The IMS Open Corpus Workbench (CWB)

CQPweb System Administrator's Manual

— CQPweb Version 3.2 and above —

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

February 2016

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 is planned but not yet achieved, so for the moment if you are on Windows you will need to install CQPweb, and all its dependencies, on top of Cygwin.

Other software you need to have installed:

- Apache or some other webserver
- MySQL (v5.0 at minimum, and preferably v5.6 or higher)
- PHP (v5.3.0 at minimum, and preferably v5.6 or v7.0+)
- Perl (v5.8 or higher)
- Corpus Workbench (see 1.4 and 1.5 for more details)
- R
- Standard Unix-style command-line tools: **awk**, **tar**, and **gzip**; either GNU versions, or versions compatible with 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) 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 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 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, MySQL 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 the non-administrator user can set in motion.

⟪*new section on disk space and advising on partitioning etc?*⟫ **TODO**

How much disk space precisely MySQL 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 MySQL daemon can get just with RAM. However, as a rule of thumb, you should aim to run CQPweb's MySQL 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-indexing 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 - MySQL 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.

1.3 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 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 say 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 4.

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 web-server 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 1.10 for more information on this process.

1.4 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 need at least version 3.2.0 of CWB if you want CQPweb to have UTF8-compliant regular expression matching; otherwise you need at least version 3.0.0.

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).

1.5 Setting up the Perl modules

CQPweb relies on the CWB-Perl interface, so you will need to install this.

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

The only package you strictly need 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 not needed just to run CQPweb.

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 "gotacha" to be aware of: if you update your system software to use a new version of Perl, the modules in directories whose path includes the old version number will 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).

CQPweb requires at least version 3.0.0 of CWB-Perl.

1.6 Setting up R

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

1.7 Setting up PHP

You need at least version 5.3 of PHP.

- The **zlib** extension is required
- Either the **mysql** or the **mysqli** extension is required (see below)
- The **gd** extension is needed if you want the user-account-creation function to be protected by CAPTCHA.

CQPweb crucially requires either the **mysql** extension to PHP or the more recent equivalent **mysqli** extension (the "i" stands for "improved"); these are used to connect to the MySQL database. A ny given version of PHP is almost certain to include one or the other of **mysqli** or **mysql**, and may well include both. In the unlikely event that your version of PHP does not contain either extension, you must recompile PHP with one of the two.

If you are running CQPweb on the internet (i.e. not simply on a standalone computer), then it is also crucial for CQPweb 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('your.address@somewhere.net','itworks...','Yes it does work.');"`

(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 webserver; one

known "gotcha" on Apple OS X is that the `php.ini` file may not exist, if not, then 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 may be different depending on how you installed PHP (or your system administrator may have subsequently adjusted them). Most of them will 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 `20M` (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 moderately generous as some CQPweb operations are RAM-intensive (e.g. loading entire files into memory); we suggest `25M` (but if the default in your system is higher than that, keep the higher value!)
- `max_execution_time` should be generous as well; we suggest `60`

Less "generous" limits for file size, memory use, and execution time may be OK if you are running CQPweb on a standalone computer rather than a shared web server - although on the other hand, on a standalone computer it doesn't matter nearly so much if CQPweb locks up all the system's resources for long periods at a time! Once CQPweb is up and running, the values of all these 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 distros, including Debian and Ubuntu) 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 very long CQPweb queries failing to work if they overrun this limit. 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. For example, some versions of Internet Explorer are known to restrict HTTP request values to 2000 bytes.

Finally, PHP has a "safe-mode" (see http://php.net/features.safe_mode). This mode restricts what PHP can do in an attempt to provide security. However, it has been recognised as misguided, and is deprecated in PHP v5.3 and removed in PHP v5.4. If your system has safe-mode enabled in spite of this, you may well find that some CQPweb operations do not work:

- PHP safe-mode may not allow CQPweb to run CQP and Perl as separate child processes if the `cqp` and `perl` executables are not in its "safe" directories. If this happens, CQPweb will simply not work at all.
- PHP safe-mode will stop CQPweb from lifting restrictions on execution time for long-running operations; these functions (mostly set-up functions for big corpora, but also processes like collocations on queries with lots of results) will stop working.

- PHP safe-mode will restrict CQPweb to only opening or modifying files that CQPweb has created itself. However, some operations need CQPweb to work with files which it didn't create itself but were, rather, created by the MySQL server or manually by you. All these operations will stop working.

The solution is to turn safe-mode off. Find the line in your `php.ini` file that looks like this:

- `safe_mode on`

and change it to

- `safe_mode off`

1.8 Setting up your webserver

1.8.1 Overview

There are two 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.

1.8.2 Using HTTPS

⟪*write this*⟫ **TODO**

1.8.3 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 earlier versions of CQPweb, **.htaccess** files were used to control user access to corpora. *As of version 3.1.0, this is no longer the case.* 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 1.8). 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>
```

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).

You can then block access to the two sensitive subdirectories as follows:

```
<Directory /path/to/cqpweb/bin>
  deny from all
</Directory>
<Directory /path/to/cqpweb/lib>
  deny from all
</Directory>
```

If you cannot change the Apache configuration file, then the only option for blocking access to those directories is to use **.htaccess** files (assuming they are enabled).

In this case, simply create two identifcal files (`bin/.htaccess` and `lib/.htaccess`), each containing just the following directive:

- `deny from all`

and likewise make sure that there is a **.htaccess** file in the CQPweb main web directory which switches on the `FollowSymlinks` option using an `Options` declaration line.

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 webserver-based 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 built-in 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.
- 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 are *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).

1.9 Setting up MySQL

1.9.1 Creating the database

CQPweb uses a MySQL database 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 MySQL server in order to set it up. The following instructions are based on the assumption you are accessing MySQL via the command-line client program, but it is also possible to use your preferred graphical interface for this purpose, 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 MySQL commands are as follows:

- `create database cqpweb_db default charset utf8;`
- `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 only-mildly-secure location, so make sure that you do **not** re-use either an account name or a password that is used for any other purpose on the same system.

The reason that we set the default character set to UTF-8 is that this addresses a bug which affects some (but not all!) versions of the MySQL server software – see http://bugs.mysql.com/bug.php?id=10195.

Having created the user, we must now give it *all* permissions over the database. If you want MySQL to use file-access functions, the new user also needs to be granted the *file* permission, which is set once-and-for-all on a single MySQL server, rather than at the level of the database. File-access permission is not necessary, but may help speed things up; it can also be useful if the LOAD DATA LOCAL INFILE command is disabled (see 1.9.2 below).

- `grant all on cqpweb_db.* to cqpweb_user;`
- `grant file 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 MySQL server by a webscript. This will probably be **localhost**, assuming that the MySQL server is on the same machine as the webscripts (but see 1.9.3 below).

1.9.2 Known "gotchas" in the MySQL setup

- **Local infile permission**: MySQL can be configured to disable the `LOAD DATA LOCAL INFILE` command as a security measure – see http://dev.mysql.com/doc/refman/5.5/en/load-data-local.html . This command 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).

  The preferred way to fix this is as follows:

- Edit the MySQL configuration file (usually something like `/etc/my.cnf` depending on your operating system).
- Find the line `set-variable=local-infile=0` and change the `0` to `1`.
- Restart the MySQL daemon.

Alternatively, the problem should be fixed if you set the CQPweb configuration variable `$mysql_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 below.

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

- **PHP's connection to MySQL**: PHP needs to know how to connect to the MySQL server 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 MySQL 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).
- **MySQL binary logging**: Binary logging is a MySQL feature that can be enabled or disabled, as explained here: http://dev.mysql.com/doc/refman/5.5/en/binary-log.html. If enabled, even relatively light use of a CQPweb installation will make MySQL 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 MySQL server that will be driving CQPweb. You can disable binary logging as explained here: http://dev.mysql.com/doc/refman/5.5/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 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*.)

1.9.3 Using a separate computer for MySQL

We normally assume that the MySQL server 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 MySQL-based parts of CQPweb, for reasons of performance (for a big corpus and for queries with lots of results, both MySQL and CWB require lots of disk space and lots of processing power).

In this case, CQPweb itself (the web scripts) should be on the same system as CWB, and the MySQL server 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 MySQL server 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 system with CQPweb on it*, not the system with MySQL on it.
- The optional variable (see 2.3) `$mysql_has_file_access` must not be set to **true** if the MySQL server is on a separate computer. The system cannot currently check this for you!

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

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

- If the CQPweb system is the only user of the MySQL server machine, then for security it would make 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 packages 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 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.

1.10 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.

If you also use CWB and CQP from the commandline 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 commandline 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 if you want MySQL to use file-access functions. The easiest way to accomplish this is to give read-write-execute pemrissions 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 MySQL 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 `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.

A "gotcha" can occur here in systems that have AppArmor (for example, some versions of Ubuntu). AppArmor disallows file access to programs that it thinks ought not to be manipulating a particular directory, even if the username of that program has all the proper permissions. If it does not allow *mysqld* access to the CQPweb directories, you will not be able to run CQPweb successfully.

You can fix this by adding 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 must, of course, be edited as root. 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 (`/etc/init.d/apparmor restart`).

The alternative is to disable AppArmor completely if you do not need the extra security it supplies.

1.11 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 template - 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 you created in the installation steps above:

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

1.12 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 MySQL database, and setting up default data;
- Creating accounts for the admin usernames specified in the configuration file;
- Generating important files such as the CSS stylesheets.

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 commandline 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 commandline scripts see section 4.1.

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 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.inc.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 template 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. Then, there is a much longer list of variables that you *can* set, but if you don't, default values will be used. 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 1.10 on "Setting up directories".

Additional information on the variables relating to MySQL can be found in section 1.9 on "Setting up MySQL".

Variable name	Description
`$superuser_username`	**Type:** String This should contain the usernames of users who are to be system administrators, separated by the pipe `|` if there is more than one. For instance: `"anna|bert|craig"`. You can have as many admin accounts as you like, but you must have at least one.
`$mysql_webuser`	**Type:** String The username of the MySQL account that CQPweb should use to connect to the MySQL server. It is usual to have a single dedicated MySQL account for use by CQPweb.
`$mysql_webpass`	**Type:** String The password of the `$mysql_webuser` account on the MySQL server.
`$mysql_schema`	**Type:** String The name of the database on the MySQL server that you have created for use by CQPweb.
`$mysql_server`	**Type:** String The address of the MySQL server (either a hostname/IP address, optionally followed by a port number, or else a path to a local socket; see the MySQL documentation for more info on this). If your MySQL server is on the same computer as the rest of CQPweb, this variable will 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.

⟪*Note that not all the optional configuration variables are documented yet.*⟫ **TODO**

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

Variable name	Description
Locations of programs on the system	
`$path_to_cwb`	**Type:** String **Default:** `""` Path of the directory containing the CWB executables (`cqp`, `cwb-encode`, and so on). The path can be absolute or relative; if relative, bear in mind that CQPweb always runs from one of the immediate daughter directories of its main directory. If the path contains any space characters, they must be escaped (with an escaped back-slash, e.g. `"../Program\\ Files/..."`. If no path is given, it will be assumed the executables are in the system's usual path.
`$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 `$path_to_cwb`.
`$path_to_r`	**Type:** String **Default:** `""` Path of the directory containing the R executable; same general comments apply as to `$path_to_cwb`.
`$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.
MySQL features	
`$mysql_freqtables_size_limit`	**Type:** Integer **Default:** `6442450944` This variable places a limit on how much disk space *ad hoc* frequency tables are allowed to take up. It is expressed as a number of bytes; the default is 6GB.

Variable name	Description
`$mysql_big_process_limit`	**Type:** Integer **Default:** `5` This variable places a limit on how many *big* MySQL processes of a single type will be allowed to run at once. There are several types of "big" process (building collocation database, building frequency tables, building sort databases, and building categorised query tables) so more than the limit could run.
`$mysql_utf8_set_required`	**Type:** Boolean **Default:** `true` Controls how characters are transmitted between the MySQL server 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.
`$mysql_has_file_access`	**Type:** Boolean **Default:** `false` This variable declares to CQPweb that the MySQL daemon has access to the filesystem on which CQPweb is running. If you set it to **true** then this affects various functions that load data from file to the MySQL server: they may run faster. But this must only be set to **true** if (a) the MySQL server is running on the same computer as CQPweb and (b) the MySQL server actually does have access to the temporary directory and all other locations that CQPweb writes files to and (c) the MySQL username used by CQPweb has `GRANT FILE ON *.*` permissions. It is important you get this right, as CQPweb doesn't check these preconditions for you!
`$mysql_local_infile_disabled`	**Type:** String **Default:** `false` You should set this to **true** if your MySQL has been set up to disallow the `LOAD DATA LOCAL` command, and you can't change this set-up. If possible, changing the MySQL configuration to allow `LOAD DATA LOCAL` is far preferable, it should be noted! See also section 1.9.2.
Memory and other hardware resource limits	
`$cwb_max_ram_usage`	**Type:** Integer **Default:** `50` Some CWB programs allow a RAM usage limit to be set on their activities. When CQPweb calls these programs, it sets the RAM limit to the number of megabytes specified in this variable. The default is 50 megabytes. This variable applies only when CQPweb is run over the web; for the RAM limit that applies when CQPweb is run from the commandline, see the next variable.
`$cwb_max_ram_usage_cli`	**Type:** String **Default:** `1000` Same as `$cwb_max_ram_usage`, but applies when CQPweb is run from the commandline (see 4).

Variable name	Description
`$cache_size_limit`	**Type:** Integer **Default:** `6442450944` Controls the size of the query cache (the maximum size, in bytes, to which the temporary directory will be allowed to grow before old cached queries get deleted). Note that this only affects the size of the query cache; anything stored as a MySQL table (such as temporary frequency tables or collocation databases) does not count towards this limit. Until the cache limit is reached, the cache will just keep growing! Cached files are never deleted merely due to age, only when the disk space needs to be reused. The default value is equivalent to 6 gigabytes.
Configuring the user interface	
`$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). All tools also allow the results-per-page rate to be altered on a per-query basis.
`$default_history_per_page`	**Type:** Integer **Default:** `100` The number of items to show per-page in history-type displays (such as query history, saved queries, and so on). Note that as a general rule you would normally want more of these to display per-page than you would for `$default_per_page` - thus the difference in default values.
`$dist_graph_img_path`	**Type:** String **Default:** `"../css/img/blue.bmp"` This string is the (relative) address of an image file that will be used for the bars in the bar chart mode of the Distribution display. The default is a file internal to CQPweb that creates plain blue bars.
`$dist_num_files_to_list`	**Type:** Integer **Default:** `100` The number of files to display in File Frequency Extremes mode of the Distribution display/. The number of files is limited becauase corpora can easily have thousands or tens of texts, which would make the webpage hard to read and slow to load.
`$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.
Tweaking the look-and-feel	

Variable name	Description
`$css_path_for_homepage`	**Type:** String **Default:** (see below) The relative or absolute URL of a CSS stylesheet to use for the main menu page. Note that, if relative, this must be relative *to the homepage*, which is one level higher (in the directory tree) than all the other URLs that CQPweb runs from. So if you want to address something in the CSS folder, for instance, you would need to start this variable with `css/`, rather than `../css`. By default, a lovely blue-and-grey table effect is used.
`$css_path_for_adminpage`	**Type:** String **Default:** (see below) The relative or absolute URL of a CSS stylesheet to use for the admin control panel page. By default, a red-and-pink colour scheme is used.
`$css_path_for_userpage`	**Type:** String **Default:** (see below) The relative or absolute URL of a CSS stylesheet to use for the user-login homepage. By default, a green-and-grey colour scheme is used.
`$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.
`$homepage_welcome_message`	**Type:** String **Default:** `"Welcome to CQPweb!"` A little bit of text (which can include HTML formatting) that will appear in the header box of the main menu page.
`$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.

Variable name	Description
`$homepage_logo_right`	**Type:** String **Default:** `""` Same as `$homepage_logo_left`, but whatever you specify appears on the right side of the header box.
`$searchpage_corpus_name_suffix`	**Type:** String **Default:** `"powered by CQPweb"` A little bit of text that is suffixed to the name of the corpus in the main search page header. If you don't want anything, set it to an empty string.
RSS feed control	
`$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.
Error reporting	
`$print_debug_messages`	**Type:** Boolean **Default:** `false` If **true**, messages about what is going on behind the scenes will appear all over various CQPweb pages (totally messing up the layout, but hopefully helping you to diagnose problems).

Variable name	Description
`$debug_messages_textonly`	**Type:** Boolean **Default:** `false` If you set this to **true**, all debug and error messages will be printed as text with no HTML formatting. Plain text messages are always produced (a) when outputting a textual download; (b) in scripts that run from the command-line.
`$all_users_see_backtrace`	**Type:** Boolean **Default:** `false` If you set this to **true**, error messages will contain a PHP backtrace regardless of which user is logged in. By default, only admin accounts see the PHP backtrace.
Miscellaneous (unsorted)	
`$cqpweb_root_url`	**Type:** String **Default:** `""` You can set this to an absolute URL (in the style `"http://your-server.net/path/to/cqpweb/web/directory"`) and this URL will be used internally when CQPweb redirects the user's browser from one point in the system to another. If you don't set it, then CQPweb will try to work out its own URL based on information in PHP's global `$_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.
`$cqpweb_no_internet`	**Type:** Boolean **Default:** `false` If this is set to **true**, CQPweb assumes it is not available to the open internet and that it is only being accessed locally (i.e. by a web browser sending HTTP requests to **localhost**). This has two effects. First, CQPweb will disable all email-sending functions. Second, normal user account signup (which relies on email) will be turned off; that is, **true** here overrides `$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.

Variable name	Description
`$cqpweb_email_from_address`	**Type:** String **Default:** `""` An email address, which if provided will be used as the "From:" and "Reply-To:" address for all emails sent by the system. If this is not provided, defaults will be provided by your system's email-sending program - CQPweb will not attempt to supply a default. Be aware, however, that some mail systems will block emails that lack a clear "From" address, as they are assumed to be spam - so it si wise to set this. The email can be in the form of either a bare address (*someone@somewhere.net*) or a named address (*A. N. Other <someone@somewhere.net>*). There are two basic possibilities here: you could set this to a real email address, so that if users reply the replies will go to some moinitored mailbox; or, you could set it to an obviously fake address that (a) will signal to users they should not reply and (b) will ensure that, if they *do*, the replies go into a black hole: an example value for the latter would be `"CQPweb Server <do-not-reply@cqpweb.anytown.edu>"` or something along those lines.
`$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 specfiied 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 userrs can access the pages where it is shown. 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.

Variable name	Description
`$cqpweb_cookie_max_persist`	**Type:** Integer **Default:** `5184000` This is the maximum length of time, in seconds, that a user's login will persist if they do not visit the site (the clock is "reset" every time they do visit the site). The default value is 60 days. This does not affect the option given to the user to choose whether or not they stay logged in; if they choose *not* to stay logged in, their browser will delete the cookie anyway and the persistent login will never be re-used.
`$cqpweb_running_on_windows`	**Type:** Boolean **Default:** (see below) You can set this variable to **true** to declare that the operating systme 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. (Moreover, since Windows compatability is not yet implemented as of v3.1, this variable does not actually do anything yet.)
`$allow_account_self_registration`	**Type:** Boolean **Default:** `true` If this is true, a form will be exposed for anyone to sign up for an account on your server. If false, this form will not be available, and only admin users will be able to create new user accounts. (You can alternatively configure your web server to block or limit access to the signup form if you wish.)
`$account_create_contact`	**Type:** String **Default:** `""` This variable only has any effect if `$allow_account_self_registration` is **false**. In this case, you can supply a snippet of text here that will be added to the web interface to tell prospective users who to contact to request an account. This string can contain HTML markup, for example to add a "mailto" link, e.g. `"<a href=\"mailto:a.n.other@anytown.edu\">A. N. Other</a>"` .
`$account_create_captcha`	**Type:** Boolean **Default:** `true` Determines whether or not the account-creation form is protected by a CAPTCHA challenge. This can help protect your server against web-robots, but you might want to disable CAPTCHA for convenience if your account-creation page is inaccessible to users on the open Internet anyway. If your PHP installation lacks the GD extension to PHP (see: `http://php.net/gd`), then this setting is *always* **false**, regardless of what you specify.

Variable name	Description
`$create_password_function`	**Type:** String **Default:** `"password_insert_internal"` CQPweb uses "suggest password" functions to ease the creation of user accounts in the admin interface. The default function (`password_insert_internal`) produces randomised passwords of the form **aaaaddaaaa**, where **a** is a lowercase letter and **d** is a digit. You can optionally use Lancaster University's web-based random password generator to get nicer, more wordlike passwords: to do this, set the `$create_password_function` to `"password_insert_lancaster"`. This will probably not work if your system needs to use a proxy server to access the web, however. You can also write your own nice-password function (in the file `admin-lib.inc.php`) and specify it here.

2.4 Using the auto-configuration script

TODO

《*write this*》

The script is interactive, and will ask you a series of questions. For each question, the answer determines one of the compulsory settings. ... and follow the instructions to enter the paths and other info that you made a note of above. When you are asked to specify admin usernames, enter at least one username (the one you will use).

If a configuration file already exists, the script will not overwrite it.

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

4.2

2.5 Using the configuration file template

The file `template.config.inc.php` in the `lib` subdirectory of CQPweb is a blank template file which contains all the compulsory configuration variables. The easiest way to create a configuration file manually is using this template file, as follows:

- Make a copy of the `template.config.inc.php` file in the `lib` directory
- Rename the copy to `config.inc.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.

2.6 Changes from CQPweb version 3.0.16 or earlier

The configuration file format described in this chapter is that of version 3.1 of CQPweb. 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.

(Not mentioned here: the configuration variables, of which there are many, which are new to version 3.1.0; for those, see the table of optional configuration variables in section 2.3.)

- 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 program location, and has been abandoned. In 3.1, both 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` .

2.7 Obsolete feature: the corpus settings file

Prior to version 3.2, there existed an undocumented feature whereby any of the variables in the configuration file could be overridden on a per-corpus basis by setting a different value into `settings.inc.php` file for that particular corpus.

As of version 3.2, this is no longer possible:

3 The Admin Control Panel

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 ¿ will be added.

⟪*add a note on help re: admin control panel. Is there any?*⟫ **TODO**

3.2 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.

- Corpora
  - –
- Uploads
  - –
- Users
  - –
- Database
  - –
- System

  –

- Corpora

  –

- Corpora

  –

- Corpora

  –

4 Administering CQPweb from the commandline

4.1 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 commandline 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 MySQL server, 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.

4.2 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`.

4.3 autosetup.php

This script is used in the process of setting up a new CQPweb installation. It is described in section 1.12.

Like `autoconfig.php`, it is interactive.

4.4 cli-lib.php

This is not a script, it is a library file used by the other commandline 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.)

4.5 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`.

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.

4.6 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 ⟪*XREF*⟫). **TODO**

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.

4.7 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`.

4.8 load-pre-3.2-corpsettings.php

The use of this script is explained in 11.6.

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.

4.9 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 commandline 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.

4.10 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 nto run this script to bring the database into line, CQPweb will break (probably 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 11 for more information on upgrading.

5 Indexing corpora

5.1 Basic concepts

CQPweb is based, naturally, on CWB corpus indexes. Indexing 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 one or more 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 (6).

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*.

5.2 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 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 of the table/column namers. Other handles are used as database keys (and some database keys in CQPweb include up to three handles).

However, MySQL 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 a re 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.

Note that 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).

5.3 Annotation

5.4 Annotation templates

5.5 XML

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

5.6 XML templates

5.7 The indexing process

(all about the form) (inc use of the non-template forms)

5.8 The metadata setup process

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

5.9 Building frequency lists

5.10 Setting up corpus access rights

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

6 Metadata

6.1 Introduction

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

6.2 Text metadata

6.3 XML metadata

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

6.4 The different possible datatypes

(A heading for each metadata type)

Free text – for items where the metadata can vary across every single text (e.g. text title)

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

6.5 Metadata templates

6.6 Matadata file format

6.7 Installing metadata

7 Controlling corpus visualisation

In CQPweb, *visualisation* is a covering term for anything to do with how corpus data is rendered in the web interface. This chapter describes how the system administrator can control different aspects of the visualisation process.

8 User accounts and privileges

8.1 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.

User accounts are created by a self-signup 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 send an invitation to a specified email address, with a link that will take the user directly to the account-creation form.

User accounts must be linked to email addresses. This is because the user's email address is the only way 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 built-in groups:

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

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 copntrol 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 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 thsat 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.

8.2 User accounts

creating account / issuing invitation

viewing info on an account

deleting an account

account expiry

password expiry

8.3 User groups

Creating and deleting groups

Adding users to groups (and removeing them)

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. 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 associated with that regex.

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 group regex

running ad hoc regexes across gropups

8.4 Privileges

⟪*begin old material*⟫ **TODO**

Use "manage users" for this.

CQPweb gives access to corpora to groups of users, not individual usernames. This is done through the "manage access" link for each corpus: find this on the "show corpora" item of the admin menu.

If you're using a normal web server, CQPweb passwords are transmitted in plaintext via HTTP. If you really want secure username/password access, you need to (at least!) set up CQPweb on an HTTPS server. You can use the `$password_more_security` configuration variable to assist the process of enhancing security (see above).

⟪*end old material*⟫ **TODO**

⟪*begin rough notes on Restricted corpus access*⟫ **TODO**

restricted access permissions – that is, the ability to let people use a corpus in a limited way.

Typically for people who are not licensed to use them. The "restricted access" is designed in such a way as to stop people getting at more of a corpus than concordance-length snippets, which are less liklely to be elgallyt problematic (Though of course this depends on many other factors, most notably the legal jurisdfiction youy are in.)

The restrictions are as follows:

You can't view extended context if you only have restricted access. You can't use the "tabulation" function. If your 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 in tokens.)

To explain the last one: 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. The "half the square root" system tries to counteract this, while not making concordances in small corpora totally useless.

So, for instance, half the square root gets you 5000 examples in a 100MW corpus, but 1000 examples in a 1MW corpus. (It's always rounded upwards to a round N 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 you've searched for is – so the underlying corpus can't be reconstructed.

(And the limit is calculated relative to the amount of text you search, so you can't cheat it, e.g. by creating a subcorpus of just one text and then searching within that, because the hit limit is calculated relative to the amount of text searched.) ⟪*end rouggh notes on Restrected corpus access*⟫ **TODO**

8.5 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.

9 Using plugins

9.1 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, which allows advanced users to add capabilities to the system which it does not possess out-of-the-box.

CQPweb is such an application. A CQPweb plugin is a little bit of code that is added to the system, which is then accessed in a predefined way by CQPweb to provide extra capabilities for users.

Some plugins are supplied with CQPweb. You can also write your own.

Several forms of plugin are under development. 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!

As of the current version of CQPweb, only one type of plugin is implemented: *Custom Postprocesses*. This chapter explains some general things about plugins, and some specific things about Custom Postprocesses.

9.2 Types of plugin

9.2.1 Custom Postprocesses

A Custom Postprocess allows you to define an additional way in which a query can be postprocessed.

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 always keeps a record of all the postprocesses that have run on a query since it was originally produced by a from-scratch search.

When you install a Custom Postprocess, 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 things makes sense!

⟪*all the other types of plugin, as the code to support them is completed*⟫ **TODO**

9.3 Installing plugins

⟪*update this; may no longer be accurate*⟫ **TODO**

Declaring a plugin

Once you have added your plugin file to the correct place on the system, you must *declare* it in your configuration file (lib/config.inc.php), so that CQPweb knows it should activate this plugin where necessary.

Declarations take the following form:

```
declare_plugin('My_Plugin', PLUGIN_TYPE_POSTPROCESSOR, '/path/to/config/file');
```

This is a function call with three arguments. The first is the name of the class you have chosen. The second is the symbolic constant for the type of plugin it is (see table in previous section). The third argument is optional. It is a path to a plugin-specific configuration file. If the third argument is given, this path will be passed to the plugin object when it is created (and you then handle it however you

like). See discussion of the `__construct()` method below. The idea is to allow the same plugin file to be used on different machines without modification, with the differences abstracted into configuration variables which can be stored separately. The predefined plugins, which are intended to be use don many systems, use this approach. Of course, if you do specify a configuration file, you need ot make sure that that file actually exists and is readable!

9.4 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 9.3.

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).

9.4.1 DeleteEveryThirdHit

This 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 native postprocess types.

⟪*add more builtin plugins, as and when*⟫ **TODO**

9.5 Creating plugins

9.5.1 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 the plugin file into the `plugins` subdirectory of the `lib` directory. Like all CQPweb code files, this new file must be readable by the username your webserver runs under.

The file should contain nothing except the class that represents your plugin. 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, as follows:

Type of plugin	Interface to implement	Symbolic constant
Custom postprocess	`Postprocessor`	`POSTPROCESSOR`

(Others will be added as they are implemented.)

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
}
?>
```

There should be no whitespace before or after the `<?php` and `?>` delimiters. If there is, CQPweb may stop working properly.

9.5.2 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.

9.5.3 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 the major work of the plugin will 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 that in `lib/plugins.inc.php`. If the comments there are different from what you find here, then they should be considered definitive as they may be more recently updated or more extensive.

9.5.3.1 Methods required by every type of plugin

- `public function __construct($config_file = '')`

This method initialises the plugin. If a configuration file was declared for the plugin, its path will be passed into the object here - so you will need to catch this variable and do something with it if you want to access that path.

9.5.3.2 Methods required by Custom Postprocess plugins

- `public function postprocess_query($query_array)`

This is the method that actually does the postprocessing. It must take a CQP query, do something with it, and then return the altered query.

The parameter is an array of arrays. Each inner array contains (up to) four numbers that result from "dumping" a CQP query. See the CQP documentation for more info on this. Basically each match is represented by two, three or four numbers:

- `match, matchend, [target, [keyword]]`.

The outer array will have sequential integer indexes beginning at 0.

The return value should be the same array, but edited as necessary - for example, entries removed, or expanded, or results added... as appropriate to the purpose of the custom postprocess.

The inner arrays can contain integers or integers-as-strings (both will be OK as a result of PHP's automatic type juggling). All inner arrays must be of the same length (i.e. you do not have to supply **target** and **keyword** if you don't want to, but if you do, every match must have them). The indexes in the inner arrays are not important, only the *order* of the elements; but the outer array will be re-sorted, so order in that does not matter.

- `public function get_label()`

This method needs to return a string to be used as the menu label to invoke this postprocess.

Postprocesses are invoked from the dropdown on the concordance screen. So, custom postprocesses must tell CQPweb what they want their label on that dropdown to be. (The corresponding value for the HTML form will always be just Custom_ + the classname.) This function should return either a string, or any empty value if this postprocess is not to be displayed in the dropdown menu.

The string should not be identical to any of the options present on the built-in dropdown menu. (Nothing in particular would go wrong if they were - it would just be extremely confusing.)

- `public function get_postprocess_description()`

This method should return a string to be used in header descriptions in the CQPweb visual interface.

This is best phrased as a past participial clause. For example, "manipulated by the Jones system", or "reduced in an arbitrary way", or something else. It should be compatible with the description returned by `->get_label()`, as both are shown to the user.

Note that this function will often be called on a *separate* instance of the object to the one that does the actual postprocessing. So, you cannot include particulars about a specific run of the postprocessor. It can only be a generic description.

If the empty string is returned, then this postprocess will not be mentioned in the header interface - but the reduced number of hits will still be shown (it has to be, to make sense). So on balance, it's better to say something!

9.5.4 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 back-end 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.

```
/**
* 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);
```

9.6 Permissions for plugin users

⟪write this!⟫ **TODO**

10 Using the CQPweb API

10.1 Introduction

Use of the CQPweb API is, strictly speaking, not a system administration topic. However, effective use of the API requires an in-depth knowledge of how the system works, and so it is convenient to include the topic in this manual.

11 Updating CQPweb

11.1 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 4.10. If you are upgrading a very old version of CQPweb, you may need to perform some updates manually; see section 11.2 below.

Updating the configuration file rarely needs to be done, as we make efforts to keep it consistent. So far the only change in the format of the configuration file has been between 3.0.16 and 3.1.0. 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.

11.2 Updating the database from very old versions

Earlier 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 MySQL 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.

11.3 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 11.1).

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 4.10.

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 4.6)
- Restore your previous privileges using `load-pre-3.1-privileges.php` (see section 4.7)
- 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 1.8 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.

11.4 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.

11.5 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*).

11.6 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 11.1).

Step 3. Update the database using the upgrade script described in section 4.10.

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 4.8. 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 MySQL database.
- Second, it attempts top 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 MySQL 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 direcrtory 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 1.3).

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 inadvisible* 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.

11.7 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.

11.8 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.

