your personal SQL username for CQPweb's SQL connection.

Having created the user, we must now give it *all* permissions over the database. If you want the SQL DB to use file-access functions, the new user also needs to be granted the *file* permission, which is set once and for all, rather than at the level of the database. File-access permission is not strictly necessary, but can help speed things up; it can also be useful if the `LOAD DATA LOCAL INFILE` command is disabled (see 1.12.2 and 1.12.4 below). Finally, the new user can optionally be granted the *process* permission, which is another once-and-for-all permission; if CQPweb has this permission, it is able to generate and display more reliable statistics about database file sizes. The commands are respectively:

- `grant all on cqpweb_db.* to cqpweb_user;`
- `grant file on *.* to cqpweb_user;`
- `grant process on *.* to cqpweb_user;`

Make a note of the username and password you have used, and of the database name; you will need them for configuration of your CQPweb installation. Also make a note of the server name needed to access the SQL daemon from PHP. This will probably be **localhost**, assuming that the SQL daemon is on the same machine as CQPweb itself (but see 1.12.3 below).

1.12.2 Known "gotchas" in SQL DB setup

- **Local infile permission**: MySQL/MariaDB can be configured to disable the `LOAD DATA LOCAL INFILE` command as a security measure – see `https://dev.mysql.com/doc/refman/8.0/en/load-data-local.html` . This configuration will stop CQPweb working (you will be able to tell this is happening because you will get the error message "ERROR 1148: The used command is not allowed with this MySQL version" when you attempt to set up the metadata for a corpus).

  (This is basically the same thing as the problem mentioned above with PHP's `mysqli.allow_local_infile` setting, but in this case, the ban on this command is imposed from the database daemon rather than from PHP's interface.)

  The preferred way to fix this problem is as follows:

  - Edit the MySQL/MariaDB configuration file (usually something like `/etc/my.cnf` or `/etc/mysql/my.cnf` depending on your operating system; some Linux variants have entire folders of configuration files, allowing you to add a file of tweaks rather than modify the existing configuration).
  - Find the line which deactivates the local infile feature.
  - It will be something like `local_infile=OFF`, `local-infile=0`, or `set-variable=local-infile=0`
  - (or, it might be absent altogether, in which case, you need to add that line)
  - Change the `0` to `1` (or `OFF` to `ON`)
  - Restart the SQL daemon.

  - **Note:** The local infile feature can be switched on/off separately for the daemon and for any connecting clients; so you may need to amend (or add) the setting as described above *twice*, in both the [**mysqld**] section of the configuration and the [**client**] section. If the problem persists after inserting `local_infile=ON`, the reason is probably that only *one* of the daemon and client settings has been affected, and you need to amend the other.

  Alternatively, the problem will be fixed if you set the CQPweb configuration variable `$sql_has_file_access` to **true**, because when you do this, the `LOAD DATA LOCAL INFILE` command is never used. But there are requirements that must be satisfied to use this option – see 1.12.4 below.

  If you can neither change the SQL DB's configuration, nor meet the requirements for `$sql_has_file_access`, there is yet a third way to solve the problem: set the CQPweb configuration variable `$sql_local_infile_disabled` to **true**. This makes CQPweb avoid the `LOAD DATA LOCAL INFILE` command unconditionally. Be warned - doing this should fix things, but is likely to be a major performance hit.

- **File privilege with restricted scope**: It is possible to limit the SQL daemon so that `INFILE/OUTFILE` commands only affect a specific area of the machine's filesystem. This is done via a setting called `secure_file_priv`.

  If `secure_file_priv` is set, then the daemon can only read/write files in the directory that it specifies. This can stop CQPweb from working. There are three ways to fix this problem.

  - Use the folder specified by `secure_file_priv` as your *temporary files* directory.
    * (See 1.9 on the temporary files location).
    * This solution will not be possible if your SQL daemon is not solely devoted to CQPweb.
  - Change the `secure_file_priv` option to specify the location of your *temporary files* directory.
    * Edit the MySQL/MariaDB configuration file (usually called `my.cnf`, see above).
    * Find the line which sets this variable (it will look like `secure_file_priv=/some/path` or `secure-file-priv=/some/path`) and change the path.
    * Restart the SQL daemon.
    * Again, this not practical unless your SQL daemon is solely devoted to CQPweb.
  - Switch off the `secure_file_priv` functionality entirely.
    * Find the right line of the MySQL/MariaDB configuration file, as per above.
    * Change the path to an empty string - so the line looks like this: `secure_file_priv=""`
    * Restart the SQL daemon.
    * This solution *does* work if your SQL daemon is not solely devoted to CQPweb. It is the **preferred and recommended** solution. However, it does have the disadvantage of removing a security measure.

  If you are using a shared server and you are not able to change the MySQL configuration, the only remaining option is to stop CQPweb from using the `INFILE/OUTFILE` commands by setting the CQPweb configuration variable `$sql_has_file_access` to **false**.

  A further "gotcha": prior to about 2015/2016, the default value for `secure_file_priv` was the empty string. But MySQL was changed so that this value now defaults to an OS-specific secure path. If you *have no* `secure_file_priv` in your `my.cnf` file, then things will work fine in older versions of MySQL (because the security feature defaults to being switched off) but in newer versions, CQPweb won't work. If you find that features of CQPweb such as collocations, distribution, and so on stop working after a MySQL upgrade, the problem is probably that the default value of `secure_file_priv` has changed. In this case, instead of *changing* the value of `secure_file_priv`, to fix the problem you need to *add* a line `secure_file_priv=""` to your `my.cnf` file to restore what was previously the default state.

It's unknown whether this issue affects MariaDB as well as MySQL.

- **PHP's connection to the SQL DB**: PHP does not differentiate MySQL and MariaDB; both are simply treated as "mysql". PHP needs to know how to connect to the SQL daemon via a *socket* in the filesystem. This information is often contained in the `php.ini` file, which contains settings that PHP will load when it starts up. On many systems, connecting to the SQL daemon simply "works" by default, but on some systems you may need to edit your `php.ini` file to tell PHP where to find the socket, by changing the `mysql.default_socket setting`. For instance, if your socket is at `/tmp/mysql.sock` but PHP is looking at `/var/mysql/mysql.sock`, you need to adjust `mysql.default_socket` to `/tmp/mysql.sock`. If you edit a `php.ini` file, make sure it is the one used when PHP is run by the server (whether as CGI or as a module of the webserver itself).

- **Database binary logging**: Binary logging is a MySQL/MariaDB feature that can be enabled or disabled, as explained here: `https://dev.mysql.com/doc/refman/8.0/en/binary-log.html`. If enabled, even light use of a CQPweb installation will make the SQL DB create very large binary log files, ultimately using up all your disk space over time. For this reason, it's recommended that you **disable** binary logging on the SQL daemon used by CQPweb. You can disable binary logging as explained here: `https://dev.mysql.com/doc/refman/8.0/en/replication-options-binary-log.html`. In brief, this is done by commenting out or deleting the line containing the `log_bin` command in the MySQL/MariaDB configuration file, or removing the equivalent `--log-bin[=base_name]` directive from the command line that starts up *mysqld*. (In either case you'll need to restart *mysqld*.)

1.12.3 Using a separate computer for the SQL DB

We normally assume that the SQL daemon runs on the same machine as the CQPweb system itself. But it does not have to.

You might want to use two separate machines for the CWB-based and SQL-based parts of CQPweb, for reasons of performance (for a big corpus and for queries with lots of results, both the SQL DB and CWB require lots of disk space, disk read/write bandwidth, and processing power).

In this case, CQPweb itself (the web scripts) should be on the same system as CWB, and the SQL daemon on a separate system. This affects how you configure CQPweb as follows:

- You will need to insert the correct hostname (or IP address) for your SQL daemon's machine into the configuration file for CQPweb, instead of the (normal) `localhost` value. See section 2.2.

- Any configuration variable that involves a path to a temporary-storage directory or to the location of a program (see 2.2) needs to refer to *the machine with CQPweb on it*, not the machine with the SQL daemon on it.

- The optional variable (see 2.3) `$sql_has_file_access` can only be set to **true** if the paths to the temporary-storage directories are the **same** on both systems (e.g. if they are mounted to the same location). The system cannot check this for you! This is explained in more detail in 1.12.4 below.

You will need to make sure that your SQL machine is configured to allow network traffic through the port that your SQL daemon is using. How this is done depends on the operating system and firewall software on that machine. Under Linux, you would use a utility such as `iptables` or the more modern `nftables` to modify the operation of the Linux firewall.

The SQL daemon *mysqld* usually listens on port 3306, but this can be changed in the MySQL/MariaDB configuration file: if in doubt, check! (The SQL command `SHOW VARIABLES WHERE Variable_name = 'port'` will tell you.)

- If the CQPweb system is the only user of the SQL machine, then for security it makes sense only to open up the port for traffic coming from the IP address of the main CQPweb machine.
- On Linux, it is possible for the firewall to redirect data arriving on other ports to port 3306. So in that case it would not be port 3306 that you would open.

An additional security measure you can implement when creating the CQPweb user in MySQL/MariaDB is to link that account to the particular IP address of the CQPweb machine. The format for this is as follows:

- `create user 'cqpweb_user'@'111.222.333.444' identified by 'cqpweb_password';`
- `grant all on cqpweb.* TO 'cqpweb_user'@'111.222.333.444';`

... instead of the form given above - with the correct IP instead of "111.222.333.444", of course. When you create the account in this way, only login attempts from the specified IP will be accepted.

1.12.4 The SQL daemon's file access

General note: The issues relating to the SQL daemon's file access have been discussed in multiple sections of this chapter, where relevant; this section contains an overview.

Many CQPweb operations involve transferring data into the SQL DB from files in CQPweb's directories (or, conversely, from the database to such files). Specifically, of the directories discussed in section 1.9, the temporary-files location and the uploaded-files location, including any subdirectories, are used in this way.

There are three ways that file data can be transferred between CQPweb and the SQL daemon. In descending order of speed, these are:

- The SQL daemon reads the file directly, using the SQL command `LOAD DATA INFILE`.
- PHP's `mysqli` client module transmits the file to the daemon, using the SQL command `LOAD DATA LOCAL INFILE`.
- CQPweb reads the file incrementally and passes its content to the daemon in small chunks. No `LOAD DATA` command is used.

By default, CQPweb is configured to use the *second* of these methods. This requires the creation of a temporary copy of the file for use by the SQL daemon, which is by definition slower than the server directly accessing the existing file. However, the difference in performance is not huge.

In order for the first method to work, the SQL daemon must be able to access the two directory locations mentioned above. This requires certain preconditions to be fulfilled.

If the daemon is running on the same computer as CQPweb, the preconditions for the SQL-daemon-file-access method are:

- the SQL daemon (or, to be precise, the OS user account it runs under) must actually have access to the two directories (see 1.9 for how to make sure of this).
- the SQL user account used by CQPweb must have `GRANT FILE ON *.*` permissions (see 1.12.1 for more detail on this point).

If the SQL daemon is running on a different computer (see 1.12.3), the preconditions for the SQL-daemon-file-access method are:

- the two directories in question must be accessible on the machine the SQL daemon runs on (e.g. by mounting them as remote drives).
- the two directories must be reachable using the same paths on the SQL daemon's machine as on the CQPweb machine (i.e. use the same path for the mount point, or use a symbolic link).
- the SQL daemon (or, to be precise, the OS user account it runs under) must actually have access to the two directories - as per above.
- the SQL user account used by CQPweb must have `GRANT FILE ON *.*` permissions - as per above.

To explain the first two points in a little more detail, let's say that, for example, the temporary-files directory is located at `/var/cqpweb/temp` on the main CQPweb machine. In order for the SQL daemon to be able to access files in this folder, first, the same underlying disk location must be accessible from the SQL machine: e.g. by it being a network drive mounted remotely by both machines, or by the CQPweb machine making its local directory available remotely, by SFTP for instance. Second, the directory must be accessible at the same path on both machines, either by mounting it at `/var/cqpweb/temp`, or, if it must be mounted elsewhere (e.g. `/mnt/cqpweb/temp`), then by creating `/var/cqpweb/temp` as a symbolic link to the mount point.

Once you are sure that the relevant preconditions are fulfilled, you can switch to the SQL-daemon-file-access method by setting the configuration variable `$sql_has_file_access` to **true** (see 1.13 and 2.3.3).

When this variable is **true**, the first, fastest method of file transfer is used.

Conversely, as noted above in 1.12.2, if the second, default method does not work due to the SQL daemon's `LOAD DATA LOCAL INFILE` command being disabled, and you cannot use the first method instead, you must fall back on the third, slowest method, by setting `$sql_local_infile_disabled` to **true**.

N.B.: see note in section 1.12.2 about a possible "gotcha", a MySQL/MariaDB feature that limits `GRANT FILE ON *.*` permissions to a single directory using the `secure_file_priv` option.

1.13 Creating a configuration file

Before going any further with installation, you must create a configuration file. This can be done manually or automatically.

The CQPweb Configuration File is described in its own chapter of this manual (2). As that chapter explains, there are a small number of *compulsory* configuration variables, and a much larger number of *optional* settings. To get CQPweb up and running, you need only create a configuration file with the nine compulsory settings - optional settings can be added later at your leisure.

There are two ways to do this:

- Manually working from a framework file - see section 2.5
- Using the automatic configuration script - see section 2.4

Either way, you should note that you will need to enter several of the settings that you created at earlier points in the install process:

- The paths to the four directories you created for CQPweb's data (see 1.9);
- The username, password, database name and server name for the SQL daemon;
- You will also need to enter at least one system-admin username.

1.14 Completing setup

With your configuration file created, you are ready to run the final steps of the setup process. These include:

- Creating the structure of the SQL database, and setting up default data;
- Identifying the best Unicode handling supported by your SQL daemon;
- Creating accounts for the admin usernames specified in the configuration file;

Although these can in theory be done manually it is much more effective to let the system do it for you. That is the purpose of the auto-setup script, which is called from the command line as follows:

- `php autosetup.php`

(note that you must be inside the `bin` subdirectory of the base CQPweb directory for this to work). For a general discussion of running command line scripts see section 5.1.

This script makes your CQPweb installation ready to use. It also, as noted above, actually creates accounts for the admin users you specified when creating your configuration file. It will ask you to specify passwords for these users *only at the point when it creates the accounts.* Passwords for CQPweb are not stored in the database - only an encrypted form is stored - and are never saved anywhere else on disk.

Once this script has run, you are ready to go. Open a web browser and navigate to:

- `http://your-server.net/path/to/cqpweb/web/directory/`

2 The CQPweb Configuration File

2.1 About the configuration file

This chapter describes the CQPweb *configuration file*.

The configuration file is a text file. It is always called `config.php` and is always placed in the `lib` subdirectory alongside the code files. It contains PHP code creating variables that are used by the rest of the system to control different aspects of how things work. It's important to understand that *none of the things that are set in the configuration file can be changed through the web interface*, even if you are logged on with an admin account. This is for security reasons.

Since the configuration file is a PHP file, you can in theory put any arbitrary code that you like in it - but it is very strongly recommended that you do not do anything other than assign a series of variables as specified here.

The variables can be in any order; though it is convenient to organise the file so that groups of settings that relate to the same part of the system are close to one another, it is not necessary to do so, and in fact the ordering of the variable assignments in the configuration file makes no difference at all to CQPweb.

PHP variable assignment is very simple and will be familiar to anyone who has done a bit of programming. All assignments are of the following form:

- `$variable_name = VALUE;`

In PHP, a file must have `<?php` on its first line so that its contents will be recognised as PHP code. If you use the framework file provided (see section 2.5) then this is already present.

There are two kinds of CQPweb configuration variable. First, there is a small group of variables that you *must* set - if you don't, CQPweb just won't work. These can be set up either manually or automatically (see section 2.4).

Then, there is a much longer list of variables that you *can* set, but if you don't, default values will be used. These should be added to the configuration file manually. These two types of variables, compulsory and optional, are listed in sections 2.2 and 2.3.

Every variable has a particular *type*, one of the normal types in the PHP language. Values of different types are specified in different ways. Again, these will be unsurprising to anyone who is familiar with any programming language:

Boolean A value which is either **true** or **false** (which may mean on/off, yes/no, and so on). Use the PHP keywords `true` and `false` to represent these values.

Integer number A whole number, entered as normal decimal digits (no spaces or thousands-separators).

Floating-point number A number with a fractional part, entered in decimal with a `.` for the decimal point.

String A short bit of text. Strings can be surrounded with either single quotation marks or double quotation marks. The difference is that if double quotation marks are used, a wider range of *escape sequences* are available to represent special characters. For the CQPweb configuration file, you are unlikely to need any escape sequences except those for the delimiting quotation marks themselves, which are `\'` and `\"` in a single-quoted string and in a double-quoted string respectively.

2.2 Compulsory configuration variables

Additional information on the variables containing location paths can be found in section 1.9 on "Setting up directories".

Additional information on the variables relating to the SQL database can be found in section 1.12 on setting up the SQL DB upon installation.

Variable name	**Description**
`$superuser_username`	**Type:** String This should contain the usernames of users who are to be system administrators, separated by the pipe `|` if there is more than one. For instance: `"anna|bert|craig"`. You can have as many admin accounts as you like, but you must have at least one. To add a new administrator, first create a normal account for the person (if they don't already have one), and then edit the configuration file to add their username to this variable. To remove an administrator, delete their username from this variable (but note you must never remove the *last* username!) No other actions are necessary.
`$sql_user`	**Type:** String The username of the SQL DB account that CQPweb should use to connect to the SQL daemon. It is usual to have a single dedicated SQL account for use by CQPweb.
`$sql_password`	**Type:** String The password of the `$sql_user` account on the SQL daemon.
`$sql_schema`	**Type:** String The name of the database on the SQL daemon that you created for use by CQPweb.
`$sql_server`	**Type:** String The address of the SQL daemon (either a hostname/IP address, optionally followed by a port number, or else a path to a local socket; see the MySQL/MariaDB documentation for more info on this). If your SQL daemon is on the same computer as the rest of CQPweb, this variable should usually be the string `"localhost"`.
`$cqpweb_tempdir`	**Type:** String Location of a directory which CQPweb can use to store its query cache and temporary data files.
`$cqpweb_uploaddir`	**Type:** String Location of a directory to use for CQPweb's "upload area" (storage for files uploaded by you or by users via the web interface).
`$cwb_datadir`	**Type:** String Location of a directory for CWB index data to be stored in.
`$cwb_registry`	**Type:** String Location of a directory that CQPweb can use as its CWB corpus registry. This does not have to be the same as your system's default CWB registry, but it can be if you want.

2.3 Optional configuration variables:

This is a reference guide to the optional configuration variables; many of them are also mentioned elsewhere in this manual.

Some optional configuration variables are not documented yet. This chapter will be expanded over time until it is as close as possible to 100% complete.

《*List of undocumented ones can be found in comments in the latex code at this point.*》 **TODO**

2.3.1 Locations of programs on the system

Variable name	Description
`$path_to_cwb`	**Type:** String **Default:** `""` Path of the directory containing the CWB executables (`cqp`, `cwb-encode`, and so on). The path can be absolute or relative; if relative, bear in mind that CQPweb always runs from one of the immediate daughter directories of its main directory. If the path contains any space characters, they must be escaped (with an escaped backslash, e.g. `".../Program\\ Files/..."`. If no path is given, it will be assumed the executables are in the system's usual path.
`$path_to_gnu`	**Type:** String **Default:** `""` Path of the directory containing the GNU (or equivalent Unix-y) utility programs, namely `tar`, `gzip`, and `awk`. These will almost always be on the normal path on a Unix system but may not be on Windows. The same general comments apply as to `$path_to_cwb`.
`$path_to_perl`	**Type:** String **Default:** `""` Path of the directory containing the Perl program; same general comments apply as to `$path_to_cwb`. Only needed if you wish to use the CEQL parser in CWB-Perl, rather than the internal CEQL parser.
`$path_to_python`	**Type:** String **Default:** `""` Path of the directory containing the Python executable; same general comments apply as to `$path_to_cwb`. (The interface to Python is an incomplete feature as of version 3.3.8.)
`$path_to_r`	**Type:** String **Default:** `""` Path of the directory containing the R executable; same general comments apply as to `$path_to_cwb`.

Variable name	Description
`$perl_extra_directories`	**Type:** String **Default:** `""` Extra directories to add to the INCLUDE path when running Perl (for the CEQL parser). This is only necessary if you have installed the CWB-Perl modules somewhere other than the default places where your Perl installation would normally look for modules. The string should contain one or more absolute or relative paths, separated by a pipe `|` if there are more than one.

《*in table above, in box on path-to-perl, add crossref to where CEQL is explained.*》 **TODO**

2.3.2 Web daemon features (Apache etc.))

Variable name	Description
`$web_daemon_user`	**Type:** String **Default:** `""` You can set this variable to the username of the account on your system that the WWW daemon runs under. This is usually something like `www` depending on the server software and operating system. For example, for Apache on Debian and Ubuntu Linux, it's `www-data`. When CQPweb knows this username, it helps in some aspects of corpus file management, but if this variable is left empty, it will not result in any major problems.
`$web_daemon_group`	**Type:** String **Default:** `""` You can set this variable to the name of the user group of the account that the WWW daemon runs under. As with `$web_daemon_user`, it isn't essential for you to supply this information, but it can be useful for some aspects of corpus file management.

2.3.3 SQL database features (MySQL / MariaDB)

Variable name	Description
`$sql_big_process_limit`	**Type:** Integer **Default:** `5` This variable places a limit on how many *big* SQL processes of a single type will be allowed to run at once. There are several types of "big" process (building collocation database, building frequency tables, building sort databases, and building categorised query tables) so more than the limit could run.

Variable name	Description
`$sql_utf8_set_required`	**Type:** Boolean **Default:** `true` Controls how characters are transmitted between the SQL daemon and CQPweb. The default is usually OK. If, however, some characters do not display properly in frequency list, keyword or collocation view, then setting this to **false** may fix things.
`$sql_has_file_access`	**Type:** Boolean **Default:** `false` This variable declares to CQPweb whether or not the SQL daemon has access to the filesystem on which CQPweb is running, and in particular two key working directories. It is explained in detail in section 1.12.4.
`$sql_local_infile_disabled`	**Type:** Boolean **Default:** `false` You should set this to **true** if your SQL daemon has been set up to disallow the `LOAD DATA LOCAL` command, and you can't change this setup. If possible, changing the SQL daemon configuration to allow `LOAD DATA LOCAL` is far preferable, it should be noted! See also section 1.12.2.

2.3.4 Memory, disk cache, and other hardware resource limits

Variable name	Description
`$cwb_max_ram_usage`	**Type:** Integer **Default:** `50` Some CWB programs allow a RAM usage limit to be set on their activities. When CQPweb calls these programs, it sets the RAM limit to the number of megabytes specified in this variable. The default is 50 megabytes. This variable applies only when CQPweb is run over the web; for the RAM limit that applies when CQPweb is run from the command line, see the next variable.
`$cwb_max_ram_usage_cli`	**Type:** String **Default:** `1000` Same as `$cwb_max_ram_usage`, but applies when CQPweb is run from the command line (see 5).

Variable name	Description
`$query_cache_size_limit`	**Type:** Integer **Default:** `6442450944` Controls the size of the query cache (the maximum size, in bytes, to which the temporary directory will be allowed to grow before old cached queries get deleted). Note that this only affects the size of the query cache; anything stored as an SQL table (such as temporary frequency tables or collocation databases) does not count towards this limit, and are controlled by separate variables also listed in this section. Until the cache limit is reached, the cache will just keep growing! Cached files are never deleted merely due to age, only when the disk space needs to be reused. The default value is 6 gigabytes.
`$db_cache_size_limit`	**Type:** Integer **Default:** `6442450944` Controls the size of the user-database cache (the maximum size, in bytes, to which the SQL table containing the user-database cache will be allowed to grow before old databases get deleted). This includes data for collocations, sorting, and distribution; categorised query data also counts towards the total but as it cannot be reconstructed, it will never age out of the cache. The default value is 6 gigabytes.
`$restriction_cache_size_limit`	**Type:** Integer **Default:** `6442450944` Controls the size of the restriction cache (the maximum size, in bytes, to which the SQL table containing the restriction cache will be allowed to grow before old cached restrictions get deleted). This counts only the part of the SQL database devoted to temporarily-stored restriction data, not any other stored data. Until the cache limit is reached, the cache will just keep growing! The default value is 6 gigabytes.
`$freqtable_cache_size_limit`	**Type:** Integer **Default:** `6442450944` This variable places a limit on how much disk space *ad hoc* frequency tables in SQL are allowed to take up. It is expressed as a number of bytes; the default is 6 gigabytes.

2.3.5 Configuring the user interface

Variable name	Description
`$default_per_page`	**Type:** Integer **Default:** `50` The number of results to show per page by default (in concordances, frequency lists, keyword lists, and so on). Most tools also allow the results-per-page rate to be altered on a per-query basis.

Variable name	**Description**
`$default_history_per_page`	**Type:** Integer **Default:** `100` The number of items to show per page in history-type displays (such as query history, saved queries, and so on). As a general rule you would normally want more of these per page than you would for `$default_per_page` - thus the difference in default values.
`$default_collocations_per_page`	**Type:** Integer **Default:** `100` The number of items to show per page in the collocation view.
`$dist_graph_img_path`	**Type:** String **Default:** `"../css/img/blue.bmp"` This string is the (relative) address of an image file that will be used for the bars in the bar chart mode of the Distribution display. The default is a file internal to CQPweb that creates plain blue bars.
`$dist_num_files_to_list`	**Type:** Integer **Default:** `100` The number of texts (or other items) to display in the Frequency Extremes mode of the Distribution display. The number of texts is limited because corpora can easily have thousands or tens of texts or other item identifiers, which would make the page hard to read and slow to load.
`$colloc_max_comparison_length`	**Type:** Integer **Default:** `40` The number of characters in each word or annotation form to compare when grouping forms for collocation. If this is set high, words and tags will be compared more rigorously, but more disk space will be needed for the collocation data. The default is usually fine.
`$sort_max_comparison_length`	**Type:** Integer **Default:** `40` The number of characters in each word or annotation form to compare when sorting a concordance, or grouping forms for a frequency breakdown. If this is set high, words and tags will be compared more rigorously, but more disk space will be needed for the sort data. The default is usually fine.
`$uploaded_file_bytes_to_show`	**Type:** Integer **Default:** `102400` When a file uploaded by the admin user (or, in future, by regular users) is displayed, only a certain amount of data is shown: the first section of the file, up to the number of bytes specified in this setting (to the nearest whole line). The default is to show 100 KB. A too-large setting here may cause browser overload, since so many of the files that CQPweb deals with are so very large.

Variable name	Description
`$hide_experimental_features`	**Type:** Boolean **Default:** `false` If set to **true**, certain new features deemed "experimental" will be hidden in the interface; users will neither see nor be able to use them. What features count as "experimental" will change from version to version.

2.3.6 Tweaking the look-and-feel

《*this should go in a section of its own somewhere. the table below should have refs to it*》 About colour schemes: **TODO**

The appearance CQPweb's interface draws on a small palette of about seven or eight colours. Different corpora can be given different onscreen colours as a hint to help users keep track of what corpus they are working with. The main menu, user account page, and admin control also use configurable colours.

The *colour scheme* for a corpus can be specified in three ways. Each is coded as a string in the configuration file (and database); more user-friendly ways to specify them are provided in the interface.

- By reference to a built-in colour scheme. This is specified with a string beginning in   followed by the name of the colour scheme in question.

  The available colour schemes are *blue*, *yellow*, *green*, *red*, *brown*, *purple*, *navy*, *lime*, *aqua*, *neon*, *dusk*, *gold*, *rose*, *teal*, *charcoal*, *motley*, *cinema*, *skin*, and *floral*.

  `~blue` is the default generic CQPweb colour scheme.

- By providing the relative or absolute URL of a CSS stylesheet which contains the colour definitions. Important note: stylesheets created for earlier versions of CQPweb (before 3.3) don't work with version 3.3.0 and higher. Any custom stylesheets you are using will need to be reworked to use the new system.

- By providing raw data specifying the colour definitions (in JSON format).

If you provide a CSS file it should look something like this:

```
:root {
  --colour-layout-fg      : colour-name-or-code ;
  --colour-layout-bg      : colour-name-or-code ;   /* optional, default = #d5d5d5 */
  --colour-layout-strong  : colour-name-or-code ;
  --colour-layout-contrast: colour-name-or-code ;
  --colour-layout-data2   : colour-name-or-code ;   /* optional, default = #f0f0f0 */
  --colour-layout-outline : colour-name-or-code ;   /* optional, default = white */
  --colour-text-normal    : colour-name-or-code ;
  --colour-text-faint     : colour-name-or-code ;
  --colour-text-bright    : colour-name-or-code ;
  --colour-tooltip-frame  : colour-name-or-code ;   /* optional, default = #003399; */
  --colour-tooltip-bg     : colour-name-or-code ;   /* optional, default = #e6ecff; */
  --colour-tooltip-text   : colour-name-or-code ;   /* optional, default = #000066 */
}
```

This CSS block creates a number of colour variables which will be embedded in the HTML header of each CQPweb page, for later reference by the presentation code.

A colour scheme specified as a JSON object should be a single object which maps variable names, equal to the parts of the names in the CSS code above after `--colour-` (i.e. `layout-fg`, `layout-bg`, `text-normal`, `text-faint` etc.), to the HTML colour names or numeric codes you wish to use.

Variable name	What it controls
layout-fg	Colour of foreground layout blocks (usually a pastel colour)
layout-bg	Colour of background layout blocks (usually pale grey)
layout-strong	Colour of emphasised layout blocks (table/column headings)
layout-contrast	Colour of layout blocks that contrast with the usual colours (e.g. warnings, errors)
layout-data2	Colour of secondary data area in concordance display (usually gray)
layout-outline	Colour of the bands of empty between layout blocks.
text-normal	Colour for most text
text-faint	Colour for de-emphasised text, e.g. the page footer, query speed info
text-bright	Colour for highlighted text (mostly links)
tooltip-frame	Colour of the the outer frame of floating tooltips
tooltip-bg	Colour of the background of floating tooltips
tooltip-text	Colour of the text in floating tooltips

Variable name	**Description**
`$colour_scheme_for_homepage`	**Type:** String **Default:** (see below) A colour scheme specification to use for the main menu page. By default, a lovely blue-and-grey table effect is used. If a relative URL is used for the homepage colour scheme, it must be relative *to the homepage*, which is one level higher (in the directory tree) than all the other URLs that CQPweb runs from. So if you want to address something in the CSS folder, for instance, you would need to start this variable with `css/`, rather than `../css`.
`$colour_scheme_for_adminpage`	**Type:** String **Default:** (see below) A colour scheme specification to use for the admin control panel. By default, a red-and-grey colour scheme is used. Same notes apply as to `$colour_scheme_for_homepage`.
`$colour_scheme_for_userpage`	**Type:** String **Default:** (see below) A colour scheme specification to use for the user-account homepage. By default, a green-and-grey colour scheme is used. Same notes apply as to `$colour_scheme_for_homepage`.
`$homepage_use_corpus_categories`	**Type:** Boolean **Default:** `false` If this is **true**, then the list of corpora on the main menu page will be given as a set of lists according to the corpus category, rather than as a single long list of corpora.

Variable name	Description
`$homepage_welcome_message`	**Type:** String **Default:** `"Welcome to CQPweb!"` A little bit of text (which can include HTML formatting) that will appear in the header box of the main menu page.
`$homepage_logo_left`	**Type:** String **Default:** `""` Settings for a logo to display on the left of the header box of the main page menu. This string can contain either (a) a single URL for an image to use as the logo; or (b) two URLs with a tab between them, in which case the logo image becomes a clickable link: the first URL will be used as the address of the image, and the second as the address for the link. URLs can be absolute, or relative to the location of the main menu page (i.e. the URL of your CQPweb base directory). A Corpus Workbench logo that you can use if you wish is located at the relative URL of `css/img/ocwb-logo.transparent.gif` . If you are running CQPweb on an HTTPS server, note that the logo URL you use *must* be on the same server (if not, most browsers will warn that the webpage is only partially secure). It is a good idea to add your image files to the `css/img/` subdirectory.
`$homepage_logo_right`	**Type:** String **Default:** `""` Same as `$homepage_logo_left`, but whatever you specify appears on the right side of the header box.
`$searchpage_corpus_name_suffix`	**Type:** String **Default:** `"<em>powered by CQPweb</em>"` A little bit of text (which can include HTML formatting) that is suffixed to the name of the corpus in the main search page header. If you don't want anything, set it to an empty string.

2.3.7 User account creation

CQPweb has the facility for users to create their own accounts through a standard "validate-by-email" mechanism. The configuration options in this section control whether this facility is available, as well as various aspects of how it works.

Variable name	Description
`$allow_account_self_registration`	**Type:** Boolean **Default:** `true` If this is true, a form will be exposed for anyone to sign u for an account on your server. If false, this form will not b available, and only admin users will be able to create new use accounts. (You can alternatively configure your web server t block or limit access to the signup form if you wish.)

Variable name	Description
`$allow_account_email_change`	**Type:** Boolean **Default:** `false` If this is true, users are allowed to change their account' email address. It is false by default because of the role th email address plays in granting access to corpora (see nex variable!). At present, user email addresses are **not** revalidated afte being changed by the user, since if the user types their emai address wrong, they can correct it via same interface; in futur versions of CQPweb, email revalidation may be added.
`$allow_account_email_change_group_persist`	**Type:** Boolean **Default:** `false` This variable only has any effect i `$allow_account_email_change` is **true**. By default, when a user changes the email address associate with their account, they are removed from any user grou that has an *Auto-add regex* (see 11.4). (because they ar no longer known to meet the criteria for membership in tha group). If this variable is set to **true**, this doesn't happen the user remains a member of every group they were part o before the change of email address. Regardless of the value of this setting, a user who change their email address will subsequently be added to any grou with an auto-add regex that is matched by their *new* emai address (i.e. just as new accounts are added to any grou whose regex their email matches). This can, of course, resul in the user being added back to a group that they were jus removed from by the previous step. In most cases, you would only set this variable to **true** if yo manage group membership manually rather than via auto add regex. Otherwise, this setting is best left set to **false**.
`$account_create_contact`	**Type:** String **Default:** `""` This variable only has any effect i `$allow_account_self_registration` is **false**. In thi case, you can supply a snippet of text here that will be adde to the web interface to tell prospective users who to contac to request an account. This string can contain HTML markup, fo example to add a "mailto" link, e.g. `"<a href=\"mailto:a.n.other@anytown.edu\">A. N. Other</a>"` .

Variable name	**Description**
`$account_create_captcha`	**Type:** Boolean **Default:** `true` Determines whether or not the account-creation form is pro tected by a CAPTCHA challenge. This can help protec your server against web robots, but you might want to dis able CAPTCHA for convenience if your account-creation pag is inaccessible to users on the open Internet anyway. I your PHP installation lacks the GD extension to PHP (see `http://php.net/gd`), then this setting is *always* **false**, re gardless of what you specify.
`$account_create_one_per_email`	**Type:** Boolean **Default:** `false` Determines whether or not multiple accounts can be createc that are linked to the same email address. By default, multi ple accounts with the same email address are allowed. To dis allow this, set this option to **true**. In that case, any attemp to create a second account with an email address already o the system will fail. If you change this setting, it does not apply retroactively: an accounts already in the system that share an email addres will **not** be affected.
`$blowfish_cost`	**Type:** Integer **Default:** `11` CQPweb uses Blowfish to encrypt passwords. The "cost controls how long it will take for this encryption to run. Th higher it is, the harder it is for an attacker to crack a give encrypted password. As of 2020, the default value of 11 i reasonable, but if you are worried about security you migh try 12 or 13. In the *System diagnostics* section of the admin interface, ther is a tool to stress-test password encryption, to see if you current Blowfish cost is high enough.
`$password_minimum_length`	**Type:** Integer **Default:** `7` Passwords shorter than this number of characters will be dis allowed. The default of 7 should be treated as an absolut minimum on an externally accessible server; it should reall be *much* more for security's sake. On a machine that is no open to the net, on the other hand, it can be set as low a you like, even 0.

Variable name	Description
`$create_password_function`	**Type:** String **Default:** `"password_insert_internal"` CQPweb uses "suggest password" functions to ease the cre ation of user accounts in the admin interface (see 11.2). Th default function (`password_insert_internal`) produces ran domised passwords of the form **aaaaddaaaa**, where **a** is lowercase letter and **d** is a digit. You can optionally create a password function of your own e.g. to get nicer, more wordlike passwords. This is somethin you should only attempt if you know how to write functions i PHP! The function you create should take a single argumen (an integer), and should return an array of strings, where eac string is a suggested password, and the number of strings i the array is equal to the integer argument. Then, set this variable to the name of the function you hav created (anything beginning "password_insert_..." is guaran teed not to conflict with any existing function) and add th function somewhere in the file `useracct-lib.php`. The bes place to add a function is right at the end of the file (tha way, if you are using a checked-out copy of the code from th CWB Subversion repository, your modification to the code i likely to be preserved when you update CQPweb).

2.3.8 User corpus system

Variable name	Description
`$user_corpora_enabled`	**Type:** Boolean **Default:** `false` If this is set to **true**, the system allowing users to upload and index their own corpora will be enabled. This means it will be visible to users. However, users won't be able to actually use it until they have been granted the necessary privileges (to upload files, to use CorpusInstaller plugins, and to use disk space in the upload area and in CQPweb's index data area).
`$colleaguate_system_enabled`	**Type:** Boolean **Default:** `false` If this is set to **true**, the *Colleaguate* system will be enabled. This system mimics a social network except that the purpose of forming links to colleagues is so that users can share out access to their corpora and subcorpora.
`$max_installer_processes`	**Type:** Integer **Default:** `1` User corpus installations run on a job queue to limit the load on the system. This variable controls how many installation processes can run at once. Jobs will wait to run until they are at the top of the queue and there is a free "slot".

Variable name	Description
`$installer_process_wait_secs`	**Type:** Integer **Default:** `3` This controls how long each user corpus installer position will wait after checking its position in the job queue before checking again. The wait time will be this number of seconds, multiplied by the number of jobs ahead in the queue.

2.3.9 RSS feed control

Variable name	Description
`$rss_feed_available`	**Type:** Boolean **Default:** `false` If this is set to **true**, then an RSS 2.0 feed of all the system-administration messages currently on the system will be available; its location will be the **rss** subdirectory of your CQPweb base directory. (If the RSS feed is activated, an icon linking to the feed will appear in the header bar of the the system-messages block.) The next three configuration variables allow you to define how you want the RSS feed to behave; they will be ignored if `$rss_feed_available` is not **true**.
`$rss_link`	**Type:** String **Default:** (A URL corresponding to the root directory of the CQPweb installation.) All RSS feeds must have a URL associated with them. By default the CQPweb RSS feed's link is simply the root directory of the installation, but you can configure it to be something else by setting this variable. Note that if you set it to anything other than a valid URL the resulting RSS feed probably won't parse.
`$rss_feed_title`	**Type:** String **Default:** `"CQPweb System Messages"` The title of the feed. You can set it to whatever you like, e.g. to mention your institution or organisation; it is useful to do so to disambiguate the RSS feed of one CQPweb server from another.
`$rss_description`	**Type:** String **Default:** `"Messages from the CQPweb server's administrator"` The basic description that will pop up in subscribers' feed readers. You can set this to anything you like, but it should not be longer than a short paragraph in order to be effective.

2.3.10 Error reporting

The variable `$debug_on` is the primary "switch" for whether debugging and error messages are printed out. The other `$debug_...` variables give finer control over what is printed, where, and for who.

Likewise, `$backtrace_on` is the main switch controlling whether a backtrace (call stack) is printed when CQPweb exits prematurely due to having encountered an error. The other `$backtrace...`

variables give finer control over what is printed, where, and for who.

Variable name	Description
`$debug_on`	**Type:** Boolean **Default:** `false` If true, debugging messages will be printed (normally, in the browser, but see `$debug_to_screen` below).
`$debug_for_all`	**Type:** Boolean **Default:** `false` If true, debugging messages will be printed for all users. Otherwise, only admin users will see these messages.
`$debug_sql`	**Type:** Boolean **Default:** `false` If true, all SQL queries, and the responses from running those queries, will be printed to the screen whenever `$debug_on` is true.
`$debug_cqp`	**Type:** Boolean **Default:** `false` If true, everything sent to/received from the CQP backend will be printed to the screen whenever `$debug_on` is true.
`$debug_html`	**Type:** Boolean **Default:** `true` If you set this to **false**, debug and error messages will be printed to the browser as text without HTML formatting. Plain text messages are always produced (a) when outputting a text-file download; (b) in scripts that run from the command-line.
`$debug_to_screen`	**Type:** Boolean **Default:** `true` If true, debugging messages are sent to the user's screen; that is, they are printed in the browser, or in the CLI when CQPweb has been invoked from the command-line.
`$debug_to_log`	**Type:** Boolean **Default:** `false` If true, debugging messages are sent to the server log. The server log will also receive messages that would have been printed to the browser whenever CQPweb emits such a message *after* the user's browser has been disconnected.
`$backtrace_on`	**Type:** Boolean **Default:** `false` If true, the exit message that is generated when CQPweb exits prematurely in an error condition will include a backtrace (that is, a printout of the call stack tracing the point at which the error occurred).
`$backtrace_for_all`	**Type:** Boolean **Default:** `false` If true, backtraces will be printed for all users. Otherwise, only admin users will see backtraces.

Variable name	Description
`$backtrace_compact`	**Type:** Boolean **Default:** `true` If true, a compact backtrace (one line per function call in the stack) is printed; this is usually preferable, as the alternate, fuller format is somewhat hard to read.
`$backtrace_to_screen`	**Type:** Boolean **Default:** `true` If true (the default), backtraces (when switched on) are sent to the user's screen; that is, they are printed in the browser, or in the CLI when CQPweb has been invoked from the command-line.
`$backtrace_to_log`	**Type:** Boolean **Default:** `false` If true, backtraces (when switched on) are sent to the server log.

The variables controlling where debug messages and backtraces are sent are **non-exclusive**. It is entirely possible, for instance, to have them printed to *both* server log and screen.

2.3.11 Miscellaneous configuration options

The configuration options grouped in this section have not yet been sorted into larger groups; in future they probably will be.

Variable name	Description
`$cqpweb_switched_off`	**Type:** Boolean **Default:** `false` If this is set to **true**, CQPweb appears **switched off** when accessed in a browser. Most parts of the program simply don't run in this case; when CQPweb detects that this setting is true, it simply prints a "switched off" message and exits without connecting to the SQL database or to CQP. The message shown can be found in the file `switched-off.php`, located in the CQPweb `lib` folder. This default message can be amended by editing `switched-off.php`; alternatively you can add a short blob of extra HTML in the companion variable `$cqpweb_switched_off_extra_message`. When CQPweb is switched off in this way, you can still run operations on the command-line. This setting is mainly of use to ensure nothing that users do can affect the database while upgrades, repairs, and so on are being run; as noted, the test to detect switched-off state runs *before* CQPweb makes an SQL DB connection. It would be possible to end up with an inconsistent database state if users were able to access the interface and "do things" while an upgrade is in process.

Variable name	Description
`$cqpweb_switched_off_extra_message`	**Type:** String **Default:** `""` You can set this to contain a little bit of text or HTML which will be included in the page presented to browsers that attempt to access CQPweb at when it is switched off.
`$cqpweb_root_url`	**Type:** String **Default:** `""` You can set this to an absolute URL (in the style `"http://server.net/path/to/cqpweb/directory"`) and this URL will be used internally when CQPweb redirects the user's browser from one point in the system to another. If you don't set it, then CQPweb will try to work out its own URL based on information in PHP's global `$_SERVER` array. That may produce incorrect results if you are running CQPweb in an environment where there is a lot of server proxying; if it does, things can be fixed by setting this variable appropriately. Otherwise, this can be safely ignored.
`$cqpweb_no_internet`	**Type:** Boolean **Default:** `false` If this is set to **true**, CQPweb assumes it is not connected to the open internet, and that it is only being accessed locally (i.e. by a web browser sending HTTP requests to **localhost**). This has two effects. First, CQPweb will disable all email-sending functions. Second, normal user account signup (which relies on email) will be turned off; that is, **true** here overrides `$allow_account_self_registration`, setting it to **false**. It is convenient to set this as **true** if you have installed CQPweb for personal use on a desktop/laptop computer.
`$cqpweb_email_from_address`	**Type:** String **Default:** `""` An email address, which if provided will be used as the "From:" and "Reply-To:" address for all emails sent by the system. If this is not provided, defaults will be provided by your system's email-sending program - CQPweb will not attempt to supply a default. Be aware, however, that some mail systems will block emails that lack a clear "From" address, as they are assumed to be spam - so it is wise to set this. The email you specify can take the form of either a bare address (*someone@somewhere.net*) or a named address (*A. N. Other <someone@somewhere.net>*). There are two basic possibilities here: (a) set this to a real email address, so any replies to system emails go to some monitored mailbox; or, (b) set this to an obviously fake address to (i) signal to users they should not reply and (ii) ensure that, if they *do*, the replies go into a black hole: an example value for the latter would be `"CQPweb Server <do-not-reply@server.net>"` or something along those lines.

Variable name	**Description**
`$server_admin_email_address`	**Type:** String **Default:** `""` An email address, which if provided will be exposed in the user interface to logged-in users as a designated means of contacting the server system administrator. If this is left as an empty string, all bits of text in the interface that would have specified a contact email address will simply be omitted. Email addresses on the web are often disguised (e.g. by spelling out "at" and "dot" instead of using the equivalent punctuation marks). It is not normally necessary for the email address in this variable to be disguised in this way, since only logged-on users can access the pages where it is displayed. However, you may wish to put an obscured email address here if you are running an open server. The email address is not converted into a link, so there are no restrictions on the format of the text. HTML is accepted and will be inserted into the user interface as you specify it.
`$cqpweb_cookie_name`	**Type:** String **Default:** `"CQPwebLogonToken"` Label by which login cookies will identify themselves to users' browsers. The default value is normally fine, but if you have more than one installation of CQPweb running from the same web domain, you may find that their login cookies get confused with one another unless you set this variable to something different in each installation. The main logon cookie will use this name directly; other cookies will append a suffix. It is best to use just word characters (letters, numbers, underscore) for this setting.
`$cqpweb_cookie_max_persist`	**Type:** Integer **Default:** `5184000` This is the maximum length of time, in seconds, that a user's login will persist if they do not visit the site (the clock is "reset" every time they do visit the site). The default value is 60 days. This does not affect the option given to the user to choose whether or not they stay logged in; if they choose *not* to stay logged in, their browser will delete the cookie anyway and the persistent login will never be reused.
`$cqpweb_running_on_windows`	**Type:** Boolean **Default:** (see below) You can set this variable to **true** to declare that the operating system is Windows, or to **false** to declare that it is (some flavour of) Unix. If this variable is not set, CQPweb will use PHP's internal settings to guess the OS - so the only reason to use this variable is if CQPweb guesses wrongly on your system. The fallback if guesswork fails is **false**. It is possible to run CQPweb on Windows 10 under the *Windows Subsystem for Linux*; WSL does not count as Windows as far as CQPweb is concerned, so the correct setting in that case is **false**.

Variable name	Description
`$use_unix_tools`	**Type:** Boolean **Default:** `false` For some procedures, CQPweb can either use an external Unix tool (`sort`, `head`, `awk`) or an equivalent internal process. If this variable is set to **false**, the internal process is used. If this variable is set to **true**, CQPweb uses the Unix tool. When this setting is **true**, you must ensure that the Unix tool executables are findable; this may require setting `$path_to_gnu` (see 2.3.1). It is possible to run CQPweb on Windows 10 under the *Windows Subsystem for Linux*; WSL does not count as Windows as far as CQPweb is concerned, so the correct setting in that case is **false**.
`$use_external_ceql_parser`	**Type:** Boolean **Default:** `false` CQPweb can use either of two *Common Elementary Query Language* (CEQL) parsers to translate Simple Queries into CQP syntax. The first is an internal parser, written in PHP. This is the default. The second is the external Perl CWB-CEQL module, which was the only parser used by earlier versions of CQPweb. If you set `$use_external_ceql_parser` to **true**, the Perl parser will be used. To do this, you must have the CWB-Perl module installed: see 1.6.

2.4 Using the auto-configuration script

One of the administrative tools supplied as part of CQPweb is an auto-configuration script.

The script, also discussed in section 5.3, is an interactive tool for creating a basic configuration file. When you run the script, it will ask you a series of questions. Each question sets one of the compulsory settings (see 2.2).

To use the auto-configuration script, open a command-line terminal and go to the base directory of your CQPweb installation. Then go into the `bin` directory, and enter the following:

- `php autoconfig.php`

... and follow the instructions to enter the paths and other information that you made a note of when setting up the various directories, databases, etc. to be used by CQPweb (as discussed in chapter 1). When you are asked to specify admin usernames, enter at least one username (normally, the one you personally will use).

Once you've answered all the questions, the script writes your answers to a configuration file in the correct location. If a configuration file already exists, the script will not overwrite it.

Once you've run this script, you can edit the resulting `config.php` file to add any optional configuration variables that you might want.

2.5 Using the configuration file framework

The file `framework-for-configuration-file.php` in the `lib` subdirectory of CQPweb is a blank framework file, which contains all the compulsory configuration variables. It also has a list of the optional configuration variables (with short explanations), to make the process of changing the optional variables less taxing. The easiest way to create a configuration file manually is using this framework file, as follows:

- Make a copy of the `framework-for-configuration-file.php` file in the `lib` directory
- Rename the copy to `config.php`
- Use a text editor to edit its contents.
  - For each compulsory variable, insert the correct value for your system.
  - Add any optional variables you wish to use (remove # to un-comment an option, then enter the desired setting).

2.6 Changes from earlier versions of CQPweb

The configuration file format described in this chapter is that of version 3.3 of CQPweb. The following sections summarise the changes made to previous versions. Some of these are *not* backwards-compatible and require you to adjust your configuration file when you upgrade.

Not listed here: the many new configuration variables which have been added over time; these have simply been added to the table of optional configuration variables in section 2.3.)

2.6.1 Changes in version 3.3

Version 3.3 made substantial changes to the previous format of the configuration file.

- The configuration file was renamed from `config.inc.php` to `config.php`. However, if `config.php` is not present, as a fallback CQPweb will still look for a file with the old name, so systems with a `config.inc.php` will continue to work (at least until version 3.4)
- Configuration variables beginning in `$mysql` were renamed to begin with `$sql` instead (since many systems now use MariaDB instead of MySQL).
- In addition, the database connection variables were renamed in the interests of using more standard terminology, as follows:
  - `$mysql_webuser` became `$sql_user`
  - `$mysql_webpass` became `$sql_password`
  - `$mysql_schema` became `$sql_schema` (i.e. only the standard change to the prefix)
  - `$mysql_server` became `$sql_host`

  The old names will continue to work until version 3.4.0 at least.
- The names of certain cache-limit variables were changed to make them more consistent.
  - `$mysql_freqtables_size_limit` was renamed `$freqtable_cache_size_limit` .
  - `$cache_size_limit` was renamed `$query_cache_size_limit` .

This change was initially made in v 3.2.12, but as of version 3.3, the old variable names are no longer respected.

- Configuration variables beginning in `$css_path` were renamed to begin with `$colour_scheme` instead; they can now contain any of the valid ways to specify a colour scheme 《*crossref to a section explaining the new colour scheme system to a section explaining the new colour scheme system*》. **TODO**
  - `$css_path_for_homepage` was renamed `$colour_scheme_for_homepage` .
  - `$css_path_for_adminpage` was renamed `$colour_scheme_for_adminpage` .
  - `$css_path_for_userpage` was renamed `$colour_scheme_for_userpage` .

  The old variable names beginning `css_path` will continue to work until version 3.4.0 at least.

- A new system for debug and error message printing was introduced (see 2.3.10 above). The old variables for debug messages were removed from the official list, though they will still work until version 3.4.0. The new variables offer finer control and are more systematically named. The old variables correspond to the new ones as follows:
  - Old `$print_debug_messages` corresponds to `$debug_on`
  - Old `$debug_messages_textonly` corresponds to `$debug_html` (but the values have opposite meanings: the old variable switches HTML formatting *off*, whereas the new variable switches it *on*)
  - Old `$all_users_see_backtrace` corresponds to `$backtrace_for_all`

2.6.2 Changes in version 3.2

Version 3.2 made only minor configuration file format changes.

In version 3.2.11 and 3.2.12, the names of certain cache limit variables were changed to make them more consistent. The old names were retained as synonyms during version 3.2, but removed as of version 3.3.0.

- `$mysql_freqtables_size_limit` was renamed `$freqtable_cache_size_limit` .
- `$cache_size_limit` was renamed `$query_cache_size_limit` .

In version 3.2.32, all configuration variables relating to Perl were made optional (as the CWB Perl modules are no longer essential for CWB).

2.6.3 Changes in version 3.1

Version 3.1 made changes from the configuration file format used in 3.0 and earlier. If you have a configuration file from an earlier version, here are the things you need to know about the changes that have taken place.

- The four compulsory variables that represent CQPweb's storage locations changed their format.
  - In 3.0, they were absolute paths, but with the initial `/` left off. This was a very non-standard way of specifying a filesystem path, and has been abandoned. In 3.1, these variables are treated as relative paths if they do not begin with `/`, and as absolute paths if they do.

  - This means that if, for instance, you had the following in 3.0:
    `$cqpweb_tempdir = 'var/cqpweb/temp';`
    then in order for things to continue to work, you must change it to the following in 3.1+:
    `$cqpweb_tempdir = '/var/cqpweb/temp';`
  - This affects `$cqpweb_tempdir`, `$cqpweb_uploaddir`, `$cqpweb_datadir`, and `$cqpweb_registry`.

- `$path_to_cwb` and `$path_to_perl` changed as follows:
  - They became optional; if they are not set, then the CWB and Perl executables will be sought in the system path as per usual for programs whose precise location is not specified.
  - Their form was changed in the same way as the storage location variables: previously, they were absolute paths with the initial slash missing, now they are absolute *or* relative paths.
  - This means that if, for instance, you had the following in 3.0:
    `$path_to_cwb = 'usr/local/bin';`
    then in order for things to continue to work, you must change it to the following in 3.1:
    `$path_to_cwb = '/usr/local/bin';`
    (alternatively, if `/usr/local/bin` is on the path for the web-server user, as it usually would be, there is no need to specify this variable at all).
- `$path_to_apache_utils` was removed.
- `$password_more_security` was removed.
- `$cqpweb_uses_apache` was removed.
- `$utf8_set_required` was renamed `$mysql_utf8_set_required`.
- `$cwb_extra_perl_directories` was renamed `$perl_extra_directories`.
- `$default_mysql_process_limit` was renamed `$mysql_big_process_limit`.
- `$cache_size_limit` was given a new default value: 6GB rather than 3GB.
- `$mysql_freqtables_size_limit` was given a new default value: 6GB rather than 3GB.
- `$use_corpus_categories_on_homepage` was renamed `$homepage_use_corpus_categories`.

3 The System Administrator's Interface

3.1 Introduction

The main user interface for system administrators is the *Admin Control Panel*. This contains controls for monitoring and managing many different aspects of CQPweb's behaviour, including indexing corpora, managing user access privileges, and tweaking the look-and-feel of the system.

The Control Panel can be accessed in three ways:

- From the **Admin control panel** link that appears in the side-menu for any corpus's main query screen *if* the logged-in user is an administrator.
- From the **Admin control panel** link on an administrator's user homepage.
- Via direct URL entry: add `/adm` to the base URL of your CQPweb system.

The Control Panel is laid out just like the main query screen and the user homepage: with a side-menu on the left and a main area displaying the selected function.

⟪*add image here*⟫ **TODO**

Different chapters of this manual explain different parts of what can be done with the control panel. The notation used throughout to refer to features of the control panel is as follows:

- **CP >XXX >YYY**

where "CP" means "go to the control panel", from where the link for option YYY should be clicked, and this link is found under menu heading XXX. If any further links must be clicked to access some feature, more > are given added.

A general overview of the Control Panel is given in section 3.2.

A secondary interface exists for functions that affect just a single corpus. These are not accessed via the Control Panel, but rather through the corpus query menu.

When a system administrator is logged in, an extra menu section appears in the side-menu of the main query screen, with the heading *Admin tools*. This appears immediately before the *About CQPweb* menu section. The various menu items under *Admin tools* are discussed in section 3.3.

3.2 The Admin Control Panel: Feature list

There follows a listing of all features available from the menu in the Admin Control Panel, with cross-references to the other parts of the manual where discussion of those features can be found. If there is no discussion elsewhere in the manual, a brief explanation is given here.

- Corpora
  - *Show corpora*: list of all installed corpora (except user corpora), together with the size in types, tokens, and texts, and the disk space each uses; plus a link to the delete function. There are disk use totals at the bottom of the screen.
  - *Show user corpora*: list of corpora installed by users (with the same information as in *Show corpora*)
  - *Install new corpus*: see 6.10

  - *View upload area*: list of uploaded files, with size/date info plus controls to view, compress/decompress, or delete a file. There is also a file upload tool to add new files. However, for very big files, it is better to use an FTP or SFTP client or the like rather than the web form.
  - *Manage corpus categories*: see 6.17

- Templates
  - *Annotation templates*: see 6.7
  - *XML templates*: see 6.9
  - *Metadata templates*: see 7.6
  - *Catdesc templates*: see 《*ref todo*》 **TODO**
  - *Visualisation templates*: see 《*ref todo*》 **TODO**

- Users and privileges
  - *Manage users*: see 11.2
  - *Manage groups*: see 11.4
  - *Manage group membership*: see 11.4
  - *Manage privileges*: see 11.5
  - *Manage user grants*: see 11.6
  - *Manage group grants*: see 11.6

- Plugins
  - *Manage plugins*: see 《*ref todo*》 **TODO**
  - *Show installer jobs*: list of currently queued and complete corpus installation jobs created by uses/

- Frontend interface
  - *System messages*: tool to add/remove messages from the list that appears on the homepage and on the standard query page.
  - *Embedded pages*: tool to add/remove standalone HTML pages within the system for use as corpus or annotation documentation (etc.): see 《*ref todo*》 **TODO**
  - *Mapping tables*: see 《*reference here*》 **TODO**

- Cache control
  - *Query cache*: see 4
  - *Database cache*: see 4
  - *Restriction cache*: see 4
  - *Subcorpus file cache*: see 4
  - *Frequency table cache*: see 4
  - *Temporary data*: see 4
  - *Fragmentation check*: see 4
  - *CWB file monitor*: see 4

- Backend system

  - *System settings*: list of system settings maintained internally by CQPweb (which, unlike the configuration variables discussed in 2, cannot be controlled by the user). These concern the collations available on the current SQL daemon, and the status of the CQPweb database.
  - *Manage SQL processes*: see ⟪*ref todo*⟫ **TODO**
  - *View an SQL table*: debugging tool, allows you to print out the contents of any of tables in CQPweb's SQL database.
  - *SQL configuration*: displays configuration settings on the SQL daemon that are of relevance to CQPweb.
  - *PHP configuration*: displays PHP configuration settings that are of relevance to CQPweb.
  - *PHP opcode cache*: tool to monitor and manipulate the PHP opcode cache (explained in the web interface itself)
  - *Public frequency lists*: see ⟪*ref todo*⟫ **TODO**
  - *System snapshots*: **under development, do not use**
  - *System diagnostics*: tools to help diagnose problems with CQPweb

- Usage statistics
  - *Corpus statistics*: ranking of corpora by the number of queries run on them.
  - *User statistics*: ranking of users by the number of queries they have run.
  - *Query statistics*: list of the most frequently-run queries (across all corpora).
  - *Clear history*: a control allowing you to clear the Query History, on which the usage statistics are based (thus resetting all stats).

- Exit
  - *Exit to CQPweb homepage*: a link to the main page.

⟪*Make some of the XREFS above more specific when those sections of the manual are finished*⟫ **TODO**

3.3 Corpus Admin Tools: Feature list

The first item on this menu is simply a link to the Admin Control Panel. The other items are discussed in order below.

3.3.1 Corpus settings

This menu item leads to a screen where you can configure a large set of miscellaneous options. Also to be found here is the form for adding or deleting corpus-level metadata (⟪*crossref*⟫). **TODO**

To modify any option, simply tweak its value in the interface then press the **Update** button.

The options are as follows:

- *Corpus title*: allows you to change the title you supplied when you indexed the corpus.
- *Language*: allows you to set the language of the corpus. Language is "undetermined" by default and can be set either at install time, or using this control.
- *Directionality*: allows you change the value (left-to-right/right-to-left) that you supplied when you indexed the corpus.

- *Collation mode for word-type comparison*: see 《*XREF to a part of the manual discussing collation & case-sensitivity*》 **TODO**
- *Colour scheme*: allows you to change the colours used in the corpus interface.
- *Amount of context shown in concordance*: you can specify the width of the concordance either in words, or relative to any XML element.
- *Initial/Maximum words in extended context*: you can specify more/less words here based on, for instance, whether or not your users have the necessary permissions to see extended excerpts of the corpus data.
- *Word annotation for alternative view*: this is explained in section 10.2.
- *Hide empty values in alternative view*: this is explained in section 10.2.
- *Corpus category*: see section 6.17.
- *Visibility of the corpus*: *visible* corpora are listed on the homepage; *invisible* corpora aren't. However, an invisible corpus **is** listed on the "Corpus permissions" page of any user who has been granted the privilege to use that corpus.
- *External URL*: if you specify a link here, it will be rendered in the left-hand menu as a link under the **Corpus info** heading. The idea is that you would provide here a link to some online documentation about the corpus, if any such exists, to allow users to go straight to the relevant information. You can use an embedded page (see 《*XREF*》). **TODO**
- *Primary text categorisation*: this is explained in the chapter on metadata, specifically section 《*XREF*》. **TODO**

Below these options are the corpus metadata controls; see 《*crossref*》. **TODO**

3.3.2 Manage access

This menu item presents some information relating to the access that users have to the corpus.

The corpus-access privilege system is explained in section 11.5.1. Privileges cannot be manipulated from here - this must be done via the Admin Control Panel (links are provided to the appropriate screens in the CP).

However, here you can find two things that are not always easy to spot within the full list of privileges:

- A list of all privileges that affect access to the the corpus, together with a list of the groups and individual users to which those privileges are granted.
- A full combined list of all users with access to the corpus, whether they have it from a membership in one or more groups, or as individuals; duplicates are removed (that is, if Group A and Group B both have access to the corpus, and User X is a member of both A and B, you will see User X listed here only once).

3.3.3 Manage text metadata

《*section not written yet*》 **TODO**

3.3.4 Manage text categories

⟪*section not written yet*⟫ **TODO**

3.3.5 Manage corpus XML

⟪*section not written yet*⟫ **TODO**

3.3.6 Manage annotation

⟪*XREF to section in the chapter on linking CEQL to annotation, or XREF from that to here?*⟫ **TODO**

3.3.7 Manage parallel alignment

The controls found in this section are explained in chapter 8; see especially 8.5.

3.3.8 Manage frequency lists

⟪*section not written yet*⟫ **TODO**

3.3.9 Manage visualisations

The controls found in this section are explained in chapter 10.

3.3.10 Add corpus data

⟪*section not written yet*⟫ **TODO**

3.3.11 Corpus setup notes

When a corpus is first installed, a record is made of the output of different tools involved in the installation process. This includes commands sent to, and messages produced by, the CWB command-line corpus management tools. This information in retained in case something works incorrectly in the installed corpus. If that happens, the setup notes can provide hints as to what it is.

The setup notes can be viewed immeidately after corpus installation in the CP, or later via this menu item. They are not formatted in any particular way - just a sequence of lines of input/output, presented in the order they were collected.

3.3.12 Cached queries

⟪*section not written yet*⟫ **TODO**

3.3.13 Cached databases

⟪*section not written yet*⟫ **TODO**

3.3.14 Cached frequency lists

⟪*section not written yet*⟫ **TODO**

4 Managing the CQPweb data cache

4.1 Introduction

CQPweb is built around a strategy of extremely aggressive caching of dynamically generated data.

The core CWB system does not employ caching. In general, any part of the corpus data is only stored *once*, in the CWB index; whenever a query is run, or data is requested in any other format, it is retrieved anew from the CWB indexes - even if the same data has been requested recently. (That said, CWB benefits from the fact that modern operating systems, and especially the Unix systems that it is primarily designed for, cache disk content in RAM automatically.) CWB indexes are designed to be maximally compact on disk, even if this complicates or slows down retrieval.

By contrast, CQPweb is designed with the assumptions that (a) the same requests will often be made many times in a row, and (b) speed of response to requests is the most important thing (far more important than minimising the use of disk space). These assumptions are rooted in its origin as a teaching tool: a common use-pattern for CQPweb is a roomful of students, working on the same data, and all doing pretty much the same kinds of queries. In this situation, caching all generated data (queries, ad hoc frequency lists, and more) leads to a substantial performance improvement, as resource-intensive processes only run *once*. For every user other than the first to run a particular process, response time is much faster; the dynamically generated data does not need to be rebuilt from the underlying corpus indexes, it merely needs to be retrieved from the cache.

This chapter explains the different kinds of data that CQPweb caches, and discusses aspects of cache administration that apply especially to large, multi-user installations.

4.2 Some background on the SQL system

CQPweb uses an SQL database (MySQL/MariaDB) as its secondary datastore. The CWB indexes contain the main corpus data, but all ancillary data (corpus and text metadata, frequency lists, analysis data for collocation/distribution/etc., cached queries and analyses, user data like categorised queries, as well as CQPweb's system management data) is stored in the SQL database.

To understand the ways this database - especially its caches - can be configured, some background on these SQL database systems is needed. Beware! This is a very incomplete account of what is actually going on in the SQL database management system, focused on the points relevant to CQPweb. For the full details, see the online manuals for MySQL and MariaDB, the two SQL systems that CQPweb can use.

- Originally, CQPweb relied on MySQL. It could not be used with any other SQL database system.
- MariaDB is a "fork" of MySQL: that is, a direct descendent of an earlier version of MySQL.
- Thus, MariaDB is similar enough to MySQL that CQPweb can use either one without problem.
- MariaDB shares so much ancestry with MySQL that its programs, data directories, etc. are usually labelled "mysql".
- As of 2019, the main difference between them that matters for CQPweb is that MySQL has a more extensive range of Unicode collations for `utf8mb4`-encoded text.
- Thus, unless otherwise specified, please assume that any comment in this manual on *MySQL* applies equally to MariaDB.

MySQL is a client-server database management program: a server or "daemon" called `mysqld` does the actual work of creating, modifying and searching the databases; the daemon is always accessed via a client program which interacts with the user, issues requests to the daemon, and handles the responses sent back by the daemon. CQPweb itself acts as a client to the MySQL daemon, but some actions involved in administering CQPweb involve accessing the daemon via the MySQL command-line client (called `mysql`). An example is the process of creating CQPweb's MySQL database in the first place, as described in an earlier chapter.

Most of the time, a database can be treated as an abstract entity which we control without worrying about the details of what the daemon is actually doing. However, when considering the performance issues that arise on large, heavily used installations of CQPweb (and similar programs) it *is* necessary to dig in to what the MySQL daemon is doing behind the scenes.

The MySQL system stores all the actual data using one of two "engines".[1] The older of these is **MyISAM**; the newer is **InnoDB** (or, in some versions of MariaDB, InnoDB's near relative **XtraDB**). Each separate table within a database can be set to use a different engine. The databases and tables are ultimately represented by various disk files (though while the SQL daemon is running, there is not necessarily a guarantee that all important information has actually been written to disk at any given moment!), as follows:

- The MySQL daemon has a dedicated directory on disk for its data (its location is set in the daemon's configuration options; the default location varies across operating systems, but on many Linux systems it is something like `/var/lib/mysql`).
- For each database that is created, a folder is created with the same name as the database, directly within that dedicated directory. So, if we create a database called **cqpwebdb**, its folder would be `/var/lib/mysql/cqpwebdb` (with the first part, again, depending on the OS).
- For each table within the database, a `.frm` file is created, which contains the table definition, but not the actual data.

The location of the actual table data depends on the engine; furthermore, InnoDB can locate its data in two different places depending on whether a mode called *file-per-table* is switched on or not.

- **MyISAM** stores actual table data in the same place as the `.frm` file, in two extra files with the same name, but different file extensions (`.MYD` and `.MYI`). So, for instance, if the table **corpus_info** uses the MyISAM engine, the database directory will contain `corpus_info.frm`, `corpus_info.MYD` and `corpus_info.MYI`.
- **InnoDB** with file-per-table mode switched **off** stores all the data for all InnoDB tables from all the different databases in a single location called the "global tablespace". This takes the form of a single file called `ibdata1`, which will be found directly under MySQL's main data directory. The `ibdata1` file will therefore be very large!
- **InnoDB** with file-per-table mode switched **on** stores the data for each table in a separate `.ibd` file, which is placed (by default) alongside the `.frm` file - although it is possible for a separate location for the `.ibd` file to be specified when the table is originally created. With this setup, there will still be an `ibdata1` file but it will be much smaller as it will not contain actual table data.

MyISAM was the default in older versions of MySQL; later, InnoDB became the default (in both MySQL and MariaDB). Initially, InnoDB's default setting was to have file-per-table switched off. In

[1] There are actually a whole lot of engines, but MyISAM and InnoDB are the only two that matter in practice.

recent versions of MySQL/MariaDB, file-per-table mode is on by default. It is generally agreed that InnoDB with file-per-table switched on is the best way to store table data in most cases, if you are using an up-to-date version of MySQL/MariaDB.

CQPweb is, in general, designed to work with the default settings of MySQL/MariaDB. What that means is that originally it was built with the assumption that most tables would be MyISAM; later it was modified to force most tables to be in InnoDB, once InnoDB became preferred. It did not, however, make any difference to CQPweb whether file-per-table was on or off until version 3.2.14, when it became possible (as described below) to store different types of data in different locations. This new functionality depends on file-per-table mode being switched *on*.

To find out what the default engine is on your system, enter the following command in the SQL client:

- `show engines;`

This will display a list of available engines with additional information. The key column is "Support": in this column, the word `DEFAULT` will appear next to one of MyISAM or InnoDB.

To find out whether your SQL daemon has the InnoDB file-per-table mode switched on, enter the following command:

- `show variables like "innodb_file_per_table";`

However, it's important to note that these settings only represent the present situation. If your CQPweb server has been running for a while, especially if it's been running for over one year, it's quite possible that these settings could have been different in the past. In that case, any data that was setup under the old settings will not have been updated to the new settings.

- If your CQPweb server used to run on a version of MySQL where MyISAM was the default engine, any old tables created as MyISAM will still be MyISAM.
- On the InnoDB side, if file-per-table mode is switched on but used to be switched off, any tables created in the global table space will still be located there.

In either case, your database will contain tables running on a hodgepodge of engines and table spaces.

An administration script has been provided that will fix these issues[2]: see 5.7 on `force-innodb.php`.

[2] There is an important thing to remember if you formerly used InnoDB without file-per-table, and have switched to InnoDB with file-per-table. This is that forcing all the existing tables out of the `ibdata1` file and into their own files *will not result in the `ibdata1` file getting any smaller* - since it *never* gets smaller.

So, for instance, if you have a 100GB `ibdata1` file, then switch on file-per-table and force the tables out into their own file, the result will be a 100GB `ibdata1` file *as well as* 100GB of separate table files.

The *only* way to reclaim disk space from the `ibdata1` file is to export all your SQL databases using the `mysqldump` program; drop all the databases from the SQL daemon; delete the `ibdata1` file; and then re-insert the databases from the exported file.

This is, it should go without saying, a somewhat dangerous process. It's also outside the scope of this manual, so no more will be said about it here - but see the following links for some discussion:

- Relevant pages in the MySQL manual (v5.7, other versions linked from there)
  - http://dev.mysql.com/doc/refman/5.7/en/using-innodb-tables.html
  - http://dev.mysql.com/doc/refman/5.7/en/innodb-multiple-tablespaces.html
  - http://dev.mysql.com/doc/refman/5.7/en/innodb-configuration.html
  - http://dev.mysql.com/doc/refman/5.7/en/mysqldump-sql-format.html
  - http://dev.mysql.com/doc/refman/5.7/en/reloading-sql-format-dumps.html

4.3 Explaining the different types of cached data

Let's consider the data cached by CQPweb under broad categories

《*explanations of the different caches*》 **TODO**

《*explanation of what is cached in corpus setup*》 **TODO**

《*on the restriction cache: maybe add note - more RAM may e needed if there are restrictions based on XML elements.*》 **TODO**

4.4 Disk locations for stored data

《*discussion of how the locations of these caches affects performance - multiple disks etc. Advising on partitioning etc*》 **TODO**

《*the config settings to use for each location*》 **TODO**

《*how do we move one of the caches once it's already in a place?*》 **TODO**

It's important to note that setting the location of one of the SQL DB caches will only work **if** you have InnoDB's file-per-table mode switched **on**; see section 4.2.

`https://dev.mysql.com/doc/refman/5.7/en/tablespace-placing.html`

4.5 Moving the cache location on an existing CQPweb server

The process for moving a cache folder is different for caches involving files and caches involving SQL tables.

`https://blogs.oracle.com/mysqlinnodb/entry/choose_the_location_of_your`

4.6 Optimising the SQL DB for cache performance

《*this might get hived off into a seaprate chapter on optimisation since it's a bit of a shift of topic for this one*》 **TODO**

《*this section currently just contains rough and badly typed notes*》 **TODO**

Lots of useful stuff here: `http://dev.mysql.com/doc/refman/5.7/en/converting-tables-to-innodb.html`

Cover here things like this:

innodb_flush_method=O_DIRECT innodb_log_file_size=1G innodb_buffer_pool_size=4G

flush method – setting this to O_DIRECT measn that read/write access is direc tot disk without suing the OS's disk cache in RAM. This apparently improves performance if the SQL DB has lots of RAM to cahce disk stuff in, (in which case the OS cache will be small and twatty.)

It is named after the Linux flag that it causes mysqld to use wiht innodb files. Here is a quote from man open(2): O_DIRECT (Since Linux 2.4.10) Try to minimize cache effects of the I/O to and from

- Discussions on StackOverflow and StackExchange/DBA:
  - `http://stackoverflow.com/a/4056261`
  - `http://dba.stackexchange.com/a/24963` - on the general process
  - `http://dba.stackexchange.com/a/2227` - on exporting the databases

(all links correct as of April 2016).

this file. In general this will degrade performance, but it is useful in special situations, such as when applications do their own caching. File I/O is done directly to/from user-space buffers.

buffer_pool_size - this is InnoDB's interenal cache of stuff from disk. The default is 128MB. The manual says that on a deidcated box, you'd set it to 80% of the avaialble RAM. The percona performance blog says that ideally you'd have as much RAM in the buffer pool as the DB takes on disk. Obvs in the case of CQPweb that's a no go. But it can certainly be more than 128MB. Given the RAM we have, I gave it 2GB for now

innodb_log_file_size – the MySQL manual says it can b e up to 1/2 of the buffer pool.

THERE SEEM TO BWE TWO CHOICES....... 1 Don't use O_DIRECT, use a small buffer tool - InnoDB does not do RAM caching of disk stuff, we rely on OS caching. 2 Use O_DIRECT and a mdoerately hefty buffer - InnoDB will cache some of the disk stuff in its buffer pool and we bypass the OS cache.

But note this helps READS more than it helps WRITES I think!

Discuss relationship of this to OS RAM caching of the CWB index files.

SEE: `http://dev.mysql.com/doc/refman/5.7/en/disk-issues.html`

《*end of rough and badly typed notes*》 **TODO**

4.7 User-data cache sizes

《*recommendations on disk space to allolcate: e4xplain the defauylt size allocation of 6GB per cache, and point out that a single-user installation might well want to drop this.*》 **TODO**

4.8 Finding and fixing cache leaks

A *cache leak* is the term we use for any situation where something goes wrong in the creation or management of cached data or temporary data such that CQPweb can no longer control the data and delete it where appropriate.

This can happen, for instance, if CQPweb is interrupted halfway through running. For instance, if CQPweb aborts halfway through a cache cleanup, it is possible that the record of some block of cached data might be deleted but the block of data itself retained (or vice versa).

In either case, the data becomes "invisible" to CQPweb's normal cache-management procedures, but will still be present, not being used but taking up disk space. This can have one of two negative consequences:

- Leaked data blocks can fill up your disk space if they have completely escaped monitoring.
- Conversely, if the leaked data blocks are still counted towards the size of your cache, they constitute a fraction of the cache that can never be used by actual productive data.

The very nature of "leaked" cache data or cache records, with imperfect or incomplete tracking, means that it would not be safe for CQPweb to attempt to clean them up manually - there would be a risk of data loss. However, tools are provided to assist you in cleaning up cache leaks.

To monitor and manually delete leaked items, go to one of the five functions at **CP >Cache control**. There is one for each type of cached data. Each is slightly different depending on the nature of the cache it covers. Each of these functions also gives you some information about the size of the cache and how full it is. Once a CQPweb server has been running for a while, most of the caches will become about 100% full and stay at that level. This is the expected behaviour - it means you are getting the maximum benefit from the disk space allocated to these caches.

The cache control functions often take quite a long time to display. This is because generating information about possible leaks invovles searching the whole of the cache. So don't worry about this.

5 Administering CQPweb from the command line

5.1 Introduction

Several actions that you can take as a system administrator are more conveniently undertaken outside the web interface, or else cannot be exposed in the web interface for reasons of security. These actions are instead available from the command line using scripts in CQPweb's `bin` directory (one of the subdirectories from the base CQPweb directory).

All the scripts in this folder are straightforward CLI scripts written in PHP. Some are interactive (that is, they will require user interaction as they run); others will simply run on their own and then finish.

Some of these scripts run within a complete CQPweb environment; that is, they read in the configuration file, create a connection to the SQL daemon, and so on. For that reason, they all need to be run from within one of the subdirectories of the base CQPweb directory. If the script is intended to operate on some specific corpus, then you should change your working directory to that corpus' directory, and call the script as follows:

- `php ../bin/name-of-script.php`

Alternatively, for scripts that do not refer to a particular corpus, your working directory should be the `bin` directory itself. The script is then called as follows:

- `php name-of-script.php`

Scripts which write files (e.g. `autoconfig.php`, `load-pre-3.1-privileges.php`) need to have write access to the CQPweb directory. You have three choices:

- Run them as the username under which the webserver runs (i.e. the user that owns the CQPweb directory);
- Run them as some other username, making sure that this user has write access to the CQPweb directory;
- Run them as root, changing ownership of the resulting files to the webserver user afterwards.

In the remainder of this chapter, the details of each of the scripts are explained.

5.2 The main *cqpweb* script

This is an executable that allows you to call any function from CQPweb's internal function library. If you wish to call a function that relies on being run in the environment of a specific corpus, run the script from that corpus' folder; otherwise run it from `bin`.

The syntax is as follows:

- `./cqpweb NAME_OF_FUNCTION ARG1 ARG2 ...`

It is difficult to provide any general comments since this script can do almost anything. Here are some examples of useful calls.

- `./cqpweb add_corpus_to_privilege_scope PRIVILEGE-INTEGER-ID CORPUS-HANDLE`
- `./cqpweb remove_corpus_from_privilege_scope PRIVILEGE-INTEGER-ID CORPUS-HANDLE`
- `./cqpweb create_corpus_default_privileges CORPUS-HANDLE`
- `./cqpweb add_new_privilege 1 "" "Permission to use at restricted level (initially has scope over no corpora, they can be added later)"`
- `./cqpweb add_new_privilege 2 "" "Permission to use at normal level "`
- `./cqpweb add_new_privilege 3 "" "Permission to use at full level "`
- `./cqpweb add_new_privilege 4 5000000 "Permission to create freq lists up to 500K tokens"`
- `./cqpweb grant_privilege_to_user USERNAME PRIVILEGE-INTEGER-ID`
- `./cqpweb grant_privilege_to_group GROUP-NAME PRIVILEGE-INTEGER-ID`
- `./cqpweb remove_grant_from_user USERNAME PRIVILEGE-INTEGER-ID`
- `./cqpweb remove_grant_from_group GROUP-NAME PRIVILEGE-INTEGER-ID`
- `./cqpweb update_corpus_visualisation_gloss CORPUS-HANDLE 1-OR-0-FOR-SHOW-IN-CONCORDANCE 1-OR-0-FOR-SHOW-IN-CONTEXT P-ATTRIBUTE-HANDLE`
- `./cqpweb update_corpus_visualisation_translate CORPUS-HANDLE 1-OR-0-FOR-SHOW-IN-CONCORDANCE 1-OR-0-FOR-SHOW-IN-CONTEXT S-ATTRIBUTE-HANDLE`
- `./cqpweb add_variable_corpus_metadata CORPUS-HANDLE ATTRIBUTE-DESCRIPTION VALUE-CONTENT`
- `./cqpweb update_corpus_title CORPUS-HANDLE "new title goes here"`

5.3 autoconfig.php

Creates a configuration file with just the essential configuration variables.

This script is interactive, that is, it asks you questions and requires you to type in the configuration values that you want to appear in the finished file.

This script is also discussed in section 2.4.

Note that prior to version 3.1, this script *also* performed a variety of miscellaneous set-up actions. These are now performed by `autosetup.php`.

5.4 autosetup.php

This script is used in the process of setting up a new CQPweb installation. It is described in section 1.14.

Like `autoconfig.php`, it is interactive.

5.5 cli-lib.php

This is not a script, it is a library file used by the other command-line scripts. It is mentioned here simply so that you know not to try and run it! (Nothing bad will happen if you do, in fact nothing *at all* will happen if you do.)

5.6 execute-cli.php

This script allows you to call any function from CQPweb's internal function library. If you wish to call a function that relies on being run in the environment of a specific corpus, run the script from that corpus' folder; otherwise run it from `bin`.

This is the backend behind the `cqpweb` executable (see 5.2). It is used in exactly the same way.

The syntax is as follows:

- `php execute-cli.php NAME_OF_FUNCTION ARG1 ARG2 ...`

It is difficult to provide any general comments since this script can do almost anything. Needless to say, you *really* need to know exactly what you are doing before you start messing with this script. Caveat emptor.

5.7 force-innodb.php

This script forces all tables in CQPweb's SQL database to use the InnoDB engine, instead of the older MyISAM engine. For an explanation of these engines, see 4.2.

The script operates by running the following command on each table:

- `alter table` *`name_of_table`* `ENGINE=InnoDB`

The effect of this command is discussed in the MySQL manual here `https://dev.mysql.com/doc/refman/5.7/en/alter-table.html` and here `http://dev.mysql.com/doc/refman/5.7/en/converting-tables-to-innodb.html` .

Run this script in the following situations:

- Your CQPweb installation is an old one, from before CQPweb began enforcing the use of InnoDB, and contains MyISAM tables.
- You have switched your MySQL server from using the InnoDB global tablespace to using file-per-table mode and want to force your existing tables out into their own files.

Certain tables require support for fulltext indexes, which was added to InnoDB only in 2013. If you are using an old version of MySQL whose InnoDB engine lacks this feature, any tables that need it will not be touched by this script; they will be "held back" as MyISAM.

5.8 install-corpus.php

This script installs a new corpus. It has an internal manual explaining how to call it, which you can see by running:

- `php install-corpus --help | less`

(for "`less`" substitute *Your Favourite Pager*)

Fundamentally, each of the options to thes script replicates one of the form elements on the relevant page in the Admin Control Panel.

5.9 load-pre-3.1-groups.php

Prior to version 3.1 of CQPweb, information about what user groups exist, and what users belong to each, were stored in Apache `.htgroup` files (and if you were not using Apache, you were out of luck). In version 3.1, this was changed so that groups are instead stored in the database, and Apache is no longer necessary: CQPweb can manage its own user accounts while running on any webserver.

This script migrates groups automatically from the old system to the new system. You should only need to run it once, immediately after you upgrade the code and database to version 3.1 (see section 16.3).

This script will ask you to specify interactively the location of the group file. This is the directory which will have been set as `$cqpweb_accessdir` in your pre-3.1 configuration file.

5.10 load-pre-3.1-privileges.php

Prior to version 3.1 of CQPweb, corpus access privileges were stored in `.htaccess` files in the web directory of each corpus. In version 3.1, this was changed so that privileges and grants are stored in the database, a much more flexible system. However, this means that lists of permissions in an existing v3.0 installation would be lost. This script retrieves them.

The script creates the three default privileges for each corpus on the system, if they do not exist already (access levels normal, restricted and full).

The "normal" access level is intended to be equivalent to the one-and-only access level available in older versions of CQPweb. For this reason, every group which *previously* had *any* access rights for a given corpus is assigned "normal" access to that corpus by this script.

It should be run *after* `load-pre-3.1-groups.php`.

5.11 load-pre-3.2-corpsettings.php

The use of this script is explained in 16.6.

It should be run *after* upgrading the database as far as v3.2.0.

If the version you are upgrading to is higher than 3.2.0, you need to run the database upgrade script a second time after running this script.

5.12 offline-freqlists.php

This script should be called as follows:

- `php offline-freqlists.php lowercase_name_of_corpus`

The script performs all frequency-list setup functions, i.e.e those actions usually performed on the "Manage metadata" page of the corpus interface, after indexing a corpus and installing its text metadata table (《*XREF*》). **TODO**

For very big corpora (hundreds of millions of words or more) this process can take a long time, hours or even a day or more. In that case it is not convenient to run it from a web browser, and the command-line version may make more sense.

This script prints a long stream of debug messages to indicate its progress. These can be safely ignored until and unless something goes wrong. This is not an interactive script and no user attention is required.

5.13 upgrade-database.php

From time to time the CQPweb database format is changed. This script should be run after you update your code to a new version, and it will implement any necessary database version changes.

This is an interactive program - it will sometimes demand that you acknowledge some alert about something that has changed by pressing Enter before it will continue.

Note that if you upgrade the code, but do not run this script to bring the database into line, CQPweb will break (possibly very badly).

The biggest set of upgrades are those between version 3.0.16 and 3.1.0. However, any database upgrade can potentially take a long time.

If you are running an online server (as opposed to one on a standalone computer!) then it is recommended to take the server offline while you do the upgrade - so that users cannot access your server while it is in a half-and-half state (which, depending on what they do, might cause data integrity problems). The easiest way to do this is simply to disable your web sever software temporarily.

See also section 16 for more information on upgrading.

6 Indexing corpora

6.1 Quick checklist

This chapter is not complete. Until it is, users who are not familiar with the corpus indexing process may benefit from using this checklist (courtesy of Philipp Heinrich on the CWB Developers Mailing list).

To create a new corpus, use the following tools in turn:

- In the Admin Control Panel:
  - install new corpus
- Inside the new corpus's UI:
  - manage text metadata
  - (build CQPweb frequency lists using either "manage frequency lists" or the offline script)
  - manage corpus XML
  - manage visualisations
  - manage annotation - setup CEQL bindings
  - check corpus settings
- In the Admin Control Panel:
  - manage privileges
  - manage user / group grants

6.2 Basic concepts

CQPweb is based, naturally, on CWB corpus indexes. Setting up a corpus for use in CQPweb essentially means indexing a CWB corpus. However, the design of CQPweb means that it has some specific requirements for the layout of its corpora beyond the basics of CWB. This section is intended to explain CQPweb corpora to help you prepare appropriate data for input.

A corpus in CQPweb can be characterised in terms of the combination of its *texts*, its *annotations*, its *XML*, and its *metadata*.

- *Texts* are the fundamental organisational unit of CQPweb corpora. They are one type of XML element, but one that must always be present. Every corpus consists of a sequence of one or more texts. Texts cannot overlap, or be contained within other texts. All the words in a corpus must be contained in one of that corpus's texts.
- *Annotations* are the different layers of word-level information. This corresponds to the more general CWB notion of a p-attribute (positional attribute).
- A corpus' *XML* corresponds to the more general CWB notion of an s-attribute (structural attribute).
- *Metadata* is a set of structured information about some aspect of a corpus, typically its texts but potentially other kinds of XML as well. Since metadata is a major topic it will be dealt with in a separate chapter (7).

When indexing a corpus, it is necessary to specify the corpus' design in terms of annotation, XML and metadata, and to make sure that the corpus data to be indexed contains appropriate text tags. Annotation, metadata and XML can be specified from scratch, or their design can be taken from a predesigned *template*.

6.3 The notion of a handle

A *handle* is a short text label used to refer to any of the following entities in CQPweb:

- A corpus.
- An ID code for a text or an XML region.
- An annotation layer.
- A type of XML.
- A field in a metadata table.
- User account usernames.
- A user-specified save-name.

This notion of handle may be familiar as it is fundamentally similar to the labels used for corpora and (p- and s-) attributes in CWB/CQP. However, CQPweb enforces certain requirements for what is and is not a valid handle.

- Each type of handle is limited to a certain, short length (see below)
- Handles can only include certain characters: ASCII letters, ASCII digits, and the underscore character (i.e. the same rules as for "words" in regular expressions, or for identifiers in C and related programming languages).
  - Note that this is different to command-line CWB, which allows the hyphen to be part of an attribute or corpus name in addition to the characters mentioned above.
- Some types of handle (for corpora, annotation/p-attributes, and XML/s-attributes) cannot include any uppercase letters.

It is necessary to limit the length of handles because of features of the MySQL/MariaDB database system. When CQPweb installs a corpus, it creates a number of database tables - and handles associated with the corpus are used to generate many table/column names. Other handles are used as database keys (and some database keys in CQPweb include up to three handles).

However, MySQL/MariaDB has two important built in limits: (1) Table and field names cannot be longer than 64 characters. (2) Database key fields cannot be longer than 333 letters. CQPweb has to limit the length of handles to fit with these constraints. For this reason, there are four broad categories of handle within CQPweb:

- Handles which can be up to 20 characters in length (because they are used as part of a longer table name). This limit is applied to:
  - Corpus handles.
  - Annotation handles (p-attributes).

- Handles which can be up to 64 characters in length (because they may form the name of a table column, but will not be embedded in a longer table name). This limit is applied to:
  - Handles for XML elements or attributes (s-attributes).
  - Handles for metadata field names (any type of metadata).
  - Usernames.
- Handles which can be up to 200 characters in length (because they will never be used as part of a table name, but might be used as part of a longer database key). This limit is applied to:
  - Handles for the categories in classification-scheme metadata.
  - The save-name of a saved query or subcorpus (but see note below about categorised queries).
- Handles which can be up to 255 characters in length (because they might be used as a database key on their own). This limit is applied to:
  - ID codes for texts and for XML elements.

Categorised queries, although they are a type of saved query, cannot be given names as long as those of a normal saved query (200). This is because, when a categorised query is separated out to create a saved query from each of the categories of concordance line, the new name contains the name of the categorised query combined with the name of the category - so the total length of *both combined* can be no more than 200.

In versions of CQPweb prior to v3.2.0, usernames were limited to 30 characters; they can now be up to 64 characters, as noted above (avoiding the need for a fourth kind of handle).

6.4 File format for corpus data input

CQPweb input data should be in CWB vertical file format (or *VRT*). This format is described in detail in the *CWB Corpus Encoding Tutorial* (available at http://cwb.sourceforge.net/documentation.php).

These are the essential points:

- The format can be described as a hybrid of TSV (for token data) and XML (for structural data).
- One token per line
- Annotations (word-level tagging) are presented in columns
- Every token must have a value for every column
- Columns left empty are treated as containing an "undefined" value.
- The
-
-
-
-
-
-

《*Input file: do by XREF to other docs if possible - the CWB encoding tutorial frinstance.*》 **TODO**

6.5 Linking handles and descriptions

《*Explain that wherever there is a handle, it can become a description.*》 **TODO**

6.6 Annotation

《*primary annotation - do we introduce this here, or do we leave it for the section on CEQL?*》 **TODO**

6.7 Annotation templates

An **annotation template** is a data structure that describes a set of annotations (that is, p-attributes, corresponding to columns in the input data).

If you wish to index multiple corpora that have the same annotations, then rather than specify the details of each annotation every time, you can create and save a template that stores those details and can then be invoked when a corpus is created.

For instance, one very common pattern of annotations is for the second column of the input file to contain a part-of-speech tag, while the third contains a lemma (recall that the *first* column always contains the wordform of the token). This three-column format is produced by, among others, the **TreeTagger** software.

Rather than enter `pos` and `lemma` over and over again when setting up corpora, you can create a template description which describes this three-column format. When you set up a corpus of this kind, specifying its structure is then as simple as picking your prespecfied template.

(In fact, certain useful templates, including the three-column *word, pos, lemma* format, are actually built into CQPweb for you.

《*NOTE: explain template creation here by saying the form is the same as the one for corpus setup. BUT don't go into the forms for annotation here. that comes under "indexing-proc" below.*》 **TODO**

6.8 XML

(Include here the discussion of "text" as the compulsory element and XREF to "text" as discussed in "basic concepts".)

6.9 XML templates

6.10 The indexing process

(all about the form)

(inc use of the non-template forms)

6.11 Using a pre-indexed corpus

《*Turn following rough notes into real manual content*》 **TODO**

`Step 1 - index the corpus using command-line CWB (wherever you like on the system,`as long as the files/directories you create are in a location on the file system where the web server's user account has permission to read them)

Step 2 - go to the "Install new corpus" page in CQPweb, and click on the link at the top that says "Click here to install a corpus you have already indexed in CWB."

Step 3 - specify the location of the registry file. (this will be copied into CQPweb's own registry if not already there; the index files themselves will not be copied or moved.)

Step 4 - once you've installed the corpus thusly, proceed onto the other installation steps (generate your text metadata from the XML attributes on ¡text¿, or else install a metadata file; setup frequency lists; etc.)

6.12 The metadata setup process

《*short explanantion here only – direct to the next chapter for the full coverage of metadata*》 **TODO**

6.13 Building frequency lists

Frequency list setup is dependent on the existence of the text metadata table (see (《*XREF*》). **TODO**

Do automatically small corpora

Do bit by bit in web interface

Do offline with script

6.14 Linking annotation to CEQL syntax notation

《*explain this!*》 **TODO**

6.15 Setting up corpus access rights

When a corpus is initally set up, no users except the system administrator(s) will be able to access it. To enable access, you need to (a) create a *privilege* that covers use of the corpus, and then (b) grant that privilege to one or more users or groups.

(xref to "user accts" chapter...)

6.16 Further corpus configuration

There are many settings that can be configured for each corpus after it has been indexed. They are available via the individual corpus's interface (*not* via the Admin Control Panel). When you are logged on with an administrator account, an additional section will appear in the left-hand-side menu, labelled **Admin tools**.

Some of the options on this menu have been discussed already in this chapter, as they are involved in the setup procedure. Others are discussed elsewhere: for the **Manage visualisations** option, for instance, see chapter 10. For a full overview, see section 3.3.

6.17 Putting corpora into categories

An optional step in setting up a new corpus is putting it into a specified *category*.

Currently, what category a corpus is in affects only one thing: the layout of the list of corpora on the homepage. If category-based ordering of the homepage is switched on, then the list will be divided into sections based on the corpus category feature. What category a corpus is in will determine where it appears in the list.

See section 2.3.6 for how to set the right configuration variable to switch on category-based organisation of the home page.

Categories are created and managed in the Admin control panel: **CP >Corpora >Manage corpus categories**. The upper part of this screen shows the existing categories. The most important feature of a category is its sort order, which is represented as an integer, and which determines where it appears on the homepage. The sort order is a relative number - that is, it doesn't matter exactly what the value is, it matters what it is relative to the other categories. The [**Move up**] and [**Move down**] controls can be used to adjust the sort order. Categories can be added using the separate control on the lower part of the screen.

In the **Corpus settings** screen for any individual corpus you will find an option to change the category of that corpus. When they are first indexed, all corpora are placed in the default "Uncategorised" category, and will stay there until you move them.

Within each category, corpora are listed in alphabetical order of the corpus handle - note this is not necessarily the same as the descriptive name, which is what is actually displayed!

Finally note that if a corpus is set to be *invisible*, it doesn't matter at all what category it is in.

7 Metadata

7.1 Introduction

《*(introductory explanation of what it is, how it is stored, where it appears in the interface)*》 **TODO**

In CQPweb, *metadata* is a covering term for data about an indexed corpus, about the the texts in a corpus, or (sometimes) about individual instances of XML elements.

The underlying CWB system does not provide an easy way to store and manipulate metadata. For that reason, CQPweb uses its SQL database to store and manipulate all metadata.

Metadata, especially text/XML metadata, is crucial to several core CQPweb functions:

- *Distribution.* 《*EXPLAIN*》 **TODO**
- *Restricted queries.* 《*EXPLAIN*》 **TODO**
- *Subcorpus creation.* 《*EXPLAIN*》 **TODO**

This chapter explains the different levels of metadata (corpus/text/XML), the different datatypes that are available, and how they work; how metadata is installed for a corpus; and the use of metadata templates.

7.2 Corpus metadata

......

(explain corpus metadata, where it is displayed, how to add it)

....

7.3 Text metadata

.....

Each corpus stores text metadata in a dedicated *metadata table.*

A metadata table is a database table very much like a table you might create in a spreadsheet. It has a series of columns, where each column represents a particular *field* or *attribute* - that is, a particular bit of metadata. So, for instance, you might have columns for the *author*, *title* and *genre* of each text.

The rows represent the items described by the metadata table. The first column always contains the unique identifier for the items. So, in a text metadata table, the first column contains the text ID code. In CQPweb, text IDs and other identifier codes are always handles (see 《*XREF*》). **TODO**

Metadata tables can be loaded into CQPweb from uploaded plaintext files. Alternatively they can be generated from data already present within the indexed corpus. See 《*XREF*》 below. **TODO**

7.4 XML metadata

(Explanation of how XML metadata can be applied in different ways)

...

Unlike text metadata, XML metadata does not normally use a metadata table. So to include XML metadata, you would normally incorporate it directly into the XML tags in the underlying corpus as represented in your input file.

The exception to this is metadata for ID-linked XML attributes. Attributes of this kind which *do* use a metadata table, which therefore works in a similar way to the text metadata.

This is explained below XREF.

7.5 The different possible datatypes

Currently, CQPweb supports the following datatypes for text/XML metadata fields.

- Free text
- Classification
- Unique ID
- ID-link
- Date (currently under development)

Note that all *corpus* metadata items are necessarily free text.

7.5.1 Free text

Free text is the most basic kind of metadata. An item of free text metadata can contain any string (up to a length of 255 bytes), although it may not contain tabs or line breaks. Free text is the best datatype to use for metadata that can vary across every single item, such as (for a text) the title, the source URL, and so on - things unlikely to be repeated within the corpus and therefore unsuitable for encoding as categories within a classification.

Free text metadata fields are allowed to contain links to external resources of one kind or another. These are detected by examining the "prefix" of the field value, which is the part of the value before the first colon.

Some of the recognised prefixes are the standard URL specifiers - *http*, *https* and so on. This means that web addresses, inserted without any modification, will "count" as having a prefix, and the special behaviour will be applied.

Other prefixes are special flags that CQPweb recognises as cues to treat the metadata value as a particular kind of link to an external resource. The full list of prefixes is as follows:

http This indicates a link to an external resource on the internet. Standard web addresses have this prefix (or one of the following two). It will be shown as a link in CQPweb. Optionally, the value can specify the clickable text for the link by adding it after the URL (with the URL and the link text separated by the pipe symbol, `|`). If no link text is given, the URL will be used as the link text.

Example: `http://www.site.net/info.html|Explanation`

https Works the same as *http*.

ftp Works the same as *http*.

youtube This prefix indicates that the URL is a link to a video on YouTube. This will be rendered as an embedded YouTube player. The YouTube link does not need to include *http://* or *www.*, but it's not wrong if it does! In fact, it's also possible to give just a YouTube video ID after the prefix.

Example: `youtube:youtu.be/bfhFdffdlsk`

video This is a link to a video file; it will appear as an embedded player, with an additional file download link.

Example: `video:http://mysite.com/vid.ogv`

audio This is a link to an audio file; it will appear as an embedded player, with an additional file download link.

Example: `audio:http://mysite.com/audio.oga`

image This is a link to an image file (of any kind that the browser can open, including PDF for instance). It will appear as a clickable control which, when clicked, will load the image into an overlay on the page.

Example: `image:../images/orig-text-a1.pdf`

no prefix The text of the value will be printed in the interface as-is (but with any HTML code escaped for the sake of security).

7.5.2 Classification

《*this needs more work* 》 **TODO**

Classifications – for items where each text falls into one of a limited number of categories (e.g. written vs spoken) In this case the field values must be handles (Unix words as described above), NOT multi-word explanations

A classification can have a maximum of 65,535 different categories.

7.5.3 Unique ID

《*explain*》 Unique IDs are handles, so all the rules of handles apply: see 《*XREF*》 **TODO** **TODO**

7.5.4 ID link

《*This is where the concept of idlink is explained. Use spoken corpus as the explanation*》 **TODO**

Notion of INDIRECTION

[datatype = idlink] implies lots of other fields in the associated metadata table, which in turn all have datatypes

But multiple indirection is not allowed.

AN EMAIL I WROTE ON THE LIST TO EXPLAIN THIS 《*turn this into coherent paragraphs*》 **TODO**

```
The idea is that this offers a layer of *indirection* for the XML.

The paradigmatic case of an ID link is speaker metadata.
```

```
In a spoken corpus you have lots of utterances (<u>) and
you very often want to do operations within certain utterances
and not others based on features of the speakers.
 EG you might want to search only within speech by males,
or by people in a particular age group.

You COULD add XML attributes for each of these things ie

<u speaker_age="12" speaker_sex="male">Hello!</u>

But this is not a terribly good design, because age/sex are
not features of UTTERANCES, they are features of SPEAKERS.
This speaker will always be male and 12 in this corpus,
so why is it necessary to repeat this on every utterance?
The answer is, it is not.

The IDLINK datatype allows us to model this kind of indirection.
Instead of marking speaker features on utterances,
we can have a separate table for speakers...

ID    age   sex
===============
A001  12    m
A002  65    f

.... which is then referred to by the IDLINK attribute.

<u who="A001">Hello!</u>

So, instead of the data chain going Utterance -> sex ,
there is another layer of indirection: Utterance -> speaker -> sex.

It's called an IDLINK because once we declare the datatype of s-attribute
u_who to be IDLINK, we promise CQPweb that an IDLINK metadata table,
with all the right IDs listed, will be available. That is,  that the
content of the IDLINK  (u_who) always LINKS to an ID that exists elsewhere
(in the Speaker metadata table, which is therefore an IDLINK metadata table).

All this is then opaque to the general user, who can specify " find
instance of word X where speaker is male " in a restricted query, for instance - CQPweb will t

  - use the IDLINK table to look up the IDs of the speakers where sex = m
  - use the CWB index to find the list of regions in the corpus where u_who
    is equal to one or other of those IDs (IE utterances by one of those speakers)
  - search within only those regions of the corpus for word X

Note this is SIMILAR to how text metadata works (if you search within genre
"fiction", then  CQPweb looks up the texts where the "genre" column contains
"fiction", and searches only within those texts) but not the SAME.
```

The key difference is that text_id is a unique identifier, ie each text ID
occurs in the corpus once and exactly once.

However, IDLINKS aren't unique. There can be many, many utterances where who="A001".
A0001 is unique *in the Speaker metadata table*.
This is why we talk about "u who" as an IDLINK rather than an ID:
it is not an identifier, but something that links to an identifier.

===========================

Currently, it is not possible to have IDLINKS as a datatype for text metadata.

I was uncertain about this decision, as there is a clear use case: where one author
writes many texts within the corpus, it would make sense for the "author" column in
the text metadata table to contain an IDLINK to a separate Author metadata table
which would contain things like author sex, age, domicile etc.

I decided against this for two reasons.

First, this system for doing Restricted Queries based on things like utterances was
already *very* difficult to make work. Making it possible for there to be ANOTHER
layer of indirection might have driven me mad. Wibble.

Second, if you look at corpora in practice, people tend not to mind making the sex
of an author, say, part of the text metadata rather than having the author-people
as a separate data entity. This is the case for the written BNC for instance - in CQPweb's
predecessor, BNCweb, "sex of author" is a "written restriction" (IE text metadata).
So I just went along with this way of doing it.

now, after all that background, re Chao's question:

Since the assignments are texts, features of the students
who wrote them could be included as text metadata columns.

And if one text metadata column in a unique id for the speaker,
that makes it easier down the line to track the progress of individual
students (you can say things like "create a subcorpus of texts where student=A001
in module=101, and compare a subcorpus of texts where student=A001 and module=201.
  - and do many other comparisons, if you have the right metadata for the texts.)

This is what Jiayue recommended, but hopefully the reasons why what you want here
is text metadata rather than an idlink now make sense.

-----Original Message-----
Subject: Re: [CWB] Example of metadata file?

Hi

My solution would be to use a metadata file (ascii text file, tab
separated values) like this:

```
A201 A 201
B201 B 201
C201 C 201
D201 D 201

The first column are the text_id's; the other columns are used to make
"text categorisation". In this way the four texts are linked clearly to
the same student.

> Hello all,
>
> First time poster and also want to try if my subscription works. I do
> not have any linguistic background, so please be gentle if I am asking
> silly questions.
>
> I am wondering if any one could provide a comprehensive metadata file
> example with some brief explanation on how CQPWeb can utilise the
> information? I am particularly interest in the LinkID part and assuming
> this could be used for threading different articles in a corpus?
>
> In my example, handed in assignments from various semesters are compiled
> as a corpus, each assignment is a text file with a unique text_id. Is it
> possible to give each assignment various linkIDs to show how the student
> progress through all semesters? For instance, student 201 has four
> assignments in semester A, B, C, D. If I associate four columns of
> linkID (A201, B201, C201, D201) on all his four submissions, will I be
> able to analyse the progress/change in words for this individual student
> in CQPWeb?
>
> Not sure if this is how the linkID and other metadata are designed for,
> besides classification and description. Please correct me if this makes
> non-sense.
```

7.5.5 Date

《*(currently under development)*》 **TODO**

7.6 Metadata templates

Just as you can create templates for particular structures of annotation and XML, and reuse those templates across corpora, you can do the same for metadata strucut (see 《*XREF prev chapter on templates*》), **TODO**

7.7 Matadata file format

A metadata input file should be formatted as follows:

- It should be a plain text file.
- The encoding should be UTF-8 (or ASCII).
- The line breaks should match the format of the underlying operating system: CR+LF (U+000d, U+000a) on Windows, LF (U+000a) on UNIX-like systems.
- The file should *not* begin with a Unicode "byte-order mark".
- The data should be arranged in a tabular format, where
  - *columns* are delimited by the tab character;
  - *rows* are delimited by line breaks;
  - and therefore, individual values cannot contain either tabs or line breaks.
- The first column must contain the unique ID codes (text IDs for text metadata; the ID-link IDs for ID-linked metadata).
- No ID code can appear more than once; every text in the corpus must be listed.
- However, it is not necessary for the text IDs to be in any particular order.
- This first column is implicit in the declaration of the metadata; you **do not** include it explicitly in the definiton of the metadata (whether declared *ad hoc* or with a template).
- Every subsequent column represents a metadata field, as described when the metadata is inserted into CQPweb.
- The file itself must contain no header row.
- If a column contains values for a field of datatype *classification*, *unique ID*, or *ID-link*, then all its values must be valid *handles*. See section 6.3.
- If a column contains values for a field of datatype *date*, its contents must be formatted using CQPweb's date formalism. ⟪*XREF!*⟫ **TODO**
- Columns of datatype *free text* are not restricted in what they can contain, except for the general rule (noted above) that they may not contain tabs or linebreaks.
- Empty values should be represented as zero-length strings - that is, if there is nothing in a given field on a given row, then there should simply be nothing between one tab and the next.

This format is often called *TSV* for "tab-separated values" (by contrast to *CSV*, "comma-separated values", a text-based file format associated with Microsoft Excel).

In SQL terms, TSV is equal to the format required by the `LOAD DATA INFILE` command with escape-sequence interpretation turned off within field values (using the command `FIELDS ESCAPED BY ''`). In fact, this is precisely the command that CQPweb uses to load the metadata! See `http://dev.mysql.com/doc/refman/5.7/en/load-data.html` for more detail.

7.8 Installing metadata

Text metadata needs to be installed for every corpus. Without it, queries don't work. So this should be the first thing you do after the CWB indexing process has run to completion (see ⟪*XREF*⟫). **TODO**

There are four possibilities for creating text metadata:

- *From file.* The most common approach: install text metadata from a text file on the server. The structure of the file can be described either by specifying it *ad hoc* or by referring to a predetermined metadata template 7.6
- *From XML.* This approach translates the values of one or more XML attributes, encoded in the CWB index as s-attributes, into fields of text metadata.
- *Minimalist.* This is the approach to use if you do not have or do not want to use any text metadata. It sets up the text metadata structure in the database, allowing queries to work, but leaves that structure empty.

All

Minimalist

From file Using template

From file ad hoc

From XML

After you install text metadata, the *Manage metadata* page for the corpus will change. Instead of showing options for installing the text metadata table, it will show a single control allowing you to *Reset the metadata table* (i.e. delete it, allowing reinstallation). It should be noted that this involves total loss of the existing data!

⟪*Manage text caTEGOIRIES*⟫ **TODO**

Under Admin tools – manage text categories

Briefly,

- this page will give you a form for each text-metadata field that was installed with the datatype "classification" - each of these forms will list all the category handles that exist within the given classification - by default, category handles are mapped to a "description" that is the same as the category handle itself

XREF to notion of "descroptin" in prev chap/.

- BUT you can use the forms here to change the category descriptions to something more user-friendly - since category handles are limited to short codes with no spacing or punctuation this is often useful - if you do this, then the "descriptions" will show up in a whole lot of different places in the user interface instead of the category handles, including: – restricted query form – concordance header for a restricted query – distribution display – text metadata page.

⟪*installing idlink metadata*⟫ **TODO**

8 Parallel corpus data

8.1 Introduction

CQPweb supports parallel corpora as of version 3.2.22. This chapter explains how to link parallel corpora together, and how the display works once the alignments are set up.

8.2 Setting up parallel corpora

Parallel corpora are linked by the existence of *alignment attributes* (*a-attributes* for short) in CWB. For a full understanding of how a-attributes work, you are referred to the CWB documentation, especially the **Corpus Encoding Tutorial**.

The key feature to note about a-attributes is that they presuppose the existence of a pair of corpora, a *source* (to which the a-attribute belongs) and a *target* (at which the a-attribute points).

This means that, unlike other corpus data attributes, a-attributes cannot be created when a corpus is indexed into CWB; they must be added afterwards.

CQPweb builds on this procedure. Parallel corpus data is managed as follows:

- First, install the two corpora separately in CQPweb.
- Second, use command-line CWB tools to generate the a-attribute(s).
- Finally, return to CQPweb to register the alignment.

A note. If you install a corpus that has already been indexed via command-line CWB, it's possible that the corpus will already have one or more a-attributes. If so, these a-attributes **will be ignored** by CQPweb when it imports that corpus's registry data. This is because there is no guarantee at the point of installation that the *target* corpora for those a-attributes have also been imported into CQPweb - so it would be dangerous to register the a-attributes. In short, it is *still* necessary in such a case to register the alignment within CQPweb as a separate action.

8.3 Naming alignment attributes

A-attributes always have the name of the target corpus. So if you have two corpora `corpus_a` and `corpus_b`, which represent the same texts in Language A and Language B respectively, then an a-attribute from source `corpus_a` to target `corpus_b` will have the attribute name `corpus_b` but will *belong to* `corpus_a`.

Since a-attribute handles are actually corpus handles, they necessarily have names that follow the corpus handle rules. Note in particular that they cannot contain hyphens. The CWB encoding tutorial recommends the use of hyphenated ISO language codes when using parallel corpora, e.g. `somecorpus-en` and `somecorpus-fr` for the English and French parts of a parallel corpus respectively. For use in CQPweb, however, this practice should not be followed. However, an underscore *can* be used instead if you wish (i.e. `somecorpus_en` and `somecorpus_fr`).

It's especially worth noting that the sample *Europarl* parallel corpus made available for download on the CWB website (http://cwb.sf.net) follows this convention. Both the Europarl corpora, and the a-attributes they contain, need to be renamed to use underscores instead of hyphens before they can be imported into CQPweb.

8.4 Creating alignment attributes

There exist multiple methods for indexing alignment attributes. They are discussed in the CWB **Corpus Encoding Tutorial**. All basically revolve around identifying (sets of) ranges of corpus positions which are considered to be "aligned", i.e. translation-equivalent, across a pair of parallel corpora. An a-attribute contains the data for such a mapping.

CQPweb does not currently have a web-based interface to the alignment generation and importation tools. You must use the command-line tools directly.

The steps are as follows:

- Either create or appropriately reformat the alignment data ready to be imported.
- Create an a-attribute using the appropriate encoding utility.
- Modify the registry file for the source corpus to add a declaration of the new a-attribute (some tools for a-attribute creation do this for you).

In CWB, alignments of a pair of parallel corpora can be, but do not have to be, bidirectional. That is, creating an a-attribute for `corpus_b` within `corpus_a` only creates an A-to-B link; no B-to-A link will exist unless you *also* create an a-attribute for `corpus_a` within `corpus_b`. CQPweb replicates this feature of the underlying system: it supports both scenarios, i.e. things will work just fine with either one-way or two-way links.

Also as in the underlying CWB, it's entirely possible for a CQPweb corpus to have more than one a-attribute, where each one has as its target a different parallel dataset.

Finally it should be noted that the character encodings of any pair of corpora need to be identical or at least compatible (e.g. ASCII/Latin-1). CQPweb normally abstracts away differences in the character encoding of the CWB index - regardless of what encoding the index uses, CQPweb works in UTF-8. However, parallel corpora represent the one case where this doesn't work. If `corpus_a` is in Latin-1 and `corpus_b` is in Latin-2, for instance, then the text of the latter will be treated in all corpus display functions *as if it were Latin-1* - which will be wrong. The preferred solution to problems arising here is to use UTF-8 for all sets of corpora which must be linked together as parallel.

8.5 Registering alignment attributes with CQPweb

Once you have created the a-attributes, you must then register them with CQPweb.

This is done using a one-button control that can be found within the Corpus Admin Tools, on the **Manage parallel alignment** page. The button, labelled *Click here to scan the registry for newly-added alignments*, does exactly what it sounds like it does!

To be more specific, when this function runs, the following happens:

- CQPweb loads the corpus's registry file.
- It searches the registry of a-attribute declarations.
- It subjects any that it finds to two checks.
  - First, does a corpus by the name of that attribute exist in CQPweb?
  - Second, is the alignment already registered within CQPweb?
- If the answers to these two checks are *Yes* and *No* respectively, the alignment is registered.

Once an alignment is registered, it will appear in a table headed *Existing alignments* on this page.

Unlike other types of attribute, it is not possible to add a long description to be used on screen for an a-attribute. Instead, the "description" for an a-attribute handle is always the same as the target corpus' onscreen title (as specified when the target corpus was indexed, and managed on its **Corpus settings** page; see 3.3.1).

8.6 How alignment attributes can be used

Once one or more a-attributes has been registered in CQPweb, it has the following effects on the behaviour of the system, as seen by all users:

- Various additional controls appear, allowing the user to turn on or off the display of parallel data in query results, or to switch from one aligned corpus to another.
  - An extra dropdown control listing available parallel corpora appears on the query form.
  - A similar control is present in the concordance display.
  - A similar control is present in the extended-context display.
- When the display of parallel corpus data is activated, it is shown in a separate table row below the main result in either concordance or extended-context display.
- Options are also available to download one or more parallel corpus regions in the *Download query* tool. Parallel corpus data is printed as extra columns in the downloaded text file.

XML visualisations defined for the *source* corpus (see section 10.4) will be applied to the parallel text *if* s-attributes of the same name exist in the target corpus. XML visualisations defined for the *target* corpus are always ignored.

Similarly, when the primary annotation for the *source* corpus is visible, an annotation will only appear for the parallel text if the target corpus has a p-attribute with the same handle - in which case, moreover, the annotation shown is always the one whose name matches that of the source corpus' primary annotation, *not* the primary annotation of the target corpus.

One final note of warning: the chunk of parallel data displayed is *the entirety of the zone of the target corpus* that is aligned to the zone in the source corpus in which the query match is found. This may be a longer or shorter stretch of text than the co-text shown around the match.

Consider the following two - quite likely! - contingencies.

In concordance view: the width of concordance is very often set in words, whereas alignment is typically built on top of s-attributes. The whole of an s-attribute unit may be shown for the parallel text, even if a span of (say) +/- 10 tokens is showing for the main corpus. This is not avoidable, as there is no way to reduce the size of the unit of parallel co-text; any reduction might easily exclude the part of the parallel text that actually relates to the query match. You are recommended to set the concordance display width to be equal to 1 of whatever unit the alignment is built on (typically sentences), rather than the default word-based width, in order to mitigate this issue.

In extended-context view: only one unit of parallel text can be shown, and unlike the source-corpus context, it can't easily be widened (because, as noted above, only that unit of the target-corpus directly parallel to the source-corpus unit in which the match appears is returned by CQP when a-attribute display is switched on).

8.7 Parallel corpora and user privileges

To access any data in a parallel corpus (in concordance or extended context, or in a download), a user must have *at least* restricted-level access to that corpus.

That is, assuming Corpus A has an a-attribute linking it to Corpus B, that a-attribute will be visible to the user in the interface for Corpus A only if they have some privilege that allows them access to Corpus B.

Otherwise, it will appear to the user as if the a-attribute for Corpus B does not exist.

Typically, one would set up the different component corpora of a parallel corpus so that *all the same* users and user groups are granted the use of those component corpora.

A user *does not* need to have access to a parallel corpus to use its a-attribute within the body of a CQP-syntax query (e.g. as a limiting factor on the query; see the CQP Query Tutorial); this is allowed even without an access privilege, because it does not involve access to the text of the parallel corpus. However, this possibility is not openly flagged in the interface; the list of attributes that appears on the query form only includes a-attributes that can be viewed, omitting any a-attributes that can be used but not viewed.

9 The Common Elementary Query Language (CEQL)

9.1 Introduction

CEQL is the Common Elementary Query Language, designed by Stefan Evert as a novice-friendly alternative to the normal CQP query syntax. In the CQPweb interface (and in BNCweb), it is often referred to as "simple query syntax".

CQPweb has a built-in PDF handout summarising the CEQL language. A link to this appears next to the main Standard Query / Restricted Query tools. However, a more complete introduction to CEQL *for the end user* may be found in Hoffmann et al., *Corpus linguistics with BNCweb.*

CEQL has two major features: (1) simplified "wildcards" that make specifying patterns less complex than with regular expression syntax; and (2) special shorthand for quick access to certain word annotations, i.e. Corpus Workbench positional attributes (p-attributes represented by the columns in a vertical input file).

CEQL is highly configurable. This chapter focuses on how system administrators can activate different CEQL features on a CQPweb server.

9.2 CEQL syntax: shorthand access to positional attributes

In normal CQP syntax, access to p-attributes is via a transparent but fairly wordy system (Boolean statements testing the values of different p-attributes, referred to by name). Using it requires knowledge of what p-attributes a corpus has, and what kinds of values they contain. By contrast, CEQL provides a number of shorthands for frequently used p-attributes

The main CEQL shorthands are as follows.

- `pattern` search for "pattern" in the default word annotation
- `_pattern` search for "pattern" in the *primary annotation*
- `{pattern}` search for "pattern" in the *secondary annotation*
- `_{pattern}` search for "pattern" in the *tertiary annotation*

CEQL consists of a core grammar which can be extended. The above four query types are part of the core grammar. CQPweb adds a single extension to the core grammar; this is the following query type:

- `{pattern1/pattern2}` search for the combination of "pattern1" and "pattern2" in the *combination annotation* (or *combo annotation* for short)

These shorthands were designed around the kinds of annotations typical in English corpora, as they were initially used within BNCweb, as follows:

- `word` search for a given word
- `_tag` search for a given CLAWS5 (BNC-style) POS tag
- `{lemma}` search for a given lemma
- `_{tag}` search for a given wordclass (lemma-level POS from the Oxford Simplified Tagset)

- `{lemma/tag}` search for the combination of the given lemma and the given wordclass

CQPweb was designed with BNCweb user interface compatibility in mind, so that the necessary functionality to replicate this setup is present. As a general recommendation, it is best to use CEQL in this default style unless you have a particular reason for doing otherwise

However, CEQL is fully configurable within CQPweb, so these different syntaxes can alternatively be used to target whatever p-attributes you like. For example, you could make the `{pattern}` syntax target a semantic tag, or make the `_pattern` syntax target a phonetic transcription.

9.2.1 The primary annotation

9.2.2 The secondary annotation

9.2.3 The tertiary annotation

9.2.4 A side note: the Oxford Simplified Tagset

9.2.5 The combination annotation

The {`lemma/tag`} extension is a response to the fact that lemma headwords are often ambiguous in English: *run* is the headword (citation form) for both the verb lemma *run* (forms *run, runs, running, ran* and the noun lemma *run* (forms *run, runs*).

Searching for one of these, and not the other, requires access to both lemma form and POS tag. This is easily accomplished

⟪*more*⟫ **TODO**

10 Controlling query visualisation

In CQPweb, *visualisation* is a covering term for anything to do with how corpus data is rendered in the web interface, but with particular reference to the display of text around a query hit in the concordance display and in the extended context display. This chapter describes how the system administrator can control different aspects of the visualisation process.

10.1 How the primary annotation affects visualisation

《*How the primary annotation affects visualisation*》 **TODO**

10.2 Setting up an "alternate" view for context display

《*Setting up an "alternate" view for context display*》 **TODO**

10.3 Using position labels

《*What hey are; quick note on the control*》 **TODO**

《*don't forget to explain about the span-class they are wrapped in and the possibility of doing things with extra code files, q.v.*》 **TODO**

10.4 XML visualisations

10.4.1 Introduction

By default, none of the corpus XML (s-attributes) are displayed in either concordance or extended context view. Nor are they included in downloaded queries. This section explains the use of the XML visualisation system, which allow you (a) to display corpus XML in query results, and (b) transform the raw data of the s-attributes into customised HTML for easier interpretation by users.

An "XML visualisation" is a specified mapping from an s-attribute (XML element or combined element-attribute) to some HTML. These are created through the CQPweb web interface; see 10.4.2. The mapping takes the form of an HTML template written with a small "whitelisted" subset of HTML, with all complex features blocked to avoid the risk of Cross Site Scripting (XSS) vulnerability: see 10.4.5.

You can further manipulate the HTML generated by your visualisation with *extra code files* (see 10.4.6), additional JavaScript or CSS files which you install server-side and which can apply almost unlimited further styling to the visualised XML.

It should be noted that all XML visualisation operates at the corpus level. That is, the visualisations you set up for one corpus do not affect any other corpus on the system. Of course, it is easy enough to copy the HTML code of a visualisation from one corpus's interface to another's!

10.4.2 Creating and managing XML visualisations

The interface for creating and managing XML visualisations can be found within the Corpus Admin Tools, on the **Manage visualisations** page. This page also contains the controls for field-data display mode (see section 10.5) and for position labels (see section 10.3).

The various control forms for XML visualisation are towards the end of the page. In order, you will see:

- The interface for using extra code files: see 10.4.6
- The fallback control: see 10.4.7
- A list of existing XML visualisations
- A form to create a new visualisation command.

To create a new visualisation, you must select the XML element/attribute (s-attribute) that you want to render, and specify whether you are creating a visualisation to appear at the start of stretches of text of that type (start tag) or at the end of such stretches of text (end tag).

Next, you need to enter the actual HTML code. See 10.4.5, 10.4.4 for the content you are allowed to use here. "HTML code" in this context *includes* any plain text content you might want to use (e.g. a single punctuation code to visualise the boundary indicated by the XML) - and, as noted below, for visualisations intended to be used in downloaded queries, it is usually *preferable* just to use plain text.

Next, you must specify whether the visualsiation is to be enabled (a) in the concordance display, (b) in the extended context display, (c) in downloaded queries. It's possible to use the same visualisations in two or three of the possible places. It's also possible to use separate visualisations for the same bit of XML in concordance and/or context and/or downloaded queries.

Finally, you can make the visualisation for a start tag conditional on the value of the XML annotation; this is explained in 10.4.3.

Once you have created a visualisation command it appears on the list of existing visualisations. The definition of the target of a visualisation (XML element, start vs. end tag, condition for the value) cannot be changed once it has been created: if you need to change these, delete the visualisation and create a new one. However, you *can* change the HTML code of an existing visualisation, plus of course activate/deactivate its use in the two relevant displays and/or query download.

10.4.3 Conditional XML visualisations

XML visualisations for start tags can be made *conditional* on the value of their annotation. This means that the visualisation only applies when the annotation meets some criterion.

Currently, the only kind of condition that can be applied is a *regular expression match*. That is, when you create the condition, you specify a regular expression that the value is compared to. If it matches, the visualisation is used. If not, not (although if there is another visualisation active with a different condition or no condition, that one might well apply).

The regex flavour used in the conditions is *Perl-Compatible Regular Expressions (PCRE)* (the same regex flavour used by CQP. However, unlike regexes in CQP, the regexes for conditional visualisations are not anchored: it is not necessary for the regex to match the *whole* value, only that some part of the value must match the regex. If your regex pattern includes any forward slashes, you must escape them with a backslash. This is in addition to all the usual escaping rules for PCRE regexes.

If there is more than one conditional visualisation, then when an XML boundary is rendered, their conditions are checked in order (the conditional visualisation with the longest regex first, and then by descending length). The one that takes effect is the first where the condition is fulfilled (i.e. where the value of the XML attribute contains a match to the regex anywhere within it).

Example use cases for conditional visualisations might include:

- You have an XML attribute with a small number of values - for example, a pragmatic categorisation of annotated chunks of discourse (question/request/command/statement etc.) You want to visualise the beginning and end points as square brackets "[...]" in the interface. If you create a series of conditional visualisations, each with one of the possible values as the regex to be matched, then you could assign different appearances to the opening of each different pragmatic function - e.g. colour them differently depending on the function type.

- You have an XML attribute with two possible values, where one is assumed (the default) and the other applies in only a limited number of cases. By creating a conditional visualisation for the latter but not the former, you can have it appear in the interface only when it has its less common, unexpected value and let it remain invisible when it has the more common, default value.

Be careful with conditional visualisations. It is necessary to consider all possible contingencies, because it will always cause problems to leave any bit of XML uncovered.

For instance, if you have a single visualisation that applies to `s_type` when its value matches "question", that leaves all other values of `s_type` with no visualisation. They will instead be rendered as nothing. You can specify this explicitly by creating an empty default, that is, something like this:

- Visualise start of `s_type` as [some block of HTML], with condition that the value must match "question";
- Visualise start of `s_type` as [leave empty], with no condition.

The second condition will apply to all instances of `s_type` with other values, causing them simply not to show up in the display.

Without such an explicit visualisation, any XML boundaries that are not matched by any of the relevant conditions default to being rendered as nothing anyway.

10.4.4 The embedded variable

As well as making the use of a visualisation conditional on the value of the instance of the s-attribute, it is also possible to actually include the value in the HTML rendering.

This is done using the *embedded variable*. This is, quite simply, a sequence of four dollar signs (`$$$$`) appearing anywhere in the HTML code of an XML visualisation.

Note that the embedded variable (a) only works for start tags, not end tags; (b) cannot be used for s-attributes that don't have values! When the s-attributes have, as is usual, been created from structured XML, the s-attribute "head" of the "family" lacks values, but the "children" representing the XML attribute-value pairs have values.

So, for instance, if you have `<p>` elements in your corpus with attributes `num` and `type`, then the s-attribute `p` will have no values (so the embedded variable won't work), but the s-attributes `p_num` and `p_type` *will* have values that can be inserted into the HTML using the embedded variable.

10.4.5 HTML allowed in XML visualisation code

Only certain HTML elements are allowed in XML visualisation code. When you add a new visualisation, all the HTML in your code will be compared to the "whitelist", and any HTML that is not on the whitelist will be escaped.

So, for instance, `<script>` is not allowed in visualisation code. If you try to install a visualisation containing `<script>`, it will be escaped to `&lt;script&gt;` and appear literally in the browser as `<script>`.

The HTML whitelist is as follows. For convenience, it is divided here into (a) simple codes (just tags with no attributes) and (b) complex codes which require an attribute.

You can use the following simple HTML formatting codes (just tags with no attributes):

- `<b>...</b>` - render text as bold
- `<i>...</i>` - render text as italic
- `<u>...</u>` - render text as underline
- `<s>...</s>` - render text as strikethrough
- `<sub>...</sub>` - render text as subscript
- `<sup>...</sup>` - render text as superscript
- `<code>...</code>` - render text as code (usually means a monospace font)
- `<br>` - add a line break (*hint*, for the appearance of a paragraph break, use `<br>&nbsp;<br>`)

Of course, nothing forces you to close HTML tags that you open in an XML visualisation; however, if you don't, you may find that your formatting interacts peculiarly with CQPweb's own rendering.

The available complex codes are as follows:

- `<img src="SomeURL">` - embed an image; the URL can be relative or absolute
- `<span class="SomeCssClass">...</span>` - apply one or more CSS classes to a span of text (to link a class to one of the other tags, wrap it in a span)
- `<a href="http://SomeUrl">...</a>` - create a link (http/https only) (all links will open automatically in a new browser window or tab)
- `<bdo dir="ltr/rtl">...</bdo>` - mark text as left-to-right / right to left: CQPweb uses these tags with right-to-left alphabet corpora, and you may need to use them in your visualisation code to stop odd interactions between your HTML and the direction of the text it appears in

For both simple and complex tags you need to use the lowercase form shown above, i.e. `<br>` not `<BR>`; full HTML allows either, but CQPweb only whitelists the former. Similarly, you must use double quotes for attribute values, even though full HTML allows single quotes.

As well as HTML tags from the whitelists above, you can also use HTML entities. You don't need to use entities for accented characters or non-basic punctuation, however, although these are two typical uses for entities, because the UTF-8 encoding is used throughout CQPweb, so you can just use the characters directly. One useful entity is `&nbsp;` (no-break space), which functions as a unit of "empty" text (note that actual whitespace is ignored).

It's worth pointing out that the following frequently-used HTML codes are *not* usable in visualisations, because they would interact badly with the HTML/CSS that CQPweb itself uses.

- `<p>` and `<h`*N*`>`
- `<div>`
- `<pre>`

- `<table>` and other table-structure tags

You can, however, still make use of these and other features of HTML by means of *extra code files*, explained in section 10.4.6.

When your create visualisations for use in downloaded queries, remember that any HTML will normally *not* be rendered: users will see the actual code in the plain text file they download. For that reason, it's normally better just to use a plain text visualisation here (possibly with the embedded variable).

10.4.6 Extra code files

The combination of HTML, CSS and JavaScript in a dynamic web page allows almost limitless variation in how information is presented. However, for security reasons, it is not possible to allow all the richness of HTML to be usable in the XML visualisations that are created via the interface: without the restriction to a limited subset of HTML, an attacker who gained access to a superuser's account would be able to insert arbitrary JavaScript code into the browser of any user who subsequently accessed that corpus.

However, the simple visualisation code permitted by the whitelisted-HTML may well be *too* restrictive for users who want complex rendering of data in their corpus XML. CQPweb has an additional mechanism to allow users to perform more complex formatting: the insertion of **extra code files**.

An extra code file is a file containing either CSS or JavaScript code. You can specify one or many such files for each corpus (with separate lists for concordance view and context view to allow different renderings in each). These files will be linked in the headers when a concordance/context page is generated. The overall appearance of the concordance/context will thus be governed by the combination of CQPweb's built-in styling and the extra CSS/JavaScript you have added.

An extra CSS file allows to you write arbitrary stylesheet code to change the appearance of text within a visualisation. For instance, if you have used a `<span>` element to assign a class to some piece of text, you can use an extra CSS file to apply any style(s) you want to spans with that class.

An extra JavaScript file is even more powerful. You can manipulate the HTML doument tree in more-or-less unlimited ways. In addition, CQPweb uses the **jQuery** library, and your extra code can use this library too. jQuery uses CSS selectors to pick HTML elements to operate on, so again, the assignment of classes to particular bits of text in the visualisation code allows you to later write a JavaScript/jQuery function to pick out those specific parts of the document and modify them as you wish. An example of this is provided below.

You can, of course, use extra code files to modify aspects of the concordance and context views other than just the XML visualisations. But it is that combination which offers the most powerful options.

There are three steps to add an extra code file.

- Write the code.
- Insert the file with the code into CQPweb (must be done on the server, cannot be done via the web interface).
  - CSS files should be placed in the `css` subdirectory of the main directory.
  - Javascript files should be placed in the `jsc` subdirectory of the main directory.
  - Note that extra code files must have the canonical file-extensions: `.css` and `.js` respectively.
- Finally, activate the code file for the corpus you want to apply it to.

Extra code files are activated using the **Manage visualisations** display, under the heading *Extra code files for visualisation.* Select a file from the dropdown and press "Add this file" to activate it; use the [x] buttons to deactivate code files from a corpus. Deactivating does not delete the file, and only affects the particular corpus in question - any other corpora where the same file is activated will be unchanged.

There are two separate forms for concordance view and context view, allowing you to have very different layouts in concordance and context views. Here are two examples.

In our first example, let's imagine that the corpus contains `<u>` elements for utterances, with a `speaker` attribute, giving the s-attribute `u_speaker`. Let's assume we have created the following visualisation for `u_speaker`:

- `<span class="swanky-speaker-label">$$$$:&nbsp;</span>`

We can exploit this with an extra CSS file to make speaker labels stand out more distinctively in the display. The CSS file could contain the following:

```
/* --------------------------- THIS IS THE CSS FILE */
span.swanky-speaker-label
{
  color: pink;
  font-weight: bold;
  font-family: "Comic Sans", sans-serif;
  font-size: 16pt;
}
/* --------------------------- END OF THE CSS FILE */
```

When this file is specified as an extra CSS it will apply this highly tasteful additional styling to all speaker labels.

If we want to do something more advanced (for example: make all speaker labels clickable, causing an information box to appear when clicked) this can be done in an extra JavaScript file using either the native JavaScript method of interacting with the DOM (document object model) or the facilities of the jQuery library. The following code exemplifies the latter:

```
/* --------------------------- THIS IS THE JAVASCRIPT FILE */
function show_my_info(speaker)
{
  /* The following is a placeholder for the more complex
     behaviour you would want in a real situation. */
  alert ("The speaker is " + speaker + " !");
}

$(document).ready (function() {
  /* this passes the whole content of the span to the show_my_info function
   * (including any extra matter added around the value, like a <:> (See above).
   * In reality, you would usually want to make it easier for the jQuery code
   * to access just the value, to do interesting things with it.
   */
  $("span.swanky-speaker-label").click(function () {
    show_my_info( $(this).html() ;
    return true;
  } );
} );
/* --------------------------- END OF THE JAVASCRIPT FILE */
```

One thing that can be done using extra JavaScript files is to create renderings that involve the content of more than one s-attribute. For instance, imagine our original input-format utterance tags look like this:

- `<u speaker="ID_Code" type="QUESTION">`

If we want to make use of *both* the `code` and `type` values to build what appears in the concordance at an utterance boundary, the easiest way is to use a basic visualisation, and then join the two together using JavaScript.

First, we would add these visualisations:

- For `u_speaker` : `<span class="all-info"><span class="speaker">$$$$`
- For `u_type` : `</span><span class="type">$$$$</span</span>`

The key thing to note here is that the visualisations of attributes within the family of a single XML element will *always* appear directly together. So the above will always generate something like this:

- `<span class="all-info"><span class="speaker">ID_Code</span><span class="type">QUESTION</span></span>`

... which is easily manipulable via the HTML DOM.

```
/* --------------------------- THIS IS THE JAVASCRIPT FILE */
$(document).ready (function() {
  /* on document ready, run this function on each of the outer spans */
  $("span.all-info").each(function () {
    var outer   = $(this);
    /* extract the string contents of the inner spans */
    var speaker = outer.children().first().html();
    var type    = outer.children().last().html();
    /* having done so, we can now simply set the inner-html of
     * the outer span to the string we want */
    outer.html(
      "[[This is a <b>"
      + type
      + "</b> spoken by <b>"
      + speaker
      + "</b>]]"
      /* in reality you'd want more elaborate HTML! */
    );
  } );
} );
/* --------------------------- END OF THE JAVASCRIPT FILE */
```

Finally, just to note the obvious: extra code files can't be used with downloaded queries.

10.4.7 Fallback visualisation methods

Many corpora are set up with no XML visualisations (indeed, XML visualisations were not implemented until CQPweb had already been around for many years). In this case, the text from the corpus in the extended context display will all appear as a long, unbroken block.

To make extended context look a little nicer without the need to create a visualisation, CQPweb possesses a *fallback* method. When this is switched on, a double-line-break is rendered in context view *after every token made up only of non-medial punctuation.* This roughly simulates having each sentence in a separate paragraph. This will not work with every language or every type of corpus data. But in most cases it results in a reasonably-OK way to break up the wall of text.

If you have added a visualisation to insert breaks in context view, you might well want to turn off this fallback method. Of course, you might also just want to turn it off anyway! A control can be found to do this on the **Manage visualisations** view, under the heading *Visualisation fallback procedures.*

When a corpus is indexed, it has no visualisations initially. So the fallback is always switched on by default for newly-added corpora.

10.5 Field data presentation mode

⟪*Field data presentation mode*⟫ **TODO**

10.6 Field data mode as a workaround for parallel corpora

⟪*Field data mode as a workaround for parallel corpora*⟫ **TODO**

11 User accounts and privileges

11.1 Basic concepts

Access to CQPweb is based on *user accounts*. The user account has two functions. First, it determines what resources (corpora and analysis options) a given user has access to. Second, it provides a basis for (limited) personalised functions. For instance, every user account has its own query history, its own set of saved queries, its own set of subcorpora, its own concordance display preferences, and so on.

If you are running an online CQPweb system, then normally you will want to use CQPweb's functions for creating user accounts via a self-registration process. That is, someone who wants to use your CQPweb server should first visit the homepage, where they will find a link to a form for account creation. This is very similar to the process for signing up for an account on pretty much any website, so should not be challenging. There is also a tool in the Admin Control Panel for you to either create accounts for users yourself, or to send an invitation to a specified email address.

User accounts must be linked to email addresses. This is because knowing the user's email address is the only way to make it possible for them to reset a forgotten password. Again, this is pretty much standard procedure on the web.

Each user account belongs to one or more *user groups*. A user group is exactly what it sounds like: an arbitrary set of user accounts. There are two builtin groups: **superusers** and **everybody** - both of which do exactly what it sounds like they do!

User accounts can be added to groups other than **everybody** in two ways: either automatically at the time of account creation (based on pattern matching against their email address), or manually by you using the Admin Control Panel.

The final core concept is the *privilege*. A privilege simply defines some permission that can be granted to a user. This might be a level of access to some corpus or set of corpora, or permission to initiate a database operation of a certain size. Privileges may then be assigned to users individually, or to user groups. A link between a privilege and either a group or a user is called a *grant*. Each user then has all the privileges granted to them or to any of the groups to which they belong. This set of privileges determines what CQPweb will and will not let that user account do.

It should be noted that any user account in the group **superusers** automatically has every possible corpus-access privilege; other privileges can be granted or not granted to that group, as usual.

The system is very flexible, which makes it rather complex. This chapter explains different aspects of the system, and how you can control it.

11.2 User accounts

By default, users can create their own accounts using the self-registration system. However, if you don't want to allow access to anyone you have not vetted individually, it is possible to switch this system off: see 2.3.7. Self registration runs as follows: first, the user specifies the username and password they want, and supplies an email address (and, optionally, other information about themself, including their real name, location in the world, and organisational affiliation). They may also have to answer a CAPTCHA challenge, if you have switched that on. The system then sends an email to the address specified with a verification code. The user must click on the verification link: their account is then activated, allowing them to sign in with their username and password. This verification system (a) prevents the creation of accounts that the owner cannot retrieve in case of a forgotten username or password, since it guarantees that a correct email address has been supplied; (b) stops anyone creating an account for someone else.

- A known "gotcha" here: CQPweb's automated emails are known to be blocked by the spam filters used by Yahoo's free email service. There does not appear to be anyway around this. So if you have users on Yahoo mail, you will probably need either to validate their email addresses manually, or to instruct them to use a different free email service (Google Mail seems generally to work fine).

Whether or not self-registration is enabled, there are also tools available in the Control Panel for you to create accounts for people, as well as view (or delete) existing accounts Go to **CP >Users and Privileges >Manage users** to access these tools.

At the top of this screen is a summary of the existing accounts on the system. 《*Note that the tool for looking at a list of unverified accounts, linked at the bottom of the summary table, has not been written yet.*》 **TODO**

Next is the user-account search form. This is how you view the status of an individual account. There are two search tools. The first is the quick username search. Simply start typing the username of the account you wish to view: a list of suggestions will be provided as you type, and you can simply click on one of the links as soon as you have narrowed the list down enough. (Pressing TAB from the quick search box will move you through the suggestions list; press ENTER to view the selected account.)

The second search tool, the *Full search*, checks for the term you enter in three different fields: username, real name, and email address. The results include accounts where your search term appears *in the middle* of one or more of those fields. This is especially useful if you are looking for an account whose username you are not certain of: searching for a fragment of a person's name here is very likely to find their account. This tool has a more conventional search interface than the quick search - you must press the **Search** button to go to a table of results, each of which gives you a link to view the account details (see section 11.3).

Underneath the search tools are the account creation tools. The main account creation form is an abbreviated version of the user's self-registration form. The difference is that only a username, password and email address are needed. The optional fields (real name, affiliation, location) are not set here; the assumption is that if the administrator creates an account and tells the user what their credentials are, they will log in later and add these details themself.

This form generally has much less security than the corresponding self-registration form, based on the assumption that the administrator(s) know what they are doing:

- There is no CAPTCHA
- The password is not masked
- The password does not need to be typed twice

Similarly, when you create an account through this interface, validation by email is optional: you can cause the account to be automatically validated if you are certain that the email you've entered is valid; and you also have the option to leave it unvalidated without sending an email.

《*It is intended to add an "invitation" tool here, to allow people to be sent a "please sign up" link to the account creation form (prepopulated to the extent possible). However this has not yet been implemented*》 **TODO**

One final note on account creation relates to security.

If you're using a normal web server, CQPweb passwords are transmitted in plain text via HTTP. This is very insecure. Theoretically there is little danger in a user's CQPweb password being stolen. User data stored on CQPweb is seldom highly-sensitive, though of course there may be exceptions.

However, if self-registration is enabled, there is a high level of danger - since users select their own passwords, and it is well known that **most users tend to reuse passwords on multiple sites**. This means that a user's CQPweb credentials being stolen potentially exposes their accounts on other systems - in the worst case, highly consequential systems like social media or online banking.

To protect users from the consequences of password reuse, you have two options:

- Disable self-registration. Generate all passwords for users yourself, to prevent reuse, using highly non-memorable passwords to deter the user from going on to reuse the password you have chosen.
- Use HTTPS instead of HTTP. This is the recommended course of action.

One option that is known *not* to work is instructing users not to re-use passwords. Such instructions are routinely ignored - if the user even notices them in the first place.

CQPweb stores passwords internally in an encrypted form, so that they are protected even if your server is hacked. However, this does nothing to protect users if their credentials are transmitted in plain text via HTTP.

11.3 Viewing user account details

When you view a user's account details, you will see most of the basic information recorded about the user in CQPweb's internal database: their real name, email, and affiliation (as supplied when they signed up), plus also the time of account creation and the time the user last accessed CQPweb.

- If the date of account creation is either "0000-00-00" or a date in 1970, it means that the account was created before CQPweb started tracking the ages of user accounts (in version 3.1.0).
- If the last-visit date is either "0000-00-00" or a date in 1970, it means that the user has never logged in.

manually validating an ccount

deleting an account

resetting the password

corpus stats

setting the database size

⟪*account expiry*⟫ **TODO**

⟪*password expiry*⟫ **TODO**

⟪*bulk log out, and the log in list in the user profile. Here or in the prev section??*⟫ **TODO**

11.4 User groups

A group is exactly what it sounds like: a group of user accounts. You can organise user accounts into groups manually based on whatever principle you see fit; alternatively, you can organise users into groups automatically using patterns in their email addresses.

The point of user groups is that it is usually much more convenient to grant privileges, such as the privilege to access a certain corpus, to a group of users all at once, than to grant such a privilege to every user, one by one - especially if your server is open to the internet and anyone can create an account on it.

As noted in section 11.1, there are two builtin "special" groups, whose membership is not controlled in the usual way. They are:

- **superusers**, which contains all and only those users declared as system administrators in the configuration file (see 2.2).
- **everybody**, to which all users belong automatically, and from which no one can be removed.

Groups are created, managed and deleted via the Admin Control Panel: go to **CP >Users and privileges >Manage groups**.

- To add a new group, use the form at the bottom of the screen; all you need to specify to create a group is its name, which should be letters/digits/underscore only.
- The list of existing groups are tabulated at the top of the screen, in alphabetical order of group handle.
- This table allows you to enter a longer description for the group, which would normally be some *aide-memoire* to the group's nature and purpose.
- It also allows you to enter an *Auto-add regex*, which is explained below.
- After entering or modifying the description or regex, press **Update** to save your changes.
- Also in this table you will find [**x**] buttons to delete groups.

To add or remove users individually from a group, go to **CP >Users and privileges >Manage group memberships**. Next to each group on this screen is a pair of controls - one for adding users who are *not* already members, and one for removing users who *are* currently members.

A user can be assigned to as many groups as you like.

Users can also be added automatically to groups at the point of account creation. This works as follows. It is possible to associate a regular expression (regex) with each group (this is the *Auto-add regex* discussed above). When a new user account is created, CQPweb checks each group regex against the new user's email address. If there is a match, then the user is added to the group in question.

The typical usage case for this function is the situation where you have created a group that is associated with a particular institution. In that case, you can set up the group regex to detect email addresses "@" that institute's domain.

The regex flavour used is PCRE (Perl Compatible Regular Expressions). You can use any PCRE feature in an auto-add regex. The auto-add regex can be very long - up to 64 KB. Needless to say, it is not recommended to use such a long regex! The form limits you to 1024 characters; longer regexes can be inserted by direct manipulation of the MySQL database if necessary.

Changes to the regex are not retroactive. So if a user A signs up on Monday with an email address ending *@anytown.edu*, and on Tuesday you modify the regex for some group G so that it matches addresses ending in *@anytown.edu*, the existing user A will not be added automatically to group G.

(Similarly, users are never removed from groups based on a regex changing. So if user A is a member of group G, and group G's regex changes so that it no longer matches user A's email, user A will *not* be removed from group G in consequence.)

You can, however, force a re-application of the regex to all existing non-members via the **Bulk Add** tool. This can be found at the bottom of the **Manage group memberships** screen.

The first of the two **Bulk Add** tools is *Apply group's stored pattern-match to existing users.* To use this, select a group and press the "...run group regex against existing users" button. All existing user accoutns that are not already in the group will be checked, and added to the group if their email address matches. But note this is not exclusionary: if there are users *already in* the group whose email addresses do not, or no longer, match the regex, they will *not* be removed from the group by application of this tool.

The second **Bulk Add** tool works in exactly the same way, except that instead of using the stored auto-add regex of the group to which you wish to add members, it uses an *ad hoc* regex that you must supply via the form. This adds a further level of flexibility to the group system.

11.5 Privileges

Privileges represent things a user can do in CQPweb. There are multiple types of privilege, each controlling access to a different type of "thing" that can be done. The most important type is the corpus-access privilege, which determines which of the available corpora a user is allowed to use, and what they are allowed to do with that corpus.

This section first explains the different types of privilege, and then goes on to explain how you can create, monitor, and edit privileges in the CQPweb system.

11.5.1 Corpus access privileges

There are three levels of corpus access privilege. In order of ascending level of access, they are *normal*, *restricted*, and *full*. The latter two levels are defined relative to *normal*.

Normal access is the level of access that you would give to users who have all the necessary licences/IP rights to both use and reproduce the data. For example, if the corpus is one for which a licence must be paid, you would give normal access to any users known to have paid for a licence or to be affiliated to an organisation which has a licence. Alternatively, if the corpus is one that is openly accessible to all, you could grant normal access to *all* users on the system, via the **everybody** group.

When a user has normal-level access, they can do more or less everything. They can perform (and download) concordances without restriction, and they can use the extended-context function. These two functions open the possibility of users being able to access the complete underlying text of (one or more texts from) the corpus - by downloading, or copy-pasting, overlapping extended context chunks. By default, extended context can be expanded out to around 2,000 tokens, so it is in theory possible to extract the whole text of a corpus from the concordance of any evenly-distributed item with a frequency greater than about 0.5 per thousand tokens. This is why normal access should not normally be given to users who do not have the necessary IP rights to the corpus.

Restricted access implements certain limitations on what users can do that are intended to make it much harder, if not impossible, for them to be able to extract the complete underlying text from the CQPweb interface. The idea is that restricted access to a corpus can potentially be granted to users who don't have a licence - even if the corpus contains copyrighted material - because the restrictions stop them getting at any stretch of underlying text longer than concordance-length snippets, reproduction of which is more likely to fall under the "fair use" or "fair dealing" provision of copyright law.

- *Necessary disclaimer*: no one who works on CQPweb is a lawyer and nothing in this manual constitutes legal advice. The restricted access privilege is provided in the hope that you may find it useful as a *partial* tool for making sure you comply with whatever the copyright law is in your jurisdiction; but without any warranty whatsoever that it is either necessary or sufficient for that purpose. You use it at your own risk.

The restrictions implemented for that level of access are as follows:

- Users can't view extended context if they only have restricted access.
- Users are blocked from using the *Download tabulation* function.
- If a query has too many hits, it will automatically be thinned down to a random subset. (Too many = more than half the square root of the size of the corpus or subcorpus in tokens.)

To explain the last one: as noted above, if you do a concordance for something really common, and then download it, you could in theory reconstruct the whole corpus from the concordance lines - even without access to the extended context view. The "half the square root" limit tries to counteract this, while not making concordances in small corpora totally useless.

So, for instance, half the square root gets you 5,000 examples in a 100MW corpus, but 1,000 examples in a 1MW corpus. The limit is always rounded upwards to a whole number of thousands.

《*create table of some common sizes*》 **TODO**

The idea is that the random subset of examples will be spread out in the corpus so there will be gaps between them no matter how common the thing searched for is – so the underlying text can't be reconstructed.

Full access lets users access extra functions that involve total access to the data in one way or another. Currently, the only such function is the **Export corpus** tool, which allows the raw text of the corpus to be exported in plain-text format for analysis using other software.

A corpus access privilege is defined in terms of its *level* (restricted, normal or full, as per above) and its *scope* - where its scope is the corpus or set of corpora that it grants access to.

So, for instance, you might define a privilege granting *restricted* access to corpora A, B and C, and a second privilege allowing *normal* access just to corpus A. As explained further in section 11.6, CQPweb always uses the highest applicable privilege. Users granted both these privileges would therefore have normal access to A and restricted access to B and C.

Adding a new corpus to an existing scope is an easy way to apply the access pattern you have defined for one corpus to another. Continuing the example, if you a create a new corpus D which is governed by the same licensing/IP conditions as corpus A, you could simply add corpus D to the privilege which allows *normal* access to A. All users granted that privilege would then have access to corpus D without you needing to separately configure any grants of new privileges for corpus D.

11.5.2 Frequency list privileges

《*Explain these*》 **TODO**

《*Explain necessity of giving "everyone" at least one privilege of this sort*》 **TODO**

《*explain that the "scope" here is a number:*》 **TODO**

11.5.3 Extra runtime privileges

《*Explain these*》 **TODO**

《*explain that the "scope" here is a number:*》 **TODO**

11.5.4 Database privileges

《*Implement these; explain*》 **TODO**

11.5.5 File upload privileges

⟪*Implement these; explain*⟫ **TODO**

11.5.6 Upload-area filestore privileges

⟪*Implement these; explain*⟫ **TODO**

11.5.7 The CQP binary file privilege

Users who have this privilege are allowed to do two things that users without the privilege are not:

- Download binary-form CQP query files from saved queries
- Create a saved query by uploading a binary-form CQP query file

These two mirror-image functions are designed for advanced users who make extensive use of very large saved queries. Such users may exhaust their allowance of saved-query space. An easy way for them to clear out their disk space allowance is simply to export the binary files of the saved queries for external storage, and then reinsert those binary files if they want to use the queries in CQPweb later. This avoids extensive use of corpus-position dumps.

These functions should normally be restricted to advanced users simply because most users will not understand what CQP binary saved-query files are or how they work!

There is only one privilege of this type, and it is one of the default generated privileges. However, it is not initially granted to anyone, so the functions in question will only be available to superusers.

11.5.8 Corpus installation privileges

These privileges are required for users to be able to install their own corpus data on the server.

To allow users to install their own corpora, the first step is to enable the user-corpus system, by setting the `$user_corpora_enabled` configuration option to **`true`** (see 2.3.8).

This will only make the system *visible* to users. They will not be able to actually install corpora until their account has a privilege allowing them to.

Multiple privileges affect user-corpus installation:

- First, they must have a file-upload privilege (see above, 11.5.5).
- Second, they must have a disk-space privilege giving them sufficient space in their upload area to store the input files for corpus installation (see above, 11.5.6).
- Third, they must have the privilege to use at least one of the registered CorpusInstaller plugins.
- Fourth, they must have a disk-space privilege for user corpora.

The *third* privilege above governs how much data can be installed into a single corpus with a given installer plugin. The *fourth* governs the total amount of space taken up by all of the user's corpora (this gives the system administrator defence against some user or users taking up all the available disk space).

11.5.9 Creating and editing privileges

⟪*Make sure that the right links in the admin-ui chapter point here*⟫ **TODO**

⟪*explain the interface*⟫ **TODO**

⟪*Explain how privileges are presented: unique ID number, then a short prose explanation*⟫ **TODO**

⟪*Explain the edit interface*⟫ **TODO**

11.6 Grants: creating and managing grants of privileges

A *grant* is a link between some user, or group, and a privilege. The system works out what a user is and is not allowed to do based on the privileges granted to them.

A user has *both* any privileges that have been granted directly to them; *and* all the privileges that have been granted to *any one or more* of the groups of which they are a member.

Higher privileges override lower privileges. For instance, let us imagine there is a user A who is a member of group B, and we want to know what level of access this user has to a corpus C.

- If group B is granted restricted access to C, but user A is individually granted normal access to C, then the effect is that A has normal access to C: the grant to the user overrules the grant to the group.
- If group B is granted normal access to C, but user A is individually granted restricted access to C, then the effect is the same - A likewise has normal access to C: the grant to the group overrules the grant to the user.

The same is true with other types of privilege: when more than one applies, the one that has effect is the most expansive. This means it is always safe, and usually a good idea, to grant a lowest-common-denominator set of privileges to the "everybody" user group; users and groups with higher privileges will always benefit from those higher privileges, even though the users in question are (by definition) also members of "everybody".

When a user is removed from a group, they lose the privileges associated with that group, *unless*, of course, they are linked to that privilege in another way - individually, or by virtue of membership in another group that is granted that privilege. To put it another way, the transfer of grants from a group to its members is dynamic and ongoing, not static and persistent.

Finally, if a privilege is edited, then all users and groups who were granted the privilege pre-edit are *still* granted that privilege post-edit. (This makes it easy to give a new corpus the same access pattern as some existing corpus, by putting it within the scope of the privileges that govern the existing corpus. It is then not necessary to add any more grants on the system.)

There are two menu options in the Admin Control Panel which allow you to create, monitor, and delete grants: **CP >Users and privileges >Manage user grants** and **CP >Users and privileges >Manage group grants**.

These menu options lead to very similar screens. At the top is a form for making new grants. To grant a privilege to a user or group, select the user/group from the first dropdown; select the privilege from the second dropdown; and press the *Grant privilege...* button.

⟪*in future it will be possible to set a grant to expire on a specified date. But this is not yet implemented*⟫ **TODO**
⟪*by the present design, expired grants will just be deleted - they won't be preserved inactive. Or should they be?*⟫ **TODO**

Lower down you will find a list of existing grants. Press the [**x**] button next to any grant to delete it.

The **Manage group grants** screen has one extra function, which is the self-explanatory *Clone grants* tool.

Cloning grants from one group to another is useful if you have just created a new group that is going to have broadly similar privileges to an existing group: cloning the existing group's grants to the new group, and then editing those grants as necessary, can be quicker than granting each of the privileges to the new group one by one through the interface.

11.7 Running an open server

⟪*more*⟫ **TODO**

Even if all your corpora are completely open-access, it is often still a good idea to get users to sign up for individual accounts, so that they have access to their own private saved/categorised queries, their own query history, and so on.

⟪*more*⟫ **TODO**

⟪*note the gotcha of blog spam in the macros form*⟫ **TODO**

11.8 Access to frequency lists

⟪*section not written*⟫ **TODO**

⟪*following is text from an email to the CWB list explaining SOME of the issue*⟫ **TODO**

```
No, it's not the case that every corpus shows up - or at least, not for everyone; what you see
corpora incl. invisible because, I guess, you are superuser.

Here's the new system (as of 3.2.40 I think) in detail.

Currently, in both 3.2 and 3.3, the list of corpora/subcorpora offered as reference for keywor

    *       - Remainder                            (current corpus minus subcorpus selected
    *       - Subcorpora (local, owned)            (lus)
    *       - Granted subcorpora (local)           (lgs)
    *       - Public subcorpora (local, non-owned)    (lps)
    *       - Entire corpus                        (***)
    *       - System corpora (any access level)    (fsc)
    *       - User's own corpora.                  (fuc)
    *       - Granted corpora.                     (fgc)
    *       - User's nonlocal subcorpora           (fus)
    *       - Public subcorpora (nonlocal, nonowned)  (fps)
    *       - Granted subcorpora (nonlocal, nonowned) (fgs)

where "local" = part of the current corpus ; "owned" = installed by the user in the "user corp
think) ; "granted" = installed by another user who has given the current user access; summaris
in query-forms.php circa line 840ff in 3.3 and circa 940 ff. in 3.2) [*]

That is, as well as public subcorpora, users are able to use the freq list of any system or us
There's no such thing as a public corpus FL any more (and both the database field and UI to se
permission for.

This is part of a long-planned move away from the ancient all-or-nothing "public freqlist" mec
on the "corpus access privilege" system, which has at least some semblance of flexibility. "Pu

In future, the public FL feature will be removed completely once I work out how to handle subc

Other plans: a new privilege level, perhaps, IE "freqlist only access" or similar, which would
"public" function. Also in future, the list of corpora available for KW will be filtered by la
"undetermined".

Till then, subcorpus FL publicness is controlled just as it always was: through "cached freque
```

12 Using plugins

12.1 What is a plugin?

- A *plugin* is a small program written to operate within the framework of a larger system.
- Many applications offer a plugin framework, so that advanced users can add capabilities to the system which it does not possess out-of-the-box.
- CQPweb is such an application.
- A CQPweb plugin is a chunk of code added to the system, which is then accessed in predefined ways by CQPweb to provide extra capabilities for users.
- Some plugins are supplied with CQPweb. You can also write your own.
- This chapter explains some general things about plugins, and some specific things about the currently implemented types of plugin.
- The different types of plugin do not have much in common, they just represent a set of things that we thought users of CQPweb might want to have an easy way to customise!
- A plugin is a single PHP file, which contains a PHP *class* for the plugin.
- The class, and the PHP file, must have the same name as the plugin. For instance, **MySuperPlugin** should be in a file `MySuperPlugin.php` which must contain a PHP class called `MySuperPlugin`.
- To make use of a plugin, you must first put the code file into `lib/plugins` within the CQPweb folder.
- Only plugins at that location are recognised by the system (see 12.3.
- Some made-for-you plugins are already present under `lib/plugins/builtin`.
- To use your plugin, you must *register it* (see 12.4 to make CQPweb aware of it.
- For some plugins, you must also configure *privileges* (see 12.5) in order to be able to grant some or all of your users permission to use the plugin.
- Other plugins need to be *activated* for use with different corpora.
- All plugins can be given *extra configuration* data when they are activated by CQPweb.
- This configuration is specified when you register the plugin. A plugin can be written to make extensive use of extra configuration, or none at all - as its creator prefers.
- Extra configuration can be used to provide system-specific information that the plugin needs, or even to make the same plugin code behave in different ways depending on the extra configuration it was registered with; see 12.4.
- Different types of plugin have been added at different points in CQPweb's history; for a list, see 12.2.

12.2 Types of plugin

12.2.1 Annotators

An Annotator plugin is one that tags a file. Normally, this will be a case of interfacing with an external program such as a POS tagger or lemmatiser; but the plugin class could also be written to do the job itself.

Annotators should produce output in CWB vertical format - that is, p-attributes as columns, with XML tags for s-attributes on separate lines. That is because, when the user-corpus system is enabled, users can select Corpus Installers (see 12.2.5) to run over their uploaded files, and Corpus Installers are able to use Annotators. So the typical sequence would be:

- User uploads a text file or files;
- User selects a Corpus Installer and specifies their uploaded file(s) as input;
- Corpus Installer calls an Annotator to run over the specified files;
- Corpus Installer passes the output files (in vertical format), plus information on what attributes are used, back to CQPweb to set up the corpus.

However, ideally Annotator plugins should be written to work independently of this process; the Annotator should only "care" about the tagging of the text files, leaving the other details to the Corpus Installer plugin.

Annotators are a semi-exception to the general rule that plugins must be *registered* (see 12.4), because a Corpus Installer can be designed to use an Annotator regardless of whether it is registered with CQPweb or not. The StandardToolInstaller plugin (see 12.7.7) works this way; it uses either the TreeTagger or UcrelTagger Annotator plugin to tag input data, and can do so even if they are not registered - as long as they are in the `lib/plugins` directory.

To write an Annotator plugin, you need to write methods for (a) tagging files, and (b) describing the format of the resulting tagged file; see 12.6.3.2.

12.2.2 Format Checkers

Format Checker *plugins are currently not enabled.*

Format Checker plugins are used to check that data files are in a valid format. They will be accessible (a) to Annotator/Corpus Installer plugins, so they can check their input data; (b) to users and the system administrator via the upload area screens.

12.2.3 Script Switchers

Script Switcher *plugins are currently not enabled.*

Script Switcher plugins are used to transform the underlying data of the corpus in some way for display. The main purpose of this kind of plugin is for the rendering of non-Latin alphabet data in Latin. Script Switchers for widely used alphabets will be included.

12.2.4 Corpus Analysers

Corpus Analyser *plugins are currently not enabled.*

Corpus Analyser plugins allow the creation of custom tools for the *Analyse Corpus* interface.

12.2.5 Corpus Installers

Corpus Installer plugins act as the controllers for installing user corpora (for details on the user corpus system, see chapter 13).

When a *system* corpus is installed, all information about its p- and s-attributes, the input files, and so on are provided by the admin user. For *user* corpora it is different: to make things more usable and friendly, they only need to select a Corpus Installer to use. The plugin then (a) manages any necessary tagging, normally by handing the work off to an Annotator plugin; (b) provides to CQPweb the information necessary to install the corpus.

Users' ability to install their own corpora is determined by which Corpus Installers they have permission to use.

12.2.6 Postprocessors

***Postprocessor** plugins are currently not enabled.*

Postprocessor plugins implement *Custom Postprocesses* - methods by which a query can be postprocessed.

A Postprocess is any procedure which changes a query result after the query is initially run. The native postprocesses in CQPweb are things like the Randomise, Sort, and Thin functions. Producing a reduced set of hits by clicking on a specific item in the Distribution, Frequency Breakdown or Collocation Displays, also counts as a postprocess, as does splitting up a Categorised Query.

The header bar of the concordance display includes a record of all the postprocesses that have run on a query since it was originally produced by an initial from-scratch search.

When you install and activate a Postprocessor, it will appear on the dropdown menu in the Concordance display just like the built-in options. It can then be applied to any query. It is up to you to write the internals of the Custom Postprocess so that what it does makes sense!

12.2.7 Query Analyser

***Query Analyser** plugins are currently not enabled.*

Query Analyser plugins allow custom tools to be added (alongside distribution, collocation, etc.) that perform and present some kind of analysis of the results of a query. (Contrast Custom Postprocesses, which *alter* the data of an existing query.)

12.2.8 Query Downloader

Query Downloader plugins define default (one-click) concordance download modes.

The *Download Concordance* display (accessed from the Concordance screen of any query) gives the user numerous options for the format of their download, including *Size of context* and *Format of output - KWIC or line.*

Alternatively, predefined download modes can be accessed via one-click buttons which appear at the top of the display. In older versions of CQPweb, the system provided two one-click download modes:

- Download with typical settings for FileMaker Pro
- Download with typical settings for copy-paste into Word, Excel etc.

As of CQPweb version 3.3.0, the one-click download modes are instead generated by Query Downloader plugins. One button will appear per registered and activated Query Downloader. Builtin plugins are provided which replicate the behaviour of the older system one-click modes, but using them is now optional.

Query Downloaders can work either by providing a bundle of settings to the normal Download mechanism, or by implementing an internal mechanism to process raw concordance lines generated by CQP into the format that will be downloaded.

12.2.9 CEQL Extender

CEQL Extender *plugins are currently not enabled.*

CEQL Extender plugins allow the definition of custom CEQL-based query grammars that can be made available as extra query modes alongside unextended CEQL (Simple Query mode) and CQP syntax.

12.3 Installing plugins

Plugin files are installed by placing the PHP file in the `lib/plugins` directory. By default there are no files at all in this directory - so, by default, CQPweb doesn't have access to any plugins.

CQPweb possesses a number of builtin plugins (see 12.7). Some of these implement functions which, we anticipate, will be broadly useful on lots of installations. Others are simply included as examples of how to create and work with a plugin of that type. The builtin plugins can be found inside a directory tree under `lib/plugins`.

The directory `lib/plugins/builtin` has one subdirectory for each plugin type. At present, there do not exist examples of every type, although in future there will be.

The builtin plugins are invisible to CQPweb, as CQPweb only looks for plugins in `lib/plugins` - not in any subdirectories. To use a builtin plugin, you must first place it into that directory. While you can do this by actually moving the file, the intended usage is to create *links* in `lib/plugins` to the builtins' true location.

On Unix-like systems, this would typically be done as follows:

- `cd /path/to/cqpweb/lib/plugins`
- `ln -s builtin/Annotator/BasicTokeniser.php`

This creates a link in `lib/plugins` to the BasicTokeniser plugin's code file. BasicTokeniser will now be visible to CQPweb.

A parallel directory tree to `lib/plugins/builtin`, `lib/plugins/local`, also with one subdirectory per plugin type, is provided for convenience for you to store your own plugin files - it will always be empty. Of course, you don't have to use it! You can put your own plugin files anywhere and still put links to them in `lib/plugins` (but the actual location of the file needs to be accessible to the web daemon).

Once you have added your plugin file to the plugin directory, CQPweb can find it. But in order to make it active in the system, you need to *register* it.

12.4 Registering plugins

Once a plugin is installed, you must *register* it, so that CQPweb is aware of it, and can activate it where necessary.

The plugin registry contains a list of all the plugins in use. Each registered plugin is linked to a particular set of configuration data. This means that you can register the same plugin file multiple times, with different configuration data - and the system will perceive it as a number of different plugins.

This is useful for plugins like the builtin StandardToolInstaller plugin (see 12.7.7), which can be used to interface CQPweb to widely used annotation systems, including at present the UCREL CLAWS/USAS taggers for English and Helmut Schmid's multilingual TreeTagger, although more may be added in future. StandardToolInstaller needs to be configured (1) regarding which tagger to use, (2) if using TreeTagger, regarding which language to tag. It's therefore possible - in fact, recommended - to register the StandardToolInstaller multiple times, with different language settings. CQPweb will then contain multiple different instances of the StandardToolInstaller, each of which can be managed separately.

The plugin registry is managed through the Admin Control Panel. Go to **CP >Plugins >Manage Plugins**. At the top of this screen is a form you can use to register a new plugin. You must specify three things:

- What PHP file contains the plugin class.
- A short description of the plugin (mostly for mnemonic purposes; users won't see it)
- Configuration data for the plugin, consisting of a set of zero or more key-value pairs.

The configuration data, if any, is passed to the constructor method of the plugin class. The available configuration settings are specific to each plugin, so refer to the documentation on the plugin in question to know what is possible.

Below the *Register a new plugin* form is a table with details of all currently registered plugins. You can delete registry entries here; deleting a registry entry **does not** delete the plugin's PHP file.

Some types of plugin create tools or options inside the interface to a particular corpus, for instance, Postprocessor and Query Downloader plugins. These must be *activated* for each corpus where you wish to make them available. The interface for plugin/corpus activation is accessible via a link from the table of registered plugins. This interface allows you to control what level of access to the corpus a user must have for the plugin to be available to them by reference to the corpus access privileges. Thus, these kinds of plugins do not need to be accompanied by plugin-use privileges.

12.5 Permissions for plugins

Once you have created, installed and registered your plugins, you need to create privileges to control the use of those plugins. Plugin privileges grant access to particular entries in the plugin registry. If a given plugin is registered twice with different options, the two versions may be referenced separately by privileges. This gives you a fine degree of control over what each user or group is allowed to do with a particular plugin.

If a user has not been granted a privilege permitting them to use a particular plugin, it will appear to that user as if the plugin does not even exist.

Because plugins affect different parts of the system, the kinds of privileges required vary somewhat.

- **Annotator** plugins are currently only used in corpus installation. For that reason, access to them is mediated by Corpus Installer plugins. The Annotator itself need not be registered. Corpus Installers can be told what Annotator to use as part of their extra configuration, and likewise can pass through extra configuration items to their Annotator where necessary.
- ***Format Checker*** *plugins are currently not enabled.* *《say something about FormatChecker permissions》* **TODO**
- ***Script Switcher*** *plugins are currently not enabled.* **Script Switcher** plugins need to be activated for given corpora; once activated, access is controlled by the normal corpus access privileges.
- ***Corpus Analyser*** *plugins are currently not enabled.* **Corpus Analyser** plugins need to be activated for given corpora; once activated, access is controlled by the normal corpus access privileges.
- To use **Corpus Installer** plugins, a user must have at least one of each of four different kinds of privilege. This is discussed in the chapter on privileges (11.5.8).
- ***Postprocessor*** *plugins are currently not enabled.* **Postprocessor** plugins need to be activated for given corpora; once activated, access is controlled by the normal corpus access privileges.
- ***Query Analyser*** *plugins are currently not enabled.* **Query Analyser** plugins need to be activated for given corpora; once activated, access is controlled by the normal corpus access privileges.
- **Query Downloader** plugins need to be activated for given corpora; once activated, access is controlled by the normal corpus access privileges.
- ***CEQL Extender*** *plugins are currently not enabled.* **CEQL Extender** plugins need to be activated for given corpora; once activated, access is controlled by the normal corpus access privileges.

12.6 Creating plugins

12.6.1 Introduction to writing plugins

To write a plugin, you will need a reasonable knowledge of programming in general, and at least a basic acquaintance with PHP specifically. It's beyond the scope of this manual to explain programming/PHP from the ground up; the PHP programming language is extensively documented at `http://php.net`.

A plugin takes the form of a PHP class (see `http://php.net/class`) that performs one or more defined tasks. For example, a *Custom Postprocess* plugin takes a defined CQPweb query and changes it ("postprocesses" it) in a certain way.

Each time you create a plugin, you should place it in a single file which has the same name as the plugin itself. So, for instance, to create a plugin called **MyPlugin** you should create the PHP file `MyPlugin.php`. You should then put that file into the `plugins` subdirectory of the `lib` directory (see 12.3, or else put a link to the code file in that subdirectory. Like all CQPweb code files, this new file/link must be readable by the username your webserver runs under.

The file will normally contain nothing except the class that represents your plugin. You *can* add extra functions/classes that will be called by your plugin, but this is unliekly to be useful except for the most complex plugins (or for CEQL Extenders).

Each type of plugin has a separate PHP *interface* and your class must implement that interface to be recognised as a plugin. *Implementing an interface* in PHP, as in some other object-oriented systems, means that the class must have methods that meet certain defined signatures. These are explained below. If your plugin does not meet the definition of the interface it implements, things will stop working.

The interface that your class needs to implement depends on what type of plugin you are writing:

Type of plugin	**Interface to implement**	**Symbolic constant**
Annotator	`Annotator`	`PLUGIN_TYPE_ANNOTATOR`
Format Checker	`FormatChecker`	`PLUGIN_TYPE_FORMATCHECKER`
Script Switcher	`ScriptSwitcher`	`PLUGIN_TYPE_SCRIPTSWITCHER`
Corpus Analyser	`CorpusAnalyser`	`PLUGIN_TYPE_CORPUSANALYSER`
Corpus Installer	`CorpusInstaller`	`PLUGIN_TYPE_CORPUSINSTALLER`
Postprocessor	`Postprocessor`	`PLUGIN_TYPE_POSTPROCESSOR`
Query Analyser	`QueryAnalyser`	`PLUGIN_TYPE_QUERYANALYSER`
Query Downloader	`QueryDownloader`	`PLUGIN_TYPE_QUERYDOWNLOADER`
CEQL Extender	`CeqlExtender`	`PLUGIN_TYPE_CEQLEXTENDER`

So the overall shape of the plugin file will be like this:

```
<?php
class MyPlugin implements Postprocessor
{
    // you will need to add the methods here...
    // plus any member variables you want to use
}
```

If you wish to make use of a base class (see 12.6.4 for an explanation of base classes), then the class declaration should be:

```
class MyPlugin extends PostprocessorBase implements Postprocessor
[...]
```

The interfaces and base classes themselves are defined in the file `lib/plugin-lib.php`. The definitions are heavily commented, and the text here is based in part on those comments; it's highly recommended to take a look at this code file if you are starting out writing a plugin. In the case of any discrepancy between the information given here and the actual code, the actual code is correct (and the information here is either incomplete or awaiting an update).

There should be no whitespace before the file's leading `<?php` delimiter. If there is, CQPweb may stop working properly.

The builtin plugins provided with CQPweb provide examples of how plugins can be written.

12.6.2 Naming your plugin

All plugin names must be legal PHP class names (see `http://php.net/class`). They share a namespace with the internal CQPweb classes, so try to make sure you don't clash (PHP will crash if you do). To be safe, always prefix your plugin classes with a unique element (e.g. your name). Also, class names beginning with `My_` are guaranteed to be safe.

12.6.3 Methods your plugin must implement

This section describes each of the methods that you must implement to fulfil the demands of the interface for each type of plugin. You can, of course, have other methods if you want, but nothing outside the plugin will call them.

It is quite possible, and indeed expected, that in many cases you will want the major work of the plugin to be done outside PHP - for example, by a command-line utility or by a Perl or Python script. In which case, the PHP methods will be a thin wrapper round a call to an external system. CQPweb does not care about this, as long as it can use the methods in the interfaces to get the information it needs!

Note the documentation here is largely derived from the within-code documentation in `lib/plugin-lib.php`. If the comments there are different from what you find here, then the former should be considered definitive as they may be more recently updated or more extensive.

12.6.3.1 Methods required by every type of plugin These methods are part of the `CQPwebPlugin` interface, which is the parent of the interfaces for all of the specific types of plugin.

- `public function __construct($extra_config = []);`

This method initialises the plugin. The `$extra_config` is an array of key-value pairs, taken from the plugin's entry in the plugin registry (see ⟪*crossref*⟫). This allows you to define multiple entries in the plugin registry from the same class, but with different configuration (for instance, the built-in Annotator plugin for the TreeTagger needs to be told what language to tag in). What the plugin *does* with these variables is determined by the code of the `__construct` method. **TODO**

- `public function description();`

This method should return a string containing the title or short description of this plugin. This should be relatively short, and not contain any HTML.

- `public function long_description($html = true);`

This method should return a string describing the plugin. This may (but need not) include HTML formatting, where requested; and may either be the same as the string returned by description() or be longer. Line breaks (`"\n"`) in the string may be rendered as HTML line breaks in some contexts, unless HTML is requested.

The method should respect the `$html` parameter by not including any HTML code in the string that is returned if this parameter is **false**.

- `public function status_ok();`

This method should return boolean **true** if the plugin has not encountered any error conditions, or boolean **false** if one or more error conditions has been encountered. If this method returns **false**, something should be readable via the `error_desc()` method.

- `public function error_desc($html = true);`

This method should return a string describing the last encountered error.

If there has been no error, then it can return an empty string, or a message saying there has been no error. It doesn't matter which.

The method should respect the `$html` parameter by not including any HTML code in the string that is returned if this parameter is **false**.

12.6.3.2 Methods required by Annotator plugins

- `public function process_file($path_to_input_file, $path_to_output_file);`

Calling this method should tag the file specified as `$path_to_input_file`, placing the output at `$path_to_output_file`.

Both arguments should be relative or absolute paths. The method **SHOULD NOT** use CQPweb global variables. The input file **MUST NOT** be modified.

This function should return **false** if the output file was not successfully created. If the output file is partially created or created with errors, it should be deleted before **false** is returned.

If all goes well, the method should return **true**.

- `public function process_file_batch($input_paths, $path_to_output_file);`

This method should process a group of input files, placing the output into a single file. All the comments for `process_file()` apply here too. The return value should be boolean **true** or **false**, just as for that method.

The `$input_paths` parameter will be an array of strings, where each string is the path to an input file.

- `public function output_size();`

This method should return the size of the last output file created as an integer count of bytes, or zero if no file has yet been specified.

12.6.3.3 Methods required by Corpus Installer plugins

- `set_max_input_tokens($max);`

Corpus Installers are controlled by permissions that specify how much text the user is allowed to index at one time. CQPweb will use this method to tell the Corpus Installer what the maximum size is that it should allow. The plugin should implement this method so as to store this value, and utilise it at the appropriate time (usually: after tagging, before indexing).

The parameter is a count of tokens; if zero or a negative limit is set, no restriction at all should be applied.

- `public function set_corpus_name($name);`

CQPweb will call this method to pass in to the Corpus Installer plugin the name (lowercase CQPweb handle) of the corpus it is begin used to create. The method should store this value. The name of the corpus is needed to create some of the corpus-installation setup SQL statements.

- `public function add_input_file($path);`

This method must add the file at the path provided to its list of files to be used as input for installation. (The Corpus Installer plugin may pass these through to an Annotator, or else use them directly.)

The method should also accept an array of strings - all representing paths to input files.

- public function do_setup();

```
/**
 * Run setup - that is,  anything that needs to be done to get files
 * ready to be encoded. This might include tagging, or even building a corpus.
 *
 * @return bool    True if setup worked OK; false if not.
 */
```

- public function do_cleanup($delete_input_files = false);

```
/**
* Run cleanup, e.g., deleting temporary files, if any.
* @param bool $delete_input_files  If true, the files specified using
*                                  CorpusInstaller::add_input_file() will be deleted.
*/
```

- public function get_charset();

This method should return the CWB string indicator for the character encoding of the corpus text. (These days, usually "utf8".)

- public function get_p_attribute_info();

```
/**
  * Gets information about the p-attributes for cwb encoding
  * (an array of strings for use with -P with cwb-encode).
  * @return array
  */
```

- public function get_s_attribute_info();

```
/**
  * Gets information about the s-attributes for cwb encoding
  * (an array of strings for use with -S with cwb-encode).
  */
```

- public function get_annotation_bindings();

```
/**
  * Gets the annotation bindings that will be used in the created
  * corpus (for Simple Queries).
  * @return array     A hash with one or more keys referring to
  *                   'special' syntax in CEQL, each mapping to
  *                   the p-attribute that will be searched.
  *                   For instance, 'primary_annotation'=>'pos'.
  *                   The "tertiary_annotation_tablehandle"
  *                   is the only non-p-attribute binding; it
  *                   must be for a table that actually exists
  *                   on the system, if not, declare_maptable()
  *                   can be used.
  */
```

- `public function declare_maptable();`

```
/**
  * Hash of simple-pos to p-attribute regex. CQPweb will add it
  * as a mapping table to the system. If you don't want to bother
  * with this, just have an empty function (which is what the
  * CorpusInstallerBase does).
  *
  * @return array
  */
```

《*change code comments to writeup!*》 **TODO**

12.6.4 Methods you can inherit

The plugin system includes "base classes" which contain implementations of several important tasks for various types of plugin. If you write your plugin to inherit from such a base class, then you do not have to write code for these tasks yourself. **It is highly advisable to make use of these base classes**. (But if you don't want to, you don't have to, even if you inherit from a base class: any method in a base class can be overridden in a child class simply by creating a method of the same name.)

The base classes all have the same name as the corresponding interface with the addition of `Base` at the end. The facilities they provide are listed below.

12.6.4.1 CQPwebPluginBase `CQPwebPluginBase` is the parent of the other base classes. Therefore, its methods are passed on to those base classes. This base class provides...

- A default `__construct()` method, which simply stores each key-value pair from the `$extra_config` as an object variable; the variable name is the key string, the variable value is the value. So if this method is passed, say, the pair `"action_type" => "x"` as part of `$extra_config`, then other methods will be able to refer to the object member variable `$this->action_type`, whose initial value is "x".
- A default implementation for the `long_description()` method (making it return the same as `description()`, which all plugins must still provide).
- A default error reporting system, which provides implementations for `status_ok()` and `error_desc()` as well as an extra method, `raise_error()`, which sets the error state (and logs the message passed as its sole parameter to be returned by any subsequent call to `error_desc()`). `raise_error()` can only be called from *within* a child class; examples of its usage can be found in various builtin plugins.

12.6.4.2 AnnotatorBase This base class provides...

- A text ID generation system. All CQPweb input data must have `<text>` tags with `id="..."` attributes. Annotators generally need to add these, as they are likely not to be present in the input, or to have been removed by the tagger. The AnnotatorBase provides multiple ways to generate an ID for each input text, that can then be incorporated into the output. The function to access this is `AnnotatorBase::get_next_text_id()`.

- A pair of support functions, `validate_read_paths()` and `validate_write_paths()`, each of which checks an array of file paths to make sure they are, respectively, readable/writeable.
- A simple implementation of `output_size()` which returns the value of an internal variable, `$bytes_in_output`. Child classes which wish to use this need to actually set that class property (as `$this->bytes_in_output`) when object instances are created.
- Null implementations of `output_annotation_list()` and `output_xml()`, which always return **false**, allowing child classes to not bother implementing these methods.

12.6.4.3 FormatCheckerBase

This base class provides...

《*Write this section when this plugin type is enabled.*》 **TODO**

12.6.4.4 ScriptSwitcherBase

This base class provides...

《*Write this section when this plugin type is enabled.*》 **TODO**

12.6.4.5 CorpusAnalyserBase

This base class provides...

《*Write this section when this plugin type is enabled.*》 **TODO**

12.6.4.6 CorpusInstallerBase

This base class provides...

- Multiple systems for creation of the necessary SQL statements / CWB declarations for annotations and XML (p-attributes and s-attributes).
  - Via `declare_annotation()` and `declare_xml()`: methods that the child class can use to pass in information about the attributes.
  - Via `declare_annotations_from_template()` and `declare_xml_from_template()`: methods which can be passed template ID numbers, from which the information will be taken.
  - Via `declare_content_from_annotator()`: a method to which a child class can pass its Annotator object, whose interface will then be interrogated to get information about the attributes.
- A default system for setting, and then retrieving, CEQL bindings: the `set_binding()` method allows them to be set, and the implementation of `get_annotation_bindings()` returns them.
- Default implementations of the following methods, which will work alongside any of the different systems mentioned above, and report the declared structures to CQPweb:
  - `set_max_input_tokens()` - stores a token limit in an internal variable, which the child class can use, or which can be applied with `restrict_input_data()` (see below).
  - `set_corpus_name()` (plus an internal utility function, `check_for_corpus_name()`, to make sure it has been called)
  - `get_sql_for_corpus()`
  - `get_sql_for_abort()`
  - `get_p_attribute_info()`
  - `get_s_attribute_info()`
  - `get_infile_info()`

  - `get_charset()` - returns the internal variable `charset`; the child class needs to set this
  - `get_xml_datatype_check_needed()` - returns the internal variable `xml_datatype_check_needed`, which is **false** by default; the child class can set this to **true** if desired.

- Default implementations of the following additional methods:
  - `add_input_file()` - for use alongside `get_infile_info()`
  - `do_cleanup()` - which deletes temporary files (and does nothing else).
- A utility function, `restrict_input_data()`, designed to be called by `do_setup()`, which makes sure the input files do not exceed the limit set by `set_max_input_tokens()`
- An alternative to `set_max_input_tokens()`, the method `set_restriction_from_privilege()`, which can be passed a privilege object, from which the object will extract a token limit.
- A null implementation of `declare_maptable()`, allowing child classes to not bother implementing this method.

12.6.4.7 PostprocessorBase This base class provides...

《*Write this section when this plugin type is enabled.*》 **TODO**

12.6.4.8 QueryAnalyserBase This base class provides...

《*Write this section when this plugin type is enabled.*》 **TODO**

12.6.4.9 QueryDownloaderBase This base class provides...

《*Write this section when this plugin type is enabled.*》 **TODO**

12.6.4.10 CeqlExtenderBase This base class provides...

《*Write this section when this plugin type is enabled.*》 **TODO**

12.6.5 An API for plugin writers

When you write a plugin, the whole of CQPweb's internal function library is theoretically available for you to call. However, unless you *really* know what you are doing, it is not recommended to just start calling functions all over the place. Something may go wrong.

That said, there are clearly bits of information that you might need inside a plugin that are not provided via the methods' parameters. For instance, in a Custom Postprocess where you need to keep or reject each concordance hit, it would be nice if you could find out what each concordance hit actually contains! You could do this by accessing CQPweb's interaction layer with the CQP backend for yourself, but that's prone to all those problems discussed above

So CQPweb provides a set of helper functions that you can call to access other bits of info in such a way that it's "guaranteed" not to mess things up (barring bugs we don't know about yet). There are currently three such functions (not tested as of the current version), all designed to be of use in writing Custom Postprocesses. Their function prototypes and internal documentation are provided below.

```
/*
 * Returns a path that can be used as a temporary filename by a plugin.
 * The plugin is responsible for making sure it gets deleted.
 * (It will be pre-created as a temporary file.)
 */
function pluginhelper_get_temp_file_path();

/**
* Gets a full concordance from a set of matches.
*
* The concordance is returned as an array of arrays. The outer array contains
* as many members as the $matches argument, in corresponding order. Each inner array
* represents one hit, and corresponds to a single group of two-to-four integers.
* Moreover, each inner array contains three members (all strings): the context
* before, the context after, and the hit itself.
*
* The $matches array is an array of arrays of integers or integers as strings,
* in the same format used to convey a query to a custom postprocess.
*
* You can specify what p-attributes and s-attributes you wish to be displayed in the
* concordance. The default is to show words only, and no XML. Use an array of strings
* to specify the attributes you want shown in each case.
*
* You can also specify how much context is to be shown, and the unit it should be
* measured in. The default is ten words.
*
* Individual tokens in the concordance are rendered using slashes to delimit the
* different annotations.
*/
function pphelper_get_concordance($matches,
                                  $p_atts_to_show = 'word',
                                  $s_atts_to_show = '',
                                  $context_n = 10,
                                  $context_units = 'words'
                                  );

/**
* Determines whether or not the specified corpus position (integer index) occurs
* within an instance of the specified structural attribute (XML element).
*
* Returns a boolean (true or false, or NULL in case of error).
*/
function pphelper_cpos_within_structure($cpos, $struc_attribute);

/**
* Gets the value of a given positional-attribute (word annotation)
* at a given token position in the active corpus.
*
* Returns a single string, or false in case of error.
*/
function pphelper_cpos_get_attribute($cpos, $attribute);
```

12.7 Builtin plugins

Some plugins are provided with the CQPweb distribution. However, they will not be available in the web interface until you add them to the system, as explained in section 12.4.

There are two types of builtin plugin: those supplied purely as examples of how to write a plugin, and those supplied because they are expected to be generally useful to lots of different users. All can be found in the `lib/plugins` directory.

But beware! These plugins may be under development (or may relate to types of plugin whose integration into CQPweb is not yet complete).

12.7.1 DeleteEveryThirdHit

DeleteEveryThirdHit is an example of a custom postprocess, to illustrate how they are written. It is not a postprocess you would ever actually want to use, because its function (thinning a query) is implemented better as one of CQPweb's internal postprocesses.

Extra configuration:

- None.

12.7.2 BasicTokeniser

This is a very simple, but usable, Annotator. It processes input files to produce tokenised output in CWB vertical format. It will only work with data in languages that use whitespace as a token delimiter (i.e. *not* Chinese, Thai, etc.)

Extra configuration:

- None.

12.7.3 TreeTagger

This Annotator is designed to interface with the TreeTagger software (see `http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/`).

It makes certain assumptions about how the TreeTagger installation files are to be found on the system; if your TreeTagger installation is laid out differently, then you might not be able to use this plugin.

The class contains configuration information for a large number of the parameter files available on the TreeTagger website for various languages. Again, if you haven't downloaded the parameters for a given language, you won't be able to use the plugin to tag files in that language.

Extra configuration:

- `tt_no_s_tags` - Boolean, if **true**, sentence tags (`s`) will not be added to the TreeTagger output.
- `tt_show_unknown_lemma` - Boolean, if **true**, words whose lemma is unknown will be tagged as `<unknown>`, rather than supplying the wordform as "its own lemma". The default behaviour of the plugin is for both this and the previous setting to be **false**. That is contrary to the default TreeTagger behaviour, but is the normal way to use TreeTagger with CQPweb.

- `tt_bin_path` - string, path to the TreeTagger installation folder (i.e. the directory which contains the `bin`, `cmd`, and `lib` subdirectories).
- `language` - string, sets the language to tag in. The valid language labels are abbreviated from the descriptions of the corresponding parameter sets as distributed via the TreeTagger website. Possible languages are those used as keys in the LANG_INFO array (which can be found at the bottom of the TreeTagger class; a utility function (static class method) is provided to check language-identifier strings (`TreeTagger::is_valid_language()`).

12.7.4 UcrelTagger

This Annotator is a very thin wrapper around the Lancaster University UCREL research centre's dual taggers, namely CLAWS (part of speech) and USAS (lemmata and semantic tags), via a controller script that generates CQPweb-ready output. It is only of use to you if you are already running CLAWS on your machine.

See:

- http://ucrel.lancs.ac.uk/claws/
- http://ucrel.lancs.ac.uk/usas/

Extra configuration:

- Many possible items, not documented here (yet).

12.7.5 BasicVrtInstaller

This installer assumes input files that are already in vertical format (and therefore do not need to be tagged).

Extra configuration:

- Does not make use of any extra configuration values.

12.7.6 SimplePlaintextInstaller

This is a simple corpus installation manager assuming plain-text input files. It calls a configurable Annotator in a standard way.

Extra configuration:

- `annotator_plugin_id` - integer ID for the plugin registry entry of the Annotator plugin to use to tag the files.
- `vrt_file_path` - (optional) path specifying where to place the output file; if not given, a temporary location will be used.

12.7.7 StandardToolInstaller

This is a corpus installer which uses the UcrelTagger and TreeTagger Annotators.

Extra configuration:

- `tool` - a string ("UCREL" or "TreeTagger") to tell the plugin which Annotator to use.
- `language` - a string that will be passed through to the TreeTagger plugin (see above).
- `semtag-resources` - a string that will be passed through to the UcrelTagger plugin (see above).
- `annotation_template_id` - integer ID of the annotation template describing the tagger output.
- `xml_template_id` - integer ID of the annotation template describing the tagger output.

The last two of these are optional. If they are not supplied, the plugin will find the templates itself (this will work fine as long as you have installed the default annotation and XML templates; see 6.7, 6.9).

13 User corpora

13.1 Introduction

《*write full chapter.*》 **TODO**

14 Extensible CQPweb

14.1 Introduction

This chapter explores the various ways of extending CQPweb by adding your own code to support features that it doesn't possess out-of-the-box.

14.2 Extending CQPweb with non-PHP modules

14.2.1 Overview

CQPweb is written in PHP. However, it is possible to write an extension for CQPweb in some other programming language, and then connect it to CQPweb using builtin tools for this purpose.

This is useful, for instance, with plugins. Plugins must be written in PHP, but it is quite possible to write a plugin which does nothing but delegate its task to a module in Python or R or Perl, and then interpret the results into the for that CQPweb expects. 《*add xref for "plugins" to the above*》 **TODO**

《*more*》 **TODO**

14.2.2 Perl

《*explain how to use it.* 》 **TODO**

14.2.3 Python

《*explain the PyFace interface.* 》 **TODO**

14.2.4 R

《*explain the RFace interface*》 **TODO**

《*add node js as nodeface? people seem to like it.*》 **TODO**

14.3 CQPweb Applications

《*explain them*》 **TODO**

15 Using the CQPweb API

15.1 Introduction

Use of the CQPweb API is, strictly speaking, not a system administration topic. However, effective use of the API requires some detailed knowledge of how the system works, and so it is convenient to include the topic in this manual.

The API was added, in rudimentary form, in version 3.2.32 and gradually refined through to versions 3.2.42 and 3.3.0+.

15.2 Structure of API HTTP requests

The API is accessed via a single entry point, the script `api.php`. All requests - even those which are associated with specific CQPweb URLs such as `concordance.php` - go via this entry point.

To use the API, you need to construct an HTTP request which specifies the function you want to call and the values of the arguments you wish to pass.

Requests to the API can be either GET or POST requests; the outcome is the same either way.

The URL for the request should be the standard URL for the corpus you wish to access, represented below by the placeholder ***$corpus***, plus `api.php`. In the remainder of this chapter, this URL will be assumed to have the following form:

```
http://your.server.net/path/to/cqpweb/$corpus/api.php
```

Instead of a corpus handle, you can use the paths

```
http://your.server.net/path/to/cqpweb/usr/api.php
```

or

```
http://your.server.net/path/to/cqpweb/exe/api.php
```

for function calls that don't relate to any particular corpus. (For such functions, specifying a corpus is not *wrong*; it just has no effect.)

Finally, for user corpora, the path to the corpus involves additional levels:

```
http://your.server.net/path/to/cqpweb/usr/61/anybody/_00128/api.php
```

... as per usual.

References to `api.php` in the remainder of this chapter should be interpreted as shorthand for the appropriate URL as described above.

15.3 Calling a function

The function you wish to call should be specified to the API as the HTTP parameter `f`.

Using HTTP GET, this is specified in the URL as follows:

- `api.php?f=`*`name_of_function`*

The parameters of the function (if it has any) are also encoded as HTTP parameters, with the parameter name being the same in both the API definition and the HTTP request.

While the function definitions present the parameters in a particular order, this is a convenience for the reader. In fact, the function parameters can be added to the URL in any order (as can even the **f** parameter).

The URL for a function call with three arguments specified might, then, look like this:

- `api.php?f=`*`name_of_function`*`&arg1=`*`something`*`&arg3=`*`something+else`*`&arg2=`*`23%2A`*

Using HTTP POST, the request would look something like this overall:

```
POST /path/to/cqpweb/$corpus/api.php HTTP/1.1
Host: your.server.net
Content-Type: application/x-www-form-urlencoded

f=name_of_function&arg1=something&arg3=something+else&arg2=23%2A
```

15.4 Parameters and types

Parameters are all transmitted as strings - because HTTP parameter values are, fundamentally, strings.

However, upon reaching the server, parameters are treated as being typed: the type determines how CQPweb will treat them.

So, for instance, integer arguments (`int`) are transmitted as strings, but then are always typecast to integer by CQPweb before being used.

The following pseudo-types are used in describing the API parameters and return values:

void Indicates *no parameters* or *no return value* (i.e. content of the response object will be `null`).

bool Boolean; can be true or false only (encoded as “1” and “0” respectively).

int Integer (encoded as decimal, with no thousand-separators).

float Floating-point number (encoded in one of the forms that PHP can typecast from string to float, using the standard C `strtod()` function).

string String data; should be URL-encoded.

array Numerically indexed array of any number of identically-typed values. The API does not use any array parameters, but functions may return arrays.

object Associative array or hash (which, in JavaScript, is the same thing as an object). As with arrays, this is a return type, not a parameter type.

In section 15.7, every function is given a *signature*, that is a specification of its return type and parameter names and types.

For example, this is the signature of the function `log_in`:

- *void* log_in(*string* username, *string* password [, *bool* persist])

The return type is given before the function name.

After the function name, within brackets is a list of parameters. Each one is given as a type, followed by the name of the parameter. Commas separate the different parameters.

Compulsory parameters are given first. Optional parameters follow. The set of optional parameters is enclosed in square brackets. If there are no parameters at all, the single word *void* will be given inside the function's brackets.

So, the signature above indicates a function that has no return value (i.e. it will send back `null`), and must be called with two compulsory string arguments, with the names *username* and *password.* There is a third argument, but it is optional; its type is Boolean.

The discussion of `log_in` in section 15.7.5 expands on the summary that the function signature provides.

15.5 Structure of API HTTP responses

The response from the API entry point will always be a single object, encoded in JavaScript Object Notation (JSON).

This object has four possible member variables:

- `status` - string; simple status flag, either `ok` or `error`.
- `errno` - status code describing the exact problem.
- `content` - a value of any kind representable in JSON, this is the return value from the API function that was called. It may be absent in case of error.
- `errors` - array of strings (debug and error messages).

The procedure on receipt of this response object should be as follows:

- Check that its status flag is set to `ok`.
- If yes: take the function call's return value from `content` and proceed.
- If no: examine the error number and error message array to try to determine what went wrong, and at what point.

The type of the return value is specified per function. It could be a single value (string, integer, floating-point, Boolean) or a complex value (an array or an object).

JSON does not have a concept of associative arrays (hashes/dictionaries) distinct from objects, which is why the CQPweb API does not distinguish objects from associative arrays in its return types. Any return values of type object (unless otherwise specified in the documentation below) are derived from

PHP objects of class **stdClass** (the ultimate ancestor of all PHP classes, and default class of objects with no specified class).

Likewise, JSON does not distinguish integers from floating-point numbers, using a single floating-point type for both; so, when the return type is specified as an integer, what this really means is that it is a floating-point number *converted from a PHP integer*.

JSON `null` is used for the `content` member variable if the function has no return value (i.e. its return is of type `void`).

15.6 Logging in and use of login tokens

In order to use a CQPweb server, you must be logged in as some user. The set of corpora that you can access via the API will be the same as the set of corpora that your user account can access via the browser.

⟪*more*⟫ **TODO**

15.7 Available functions

The following functions can currently be called in the API.

15.7.1 *string* get_version(*void*)

Find out what version of CQPweb the server is running.

- **Returns**: String in the form "major.minor.increment"

15.7.2 *string* get_cwb_version(*void*)

Find out what the version of the CWB core the server is running.

- **Returns**: String in the form "major.minor.increment"

15.7.3 *array* list_api_functions(*void*)

Get a list of functions exposed by the API. This may not be precisely the same as the list in this chapter, depending on how up-to-date the server and your copy of this manual is!

- **Returns**: Array of strings (names of functions)

15.7.4 *string* get_api_error_info(*int* code)

Get information about what a particular API error code indicates.

- `code`: integer error code.
- **Returns**: A string describing the error indicated by the code.

15.7.5 *void* log_in(*string* username, *string* password [, *bool* persist])

Log in to CQPweb. A username and password must be supplied; the `persist` parameter is optional.

- `username`: Username of the account to log in as.
- `password`: That account's password.
- `persist`: **true** is equivalent to ticking the "stay logged in" checkbox on the form in the browser. It will give all login tokens a much longer lifespan, one that might well last between sessions. The default is false.

This is a *void* function, i.e. it returns null regardless of the outcome. Any problem logging in is treated as an error, and an appropriate error code and/or message will be transmitted back within an object in error state.

15.7.6 *void* log_out(*void*)

Log out of CQPweb.

15.7.7 *array* fetch_freqlist([*int* subcorpus, *string* annotation, *string* filter, *string* filter_type, *int* freq_max, *int* freq_min, *string* sort])

Fetch the complete data of a frequency list (by default, of the whole corpus; if one is specified, of a subcorpus).

All parameters are optional.

- `subcorpus`: Integer ID of a subcorpus; the frequency list of that subcorpus will be returned.
- `annotation`: Handle of the annotation (p-attribute) to fetch frequency data for. Defaults to *word.*
- `filter`: A string used to determine which items are included in the returned frequency data.
- `filter_type`: One of *begin*, *end*, *contain*, *exact* (defaults to *begin*). Controls how the `filter` argument is interpreted.

  begin Only items that *begin with* the filter string will be returned.

  end Only items that *end with* the filter string will be returned.

  contain Only items that *contain* the filter string (anywhere) will be returned.

  exact Only items that *exactly match* the filter string will be returned.
- `freq_max`: Only items with *this number of occurrences or fewer* will be returned. Defaults to unlimited.
- `freq_min`: Only items with *at least this number of occurrences* will be returned. Defaults to 1.
- `sort`: One of *desc*, *asc*, *alph* (defaults to *desc*). Controls how the returned data is sorted.

  desc Sort by frequency, highest first (ties broken by alphabetical order).

  asc Sort by frequency, lowest first (ties broken by alphabetical order).

  alph Sort alphabetically (ties broken by frequency, higher first).
- **Returns**: Array of arrays. Each inner array represents a row of the frequency table, and thus has two values in it: the first is the type, the second is its frequency (string and integer, respectively).

15.7.8 *array* fetch_query_history([*int* limit])

Fetch the contents of the logged-in user's query history for the present corpus.

- `limit`: A maximum of *limit* history entries will be returned; default is unlimited.
- **Returns**: Array of objects. Each object represents a single history entry, and has the following members:

  timestamp String. Time when the query ran (in SQL timestamp format).

  simple_query String. The simple query (if CEQL was used in the query).

  cqp_query String. The CQP syntax query that was entered or that was generated from the simple query.

  query_mode String. Indicator of the mode used to ruin this query. One of *sq_case*, *sq_nocase*, *cqp*.

  query_scope String. Serialised representation of the sub-part of the corpus that was searched. N.B.: in future, the API may be amended to return something more tractable than just a string here.

  n_hits Integer. Number of hits returned. May be zero; can also be -1 or -3 to indicate a CEQL syntax error or a CQP run error, respectively.

15.8 Roadmap for future functions

- download query
- tabulate query
- get list of p- and s-attributes
- get list of active CEQL syntax
- create subcorpus
- compile subcorpus frequency list
- text / IDLINK metadata
- corpus statistics
- keywords
- dispersion
- distribution
- collocations
- frequency breakdown
- query postprocesses
- ... lots more ...

15.9 The CQPweb Client

《*The text in this section reflects how the client objects are intended to work; as of 2019-07, they don't actually work like this, as they are unfinished.*》 **TODO**

Client object classes are provided in a number of languages. Using them can make working with the API a little easier.

All the different languages follow, as closely as possible, the same object model.

They implement three types of method:

- Methods for managing the state of the client object. The client will monitor all state of your interaction with the server (e.g. your login), saving you from doing that explicitly. These methods include `ok()`, `set_secrets()`, `set_corpus()`, `get_error_messages()`, and so on.
- A method that encapsulates all the steps making a call to the API. This method, `call()`, takes only two parameters: first a string naming the function to run, most conveniently specified using one of the API_FUNC_ constants; and second an associative array (or hash, or dictionary) of argument names mapping to argument values for the function call.
- Convenience methods for certain commonly used API functions: `log_in()`, `log_out()`, `get_version()`, `get_cwb_version()`, `list_api_functions()`, `get_api_error_info()` and so on. All these functions could also be accessed with `call()`, of course.

The main differences among the clients for different languages are as follows:

- The JavaScript version uses asynchronous processing, in keeping with its likely use in web browsers.
- The C version's methods often require extra arguments, or different calling conventions, to deal with memory management.
- Some versions allow a callback function to be passed to `call()` to process the data received from the server before it is returned. Others do not (yet) have this.

Here is how to use any one of these client systems in your own code:

- Add the **CQPwebClient** class. How this is done depends on the language; `import` in Python, `include()` in PHP, etc.
- You should now have available to you (a) a class called **CQPwebClient**; (b) a set of constants prefixed `API_FUNC_`, each of which evaluates to a function name; (c) a set of constants prefixed `API_ERR_`, each of which evaluates to an integer error code that the API might raise.
- The constants may be implemented as variables, if you are using a language where constants aren't really a thing. For instance, in Python they are module-level variables; in JavaScript they are global variables.
- Create a CQPwebClient object using its constructor method, which takes a single argument: the URL of the CQPweb instance you wish to connect to.
- Use the `set_secrets()` method to specify username and password.
- Use the `log_in()` method to connect.

- Use the `set_corpus()` method to specify a corpus.
- Call API functions through the `call()` method. This takes the API function name as its first argument (you can use a string literal, or - to limit typo risk - one of the `API_FUNC_` constants) and a hash/associative array/dictionary/object of named parameters to pass to `api.php` as its second argument.
- Check if the call went well with the `ok()` method.
- If it went fine, use the return value from call() to do whatever further processing you need (possibly including more calls to the server).
- If the client object's state is not OK, use methods get_error_code() and/or get_error_messages() to build a meaningful error message, and do something with it.
- Repeat calls to the server as necessary.
- Log out with method `log_out()` - not strictly necessary, but makes things tidy.
- Unset/delete the client object (if necessary).

An example of this process, using the PHP client object, follows.

《*Call to log_in() seems to be missing below?*》 **TODO**

```
<?php
// Script exemplifying use of the CQPweb API
// via one of the client objects.

// Import the client class & its constants;
// we assume the code file is in the working directory.
include("CQPwebClient.php");

// Create client & set its secrets.
$c = new CQPwebClient("http://my-server.net/cqpweb");
$c->set_secrets("useraccount", "super secret passphrase");

// request the frequency list from the BNC1994.
$c->set_corpus("bnc1994");
$arg_hash = ["freq_min" => 50, "sort" => "alph"];
$fl = $c->call(API_FUNC_FETCH_FREQLIST, $arg_hash);

// check whether call ran correctly.
if ($c->ok())
{
  // If everything is OK, print out the list as table.
  echo "BNC1994: Word types with frequency no less than 50\n";
  foreach($fl as $row)
    echo $row[0], "\t", $row[1], "\n";
}
else
{
  // if not, print error diagnostics and exit.
  echo "CQPwebClient returned an error after calling "
```

```
    , API_FUNC_FETCH_FREQLIST
    , "with arguments as follows: \n\t"
    , print_r($arg_hash, true), "\n"
    ;
  echo "The error messages sent from the server were as follows:\n\t"
    , implode("\n\t", $c->get_error_messages()), "\n\n"
    ;
  exit(1);
}

exit(0);
```

16 Updating CQPweb

16.1 The update process

In general, to update CQPweb to a new version, there are three steps to take. The first step is always necessary, but the second two steps will only be needed when there have been major changes. The steps are:

- Check for new dependencies;
- Update the code;
- Update the database;
- Update the configuration file.

New dependencies on other pieces of software are rarely added. They will always be noted here when they are added:

- In version 3.1.7, a requirement for the R statistical software to be available was added (it was previously optional).

To **update the code**, simply get a new copy of the code from the CWB website, and copy the files over your existing installation. Be careful not to alter any of the folders relating to corpora you have installed. If your CQPweb is running from a Subversion checkout, then the following command, when run from the base directory, will typically complete all requisite actions:

- `svn update`

To **update the database**, you need to run the admin script `upgrade-database.php`. There is further information on this script in section 5.13. If you are upgrading a very old version of CQPweb, you may need to perform some updates manually; see section 16.2 below.

Updating the configuration file rarely needs to be done, as we make efforts to keep it consistent. The format of the configuration file changed between versions 3.0.16 and 3.1.0, and then again between 3.2.42 and 3.3.0. The changes are discussed in section 2.6.

Some updates may require additional actions as well as these three steps; if so, these are explained below.

16.2 Updating the database from very old versions

Early versions of CQPweb required MySQL schema changes to be implemented manually. A full list of the changes from version 2.15 to version 3.0.16 is given below. In each case, the SQL command given needs to run against the CQPweb database when logged in either as root or as the CQPweb database user. You need to run all the commands between the version you are upgrading *from* and the version you are upgrading *to*, inclusive (i.e. the list that follows is cumulative). Sometimes other steps are necessary, and they are listed too.

- Going from 2.15 to 2.16
  - `alter table user_settings drop key `username`;`

  - `alter table user_settings add primary key (`username`);`
  - `alter table user_settings add column `password` varchar(20) default NULL;`

- Going from 2.16 to 3.0.0
  - No database changes.
- Going from 3.0.0 to 3.0.1
  - `CREATE TABLE `corpus_categories` ( `idno` int NOT NULL AUTO_INCREMENT, `label` varchar(255) DEFAULT '', `sort_n` int NOT NULL DEFAULT 0, PRIMARY KEY (`idno`) ) CHARACTER SET utf8 COLLATE utf8_general_ci;`
  - `ALTER TABLE `corpus_metadata_fixed` MODIFY COLUMN `corpus_cat` int DEFAULT 1;`
  - Since this upgrade will wipe out your existing corpus categories, you must re-add them.
- Going from 3.0.1 to 3.0.2
  - `DROP TABLE IF EXISTS `xml_visualisations`;`
  - `CREATE TABLE `xml_visualisations` ( `corpus` varchar(20) NOT NULL, `element` varchar(50) NOT NULL, `xml_attributes` varchar(100) NOT NULL default '', `text_metadata` varchar(255) NOT NULL default '', `in_concordance` tinyint(1) NOT NULL default 1, `in_context` tinyint(1) NOT NULL default 1, `bb_code` text, `html_code` text, key(`corpus`, `element`) ) CHARACTER SET utf8 COLLATE utf8_bin;`
  - Go to System diagnostics and run the check for ``PHP inclusion files''
- Going from 3.0.2 to 3.0.3
  - `ALTER TABLE `xml_visualisations` DROP KEY `corpus`;`
  - `ALTER TABLE `xml_visualisations` ADD COLUMN `cond_attribute` varchar(50) NOT NULL default '';`
  - `ALTER TABLE `xml_visualisations` ADD COLUMN `cond_regex` varchar(100) NOT NULL default '';`
  - `ALTER TABLE `xml_visualisations` ADD PRIMARY KEY (`corpus`, `element`, `cond_attribute`, `cond_regex`);`
  - `ALTER TABLE user_settings add column thin_default_reproducible tinyint(1) default NULL;`
  - `UPDATE user_settings set thin_default_reproducible = 1;`
- Going from 3.0.3 through to 3.0.16
  - Nothing.

From version 3.1.0 onwards, the database template can be updated automatically by running the admin script `upgrade-database.php`. However, before you do this, you must manually apply all the relevant updates from the list above if you are moving from a version earlier than 3.0.16.

16.3 Updating from version 3.0.16 to version 3.1.0

This is a major upgrade, and careful adjustment will be needed. The following steps should be followed *in order*.

Step 1. Update the code (see section 16.1).

Step 2. Update your configuration file, to take account of the changes in the format listed in section 2.6.

Step 3. Update the database using the upgrade script described in section 5.13.

The above steps are essential. Some further steps are useful if you previously managed users/groups and their access rights using CQPweb's interface to Apache:

- Restore your previous groups using `load-pre-3.1-groups.php` (see section 5.9)
- Restore your previous privileges using `load-pre-3.1-privileges.php` (see section 5.10)
- Either (a) remove all `.htaccess` files from the web folders for particular corpora and/or (b) turn off the use of `.htaccess` files within the CQPweb web folder completely (see section 1.11 for a full account of setting up Apache under CQPweb 3.1 and higher).

Note that the first two of the above steps must be followed *in that order*, and *after* you have upgraded the database.

16.4 Updating from version 3.1.7 or earlier to version 3.1.8 or later

In version 3.1.8, the limit on how big a frequency list a user can create was changed from a single global value to a configurable privilege.

This means that, having upgraded to 3.1.8 or higher, you will need to use the Admin Control Panel (see 3) to add at least one privilege of this sort to your system, and assign it to users/groups.

The four default privileges of this sort enable users to create frequency lists for subcorpora of 1 million, 10 million, 25 million and 100 million tokens.

At least one privilege of this sort should normally be assigned to the "everybody" group, or else these users will not be able to create frequency lists for subcorpora at all.

16.5 Updating from version 3.1.8 or earlier to version 3.1.9 or later

In version 3.1.9, the "Analyse Corpus" function was added to CQPweb. In order to add the webpage supporting this function to existing corpora, you should go to the Admin Control Panel (see 3), and run the "Check corpus PHP inclusion files" function (found under *System diagnostics*).

16.6 Updating to version 3.2.0

There were substantial architectural changes in version 3.2.0, which is why it was a major version change. Although these changes caused little to be different on the surface, they were an essential step towards future developments.

If all goes well in the upgrade you will never need to know what the changes are. However, if something goes wrong, you may need the following information on the architectural changes to be able to effect a manual repair. The differences are as follows:

- In earlier versions, each corpus had a separate web folder inside the main CQPweb directory, with a set of PHP scripts in it. In 3.2.0 this was changed so that there was a single location for the corpus-interface scripts (the built-in sub-directory `exe`) and each corpus's web folder is now a symbolic link to `exe`.
- In earlier versions, a lot of important information about each corpus was stored in its "settings" file, stored in its web folder (filename `settings.inc.php`). In 3.2.0 all this information is transferred to the database, and the settings files are removed.
- In earlier versions, CQPweb relied on the CWB registry to keep track of s-attributes (XML elements/attributes). In 3.2.0, CQPweb has a database structure that keeps track of this information.

To upgrade from version 3.1.16 (or earlier) to version 3.2.0 (or later), you should follow these steps.

Step 1. It's recommended to take a backup copy of the entire web-directory containing the code of CQPweb and the web folders of the individual corpora before starting.

Step 2. Update the code (see section 16.1).

Step 3. Update the database using the upgrade script described in section 5.13.

Normally, the database upgrade script will bring the system right up to the current version of the code. However, it will always stop at version 3.2.0, to allow you to map across the corpus/XML settings from the earlier format. If your code version is above 3.2.0, you will need to run the database upgrade script *again* after finishing the rest of these steps.

Step 4. Run the special script to transfer existing corpus/XML settings to the new format.

The script for this step is listed in 5.11. To run it, go into the `bin` subdirectory, and enter the following command:

- `php load-pre-3.2-corpsettings.php`

The script goes through your list of corpora, and for each corpus it finds, it takes the following actions:

- First, it loads that corpus's settings file and inserts the information it finds into the SQL database.
- Second, it attempts to replace the corpus's web folder with a symbolic link to the `exe` folder.
- Third, it interrogates the CWB registry to discover the corpus's s-attributes, and creates a record of each attribute in the database.

Step 5. You may see error messages from the special script if any step of the process does not complete correctly, so the next step is to address these messages by making manual adjustments.

- If no settings file is found for a particular corpus, this is probably not a problem: the setup of the corpus in question was already broken.
- If the script reports that, for a particular corpus, it could not replace the web directory with a symlink, then the manual fix for this is to run the following shell commands within the CQPweb main directory:
  - `rm -r corpus`
  - `ln -s exe corpus`

but with the actual corpus handle instead of “corpus”! You may then need to adjust the ownership/permission of the resulting symlink (see notes on web-directory ownership and permissions in 1.4).

Step 6. In earlier versions, as noted above, the “settings” file stored key information. It was possible for system administrators to manually add variables/code to these files, to add extra tweaks to the interface on a per-corpus basis. This has long been *highly inadvisable* due to the increasing complexity of the system, but from version 3.2.0 onwards, it is no longer possible. So your final step, *if and only if you have made any such modifications*, is to review your old `settings.inc.php` files (from your backup created as per above!) and double-check the effects of the loss of your manual tweaks. Some of them may be replicable through the usual administrative tools described in the rest of this manual.

If you never manually edited any of the settings files, then you have nothing to do under this step.

16.7 Updating to version 3.2.4

When you update to version 3.2.4 all users who are currently logged in will be automatically logged out. This is due to a change in the database format regarding the storage of login tokens.

16.8 Updating to version 3.2.6

Running the database upgrade script for the update to 3.2.6 can take a long while, especially if you are updating a server with lots and lots of users, or if CQPweb has been installed for a very long time. DO NOT abort the update script - let it run to completion. If you abort it before it has finished, your database may end up in a half-and-half state, in which case it would become very difficult to repair it without losing some of your users' data.

16.9 Updating to version 3.2.23

In version 3.2.23, the limit on how large a file a user can upload was changed from a single global value (hardcoded as 2 MB) to a configurable privilege.

This means that, having upgraded to 3.2.23 or higher, you will need to use the Admin Control Panel (see 3) to add at least one privilege of this sort to your system, and assign it to users/groups.

The three default privileges of this sort are for file size limits of 0.5 MB, 1 MB, and 2 MB.

At least one privilege of this sort should normally be assigned to the “everybody” group, or else these users will not be able to upload files at all.

16.10 Updating to version 3.2.32

In version 3.2.32, the STTR statistic was added to the information stored for corpora.

You should run the following command in order to add the STTR to your existing corpora:

```
php execute-cli.php update_all_missing_sttr
```

Be aware this can take a LONG time to run, and must only be done *after* you have upgraded the database!

(A message to this effect is also printed when you run the database upgrade script.)

The STTR for *new* corpora will be calculated when they are installed.

This version also adds the system allowing users to install their own corpora. However, before this is possible, you will need to set up the appropriate plugins, and assign upload and corpus creation privileges; likewise you will need to set the `$user_corpora_enabled` configuration variable (see 2.3.8).

16.11 Updating to version 3.3.0

Various major changes were made in the move from 3.2 to 3.3.

Configuration file format: this changed in ways that may require you to modify your existing configuration file upon update. See section 2.6.1.

Extra code files (CSS and JavaScript): these were previously to be placed in the `css` and `jsc` folders within the main CQPweb directory. From v 3.3.0 onwards, they should instead be placed in the `extra` subdirectories (that is: `css/extra` and `jsc/extra`). See 10.4.6

Plugins: PHP files for plugins are no longer directly contained in `lib/plugins`. Instead, that folder contains a folder tree called `builtin`, which has one folder per type of plugin; e.g. `lib/plugins/builtin/CorpusInstaller`.

Plugins do still need to be in `lib/plugins` to be used; the principle is that you should create a symlink to just those builtin plugins that you wish to use, as follows:

```
cqpweb/lib/plugins > ln -s builtin/CorpusInstaller/BasicVrtInstaller.php
```

This system was added because, as the number of builtin plugins increased, the plugins folder became harder to manage.

A `local` folder, with the same subdirectories as `builtin`, is provided for convenience; it is for you to place your own plugins into. However, you can symlink (or copy) your plugins from anywhere. You do not *have* to use `lib/plugins/local`.

See 12.

The **database upgrade** procedure for v 3.3.0 involves major architectural changes. It may take a long time to run. So, it is recommended to run the process with your CQPweb server offline (see 2.3.11 on the `$cqpweb_switched_off` setting).

Because this is a major upgrade, it is possible to generate the SQL commands that will run during upgrade without running them, by adding the `--rehearse` flag to the call to the upgrade-database script. A second flag, `--stepwise`, allows you to run the upgrade a chunk at a time (you must keep running the script repeatedly until all chunks are complete). Note that `--rehearse` implies `--stepwise`, i.e. you can't rehearse the second chunk until you have run the first chunk for real (and so on).

17 Glossary

⟪*turn this into an appendix*⟫ **TODO**