The IMS Open Corpus Workbench (CWB)
Corpus Encoding and Management Manual

— CWB Version 3.5 —

Stephanie Evert & The CWB Development Team
http://cwb.sourceforge.net/

July 2022

Contents

1 Prerequisites

In order to follow the examples given in this manual, you need to install the **IMS Open Corpus Workbench (CWB)**, version **3.5**, which can be downloaded from

http://cwb.sourceforge.io/download.php

It is easiest to install a pre-compiled **binary package**, following instructions on the Web page and in the enclosed `README` file.

If you are running CWB on a Unix-like operating system (Linux, Mac OS, or the Windows Subsystem for Linux), you should also install the **CWB/Perl interface**, which includes the useful `cwb-make` and `cwb-regedit` programs. Unfortunately CWB/Perl is not available for native Windows.

A **data package** with all input files needed for the example commands in this manual is available from

http://cwb.sourceforge.io/documentation.php

2 First steps: Encoding and indexing

The standard CWB **input format** is one-word-per-line text,[1] with the surface form in the first column and token-level annotations specified as additional `TAB`-separated columns. XML **tags** for sentence boundaries and other structural annotation must appear on separate lines. This file format is also called **verticalized text** and has the customary file extension `.vrt`. An example of the verticalized text format for a short sentence with part-of-speech and lemma annotations is shown in Figure 1. This file, as well as all other input files required by the following examples are made available in the accompanying **data package**.

```
<s>
It        PP      it
was       VBD     be
an        DT      an
elephant  NN      elephant
.         SENT    .
</s>
```

Figure 1: Verticalized text file *example.vrt*

In order to encode the file as a corpus, follow these steps:

[1] Or, more precisely, one token per line; i.e., CWB expects punctuation marks, parentheses, quotes, etc. on separate lines. The precise **tokenization rules** depend on your theoretical assumptions and the requirements of annotation software such as part-of-speech taggers. CWB does *not* include any components for any kind of tagging, and has to be provided with a tokenized and annotated corpus.

- Create a **data directory** where files in the binary CWB format will be stored. Here, we assume that this directory is called **/corpora/data/example**.[2] If this directory already exists and contains corpus data (from a previous version), you should delete all files in the directory. **NB:** You need a separate data directory for each corpus you want to encode.

- Choose a **registry directory**, where all encoded corpora have to be registered to make them accessible to the CWB tools. It is recommended that you use the **default registry** directory. This varies depending on your operating system and method of installation. The most common locations[3] of the default registry are:

  - On Unix, installed by compiling from source: **/usr/local/share/cwb/registry**
  - On Unix, installed via package manager: **/usr/share/cwb/registry**
  - On Windows, depends on your choice at install time, but most usually something like **C:\Program Files\CorpusWorkbench\Registry**

  If you don't use the default registry, you will have to specify the path to your registry directory with a `-r` flag whenever you invoke one of the CWB tools (or set an appropriate environment variable, see below). In the example commands in this manual, we assume that you use the standard registry directory.

- The next step is to **encode** the corpus, i.e. convert the verticalized text to CWB binary format with the `cwb-encode` tool. Note that the command below has to be entered on a single line.

  ```
  $ cwb-encode -d /corpora/data/example
               -xsBC9 -c ascii -f example.vrt
               -R /usr/local/share/cwb/registry/example
               -P pos -P lemma -S s
  ```

  (The `$` character indicates a command line to be entered into your terminal. It is inspired by the customary input prompt used by the Bourne shells `sh` and `bash`.)

The first column of the input file is automatically encoded as the default **positional attribute** (**p-attribute**) named `word`. `-P` flags are used to declare additional p-attributes, i.e. token-level annotations. `-S` flags declare **structural attributes** (**s-attributes**), which encode *non-recursive* XML tags and whose names must correspond to the XML element names. By convention, all attribute names must be *lowercase* (more precisely, they may only contain the characters `a-z`, `0-9`, `-`, and `_`, and may not start with a digit). Therefore, the names of XML elements to be included in the CWB corpus must not contain any non-ASCII or uppercase letters.

The `-R` option automatically creates a **registry file**, whose filename has to be written in *lowercase*. Note that it is necessary to specify the full path to the registry file, even if the default registry directory is used. The **CWB name** of the corpus (also called the corpus ID) is identical to the name of the registry file, but is written in *uppercase* (here it will be `EXAMPLE`). The CWB name is used to activate a corpus in the query processor CQP, for instance.

`-xsBC9` is a cluster of options which switch on data cleanup procedures. They are, in order: recognise and handle basic XML features (`-x`); ignore any empty lines (`-s`); tidy up stray blank space characters (`-B`); remove characters that are invalid for the specified encoding (`-C`); silently discard unrecognised XML tags (`-9`). Most of the time, you would want to use all of these; the only time to omit them is

[2] The filesystem paths referred to in this manual are all Unix-style; however, CWB on Windows works happily with Windows-style paths.

[3] In previous versions of CWB, the default registry directory used to be **/corpora/c1/registry** (for historical reasons). All binary packages of CWB 3.0 and newer use the new default setting. If you already have a working environment with the old registry path, you may want to compile the CWB source code yourself, selecting the `classic` site configuration.

when you are working with files that you *know* have no encoding or formatting problems (or if you use the new `-n` or `-N` formats; see section 3). Using `-x` and `-9` does not preclude more complex XML; see section 5.

The `-c` option specifies the character encoding (or *charset*) of the input data. The *example.vrt* file does not contain any non-ASCII characters, so in this example we specify `-c ascii`. The other commonly used charsets are Unicode UTF-8 (`-c utf8`) and ISO 8859-1(`-c latin1`). We strongly recommend use of UTF-8 over ISO 8859 charsets wherever possible. A full list of charsets supported by CWB, and the corresponding single word labels used with the `-c` option, is available in the manual file for `cwb-encode`.[4]

Input files with the extensions `.gz`, `.bz2` or `.xz` are assumed to be in the gzip, bzip2 and xz compressed formats, respectively. Such files are automatically decompressed (provided that `gzip`, `bzip2` and/or `xz` are available).[5]

Multiple input files can be specified by using the `-f` option repeatedly. Files will be read in the order in which they appear on the command line. Shell wildcards (e.g. `-f *.txt`) *do not* work, since each file name must be preceded by `-f`. However, it is possible to read *all* files named `*.vrt`, `*.vrt.gz`, `*.vrt.bz2` or `*.vrt.xz` in a given directory using the `-F` option (possibly repeated for multiple directories). The input files in each specified directory will be read in alphabetical order.

All options (`-d`, `-f`, `-R`, etc.) *must precede* the attribute declarations (`-P`, `-S`, etc.) on the command line. It is mandatory to specify a data directory with the `-d` option.[6] This directory should always be given as an *absolute* path, so the corpus can be used from any location in the file system.

Before a corpus can be used with CQP and other CWB programs, various **index files** have to be built. It is also strongly recommended to **compress** these index files, especially for larger corpora:

- The easiest and recommended method for indexing and compression is to use the **cwb-make** script that comes with the **CWB/Perl** interface modules. If you are unable to install the modules and use this script (e.g. if you are using the Windows version of CWB), refer to Section 4 for a manual procedure.

  ```
  $ cwb-make -V EXAMPLE
  ```

- If you did not use the standard registry directory **/usr/local/share/cwb/registry** when running `cwb-encode`, you will have to specify the path to your registry directory with the `-r` option. Alternatively, you can set the environment variable `CORPUS_REGISTRY`, which is automatically recognized by all CWB programs. In a Bourne shell (`sh` or `bash`), this is achieved with the command

  ```
  $ export CORPUS_REGISTRY=/home/stephanie/registry
  ```

  In a C shell (`csh` or `tcsh`), the corresponding command is

  ```
  $ setenv CORPUS_REGISTRY /home/stephanie/registry
  ```

[4] Older versions of CWB - including the long-term "stable" 3.0 - only fully supported ISO 8859-1. While it is possible, just about, to work with other charsets in CWB 3.0, it is *very strongly* recommended that you upgrade to **CWB version 3.4** to get full support for all ISO-8859-x encodings as well as UTF-8. While as late as the mid-2010s, there were corpus annotation programs in wide use that generated ISO 8859 output, as of this writing UTF-8 is now finally the accepted standard, and the recommended encoding for use with CWB. Nevertheless, `cwb-encode` still defaults to Latin-1 (for backward compatibility with 3.0) if no `-c` option is supplied; it is for this reason that we recommend always specifying the charset explicitly.

[5] By "available" we mean that the program in question must be both installed on your computer, and findable to CWB. See Appendix C.

[6] Previous versions of the CWB would default to the current working directory in the absence of a `-d`. As a result, simply typing `cwb-encode` on the command line would litter this directory with a number of empty data files and then hang, waiting for corpus data on the standard input.

In either case, it is probably a good idea to add this setting to your login profile (`~/.profile` or `~/.login`). If you do not want to set the environment variable, you need to invoke `cwb-make` with

```
$ cwb-make -r /home/stephanie/registry -V EXAMPLE
```

On Windows, as noted above,`cwb-make` is not available, as it is part of CWB/Perl. However the same methods of setting the registry apply to use of the uitilities discussed in Section 4. Environment variables can be set persistently in Windows by going to the Settings app; clicking on "find a setting"; typing "environment"; and selecting "Edit environment variables". In the interface that pops up, open the environment variables dialogue, and add a new variable with the name `CORPUS_REGISTRY` and the path to your registry as its value. To set the registry temporarily in a terminal session, use this command:

```
$ set CORPUS_REGISTRY=C:\Users\stephanie\registry
```

The following examples assume that you either use the default registry directory or have set the `CORPUS_REGISTRY` variable appropriately.

- You can also specify multiple registry directories separated by colon characters (`:`), both in the `CORPUS_REGISTRY` environment variable and the `-r` options of command-line tools. This is convenient e.g. if some corpora are stored on external hard drives that are not always mounted. Such *optional* registry directories may be prefixed by a question mark (`?`) in order to indicate that they may not be accessible (otherwise CQP and some other tools will print warnings to alert you to possible typos in the registry path). For instance, one of the lead CWB developers has the following registry path in his `~/.bashrc` configuration:

  ```
  $ export CORPUS_REGISTRY=/Corpora/registry:?/Volumes/X/CWB/registry
  ```

  The built-in default registry directory is **not** automatically appended to this path. If you want to specify additional registry directories but keep the default one, you need to include the default location explicitly in the value of `CORPUS_REGISTRY`.

The `-V` switch enables additional validation passes when an index is created and when data files are compressed. It should be omitted when encoding very large corpora (above 50 million tokens), in order to speed up processing. In this case, it is also advisable to limit memory usage with the `-M` option. The amount specified should be somewhat less than the amount of physical RAM available (depending on the number of users etc.; too little is better than too much). For instance, on a Linux machine with 8 GiB of RAM, `-M 2048` is a safe choice. The `cwb-make` utility applies a default limit of `-M 75` if no explicit `-M` option is given, which is unreasonably small for current hardware, being optimised for machines of the last millennium.

- Use the `cwb-describe-corpus` utility to display some information about an encoded corpus (add the `-s` option for details and to reassure yourself that all necessary data files have been created):

  ```
  $ cwb-describe-corpus EXAMPLE
  ```

3 Input format extensions

Recent versions of CWB have added extended options for the format of the input files.

As of CWB v3.4.37, `.xz` is now a supported compression format in addition to `.gz` and `.bz2` (as long as the relevant program, or 7-zip, is available), and compressed files are accepted for input and output

by CQP and all CWB command-line tools.[7] Moreover, it is possible to read from or write to shell pipes in these versions, by specifying a quoted filename that starts with a pipe character (|).

As of CWB v3.4.27, an alternative input format can be activated with the `-n` option, which requires all token lines to be numbered in the first TAB-separated column (see Fig. 2). The numbering itself is ignored but helps to make an unambiguous distinction between XML tags and token lines. This is a useful and robust alternative to encoding metacharacters as XML entities (see section 5), which many other command-line tools do not process correctly. The options `-n` and `-x` can safely be combined.

```
<s>
1       The     DT      the
2       tag     NN      tag
3       <s>     SYM     <s>
4       is      VBZ     be
5       useful  JJ      useful
6       !       SENT    !
</s>
```

Figure 2: Verticalized text file *example-numbered.vrt* in `-n` format

As of CWB v3.4.28, the token numbers in the first column can be captured in an additional p-attribute (e.g. `id`) by using the `-N` option instead of `-n` (e.g. `-N id`). In either case, comment lines starting with a hash (`#`) are now ignored silently. The attribute declared with `-N` will always be the first p-attribute in the registry file, followed by the first attribute declared with `-p` or the default `word` attribute.

CWB v3.4.28 also introduces support for processing empty lines as sentence breaks with the `-L` option,[8] which implies `-s`. The segmentation is stored in a user-specified s-attribute, e.g. `-L s`. Note that this is a special "hidden" attribute, so explicit `<s>` and `</s>` tags will be treated as unknown elements. The combination of both options makes it possible to encode input files in one of the **CoNLL** formats[9] without additional pre-processing:

```
$ cwb-encode -N id -L s -f conll.vrt ...
```

Several caveats apply:

- CWB does not recognise any specific CoNLL flavour, i.e. all columns (except for the token numbers in the first column) have to be declared explicitly as p-attributes.
- Multiword tokens (labelled with a number range, e.g. `3-4`) and empty tokens (labelled e.g. as `5.1`) are silently discarded.
- All comment lines are discarded, even the special notation used for text structure boundaries and metadata in CoNLL-U (which is entirely misguided, of course). Such metadata comments can easily be converted into XML tags in a pre-processing step and encoded as described in Sec. 5.
- All annotation columns are encoded as-is into positional attributes. Dependency relations or phrases in bracketing notation are not transformed into graph or tree structures (which are not supported by CWB 3); chunks in IOB notation are not expanded into a structural attribute.
- CoNLL feature set notation can be transformed into CWB syntax, but this has to be requested explicitly on the command line for each attribute, e.g. `-P morph/` (see further Sec. 6). Sets will also be re-sorted in alphabetical order with this option.

[7] Previous CWB versions had partial support for gzip-compressed input and output files, indicated in the respective man pages.

[8] Mnemonic: `-L` stands for sentence **L**imits.

[9] see e.g. https://universaldependencies.org/format.html and format examples on this page

4 Indexing and compression without CWB/Perl

If you do not have the CWB/Perl interface installed, the best thing you can do is to install the CWB/Perl modules and the scripts it includes, and then go back to Section 2. If it is impossible to install CWB/Perl (for example, on Windows), or if you really want to learn the nitty-gritty of corpus encoding, continue here.

- In the manual procedure, indexing and compression are performed in separate steps by different tools. First, you have to run `cwb-makeall` in order to build the necessary index files.

  ```
  cwb-makeall -V EXAMPLE
  ```

  `cwb-makeall` accepts the same `-V`, `-M` and `-r` options as `cwb-make`. The comments on enabling/disabling validation given above with regards to `cwb-make` naturally apply to `cwb-makeall` as well.

When the index files have been created, the corpus can already be used with CQP and other CWB tools. However, it is recommended that you compress the binary data files to save disk space and improve performance. For very small corpora (under 10 million tokens) the compression won't make a lot of difference; for corpora larger than that, it probably will. Compression is only supported for p-attributes at present.

- For positional attributes, both the token stream data and the index can be compressed. There are separate tools for compressing the two types of data files.
- The token stream can be compressed with the `cwb-huffcode` tool. Use the `-P` option to process a single attribute, or compress all p-attributes with `-A`.

  ```
  $ cwb-huffcode -A EXAMPLE
  ```

- Index files can be compressed with the `cwb-compress-rdx` tool, which accepts the same options.

  ```
  $ cwb-compress-rdx -A EXAMPLE
  ```

When compression was successful, both tools will print the full pathnames of uncompressed data files that are now redundant and can be deleted: *attrib*`.corpus` after running `cwb-huffcode`; *attrib*`.corpus.rev` and *attrib*`.corpus.rdx` after running `cwb-compress-rdx`.

If you run `cwb-makeall` again, it will show now that the p-attributes are compressed. The compressed data files are validated by default, so it is safe to remove the redundant files.

Validation can be turned off in both `cwb-huffcode` and `cwb-compress-rdx`, using the `-T` option (T for *trust*). However, letting validation run causes much less performance problems than can arise from validation with `cwb-makeall`.

- **NB:** If you re-encode a corpus, it is important to *erase all files* in the data directory first. The `cwb-makeall` program will not recognize that existing index files or compressed data files are out of date, and will therefore fail to rebuild them automatically. (This is one of the reasons why the CWB/Perl `cwb-make` tool should be preferred.)

5 CWB corpora and XML

Nowadays, machine-readable text and linguistic annotations are often provided in **XML format**. CWB's XML support is activated by the following encoding options: `-x` for XML compatibility mode (recognises default entities and skips comments as well as an XML declaration), `-s` to skip blank lines in the input, and `-B` to strip whitespace from tokens. All three options `-xsB` should (almost) always be used.[10] The vertical text format with `TAB`-separated p-attributes is still required by `cwb-encode`, but this format can easily be generated from an arbitrary XML file with the aid of a little script in any suitable language. Figure 3 shows a typical example of an XML input file for the CWB, including an XML declaration and a comment line that options `-xsB` will cause to be ignored. Note that despite the use of tabs for columns, this is *still* a well-formed XML file.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<!-- A Thrilling Experience -->
<story num="4" title="A Thrilling Experience">
<p>
<s>
Tick    NN      tick
.       SENT    .
</s>
<s>
A       DT      a
clock   NN      clock
.       SENT    .
</s>
<s>
Tick    VB      tick
,       ,       ,
tick    VB      tick
.       SENT    .
</s>
</p>
...
</story>
```

Figure 3: Verticalized XML file *vss.vrt*

XML elements (i.e. matching pairs of start and end tags) can be encoded as s-attributes, which have to be declared with `-S` flags (for the file *vss.vrt*, the flags `-S story -S p -S s` would be used). If XML regions of the same type are **nested**, encoding will only work correctly if you add `:0` to the s-attribute declaration, which enables a rudimentary XML parser built into `cwb-encode`. **Attribute-value pairs** in XML start tags, such as

```
<story num="4" title="A Thrilling Experience">
```

can be stored as a single unparsed text string (`num="4" title="A Thrilling Experience"`) by using the flag `-V` instead of `-S`. This form of encoding is not convenient for CQP queries, though. It is more desirable to declare XML tag attributes explicitly; doing so will automatically split the XML elements into multiple s-attributes.

[10] Along with the `-C` option for charset cleanup; see section 2.

- Encode the verticalized XML file *vss.vrt* as a CWB corpus, with indexing and compression. **NB:** The last attribute declaration flag (`-0 collection`) is a *digit zero* (for a "null attribute", see below).

```
$ cwb-encode -d /corpora/data/vss -f vss.vrt
             -R /usr/local/share/cwb/registry/vss
             -xsBC -c ascii -P pos -P lemma
             -S s:0 -S p:0 -S story:0+num+title -0 collection

$ cwb-make -V VSS
```

  If you do not have the `cwb-make` script available, follow the steps in Section 4.

These commands will encode the corpus `VSS` and create a registry file, including the s-attributes `s`, `p`, `story`, `story_num`, and `story_title`. The `<story>` start tags are parsed and the attribute values are stored as annotations of the attributes `story_num` (value: `4`) and `story_title` (value: `A Thrilling Experience`). Regions of the `story` attribute itself will not be annotated. Use `-V` instead of `-S` to store all attribute-value pairs as a single string, which can be useful for displaying and re-exporting the XML tags.

XML elements with different names (such as `<s>` and `<p>`) are encoded independently, so they can nest and overlap in arbitrary ways. The `cwb-encode` program does not perform any validation or well-formedness tests on the XML elements. When elements are nested recursively (e.g. a `<table>` within a `<table>`), the embedded elements will be ignored, because of the use of `:0` specified above. After encoding, `cwb-encode` prints a summary listing the number of dropped XML elements. If you instead want to preserve nested elements, you can specify a maximal level of embedding instead of `:0` in the examples above. For instance, `-S table:2` allows two levels of embedding for `<table>` elements. Nested elements are automatically renamed to `<table1>` and `<table2>`, respectively, and stored in separate s-attributes.

Sometimes, the input data may contain XML tags that should not be encoded in the corpus. For instance, the stories in *vss.vrt* have to be wrapped in a single root element `<collection>` in order to obtain a well-formed XML file. Instead of removing such tags during data preparation, they can directly be filtered out by the `cwb-encode` tool. For this purpose, they have to be declared with the flag `-0` (digit zero, for "null attribute") instead of `-S` or `-V`. All start and end tags of these elements will be ignored completely. There is no need to add `:0` or XML attribute declarations. Note that all XML tags that have not been declared with a `-S`, `-V` or `-0` flag will be encoded as literal tokens (that is, words, without annotations), accompanied by a warning message.

Starting with CWB 3.4.21, unknown XML tags can automatically be declared as null attributes with the `-9` ("auto-null") option. This is recommended to capture the correct token stream for an input file with very many and/or undocumented XML elements:

```
$ cwb-encode -d /corpora/data/vss -f vss.vrt
             -R /usr/local/share/cwb/registry/vss
             -xsBC -c ascii -9 -P pos -P lemma
```

You may have noticed in Figure 3 that the XML file is declared to be in **ISO 8859-1** (or **Latin-1**) encoding rather than the standard UTF-8 format. CWB ignores this, along with the rest of the XML declaration. The charset still needs to specified on the command line; here, it is `-c ascii`, since we know there are no non-ASCII characters in this particular file; see also section 2.

6 Adding attributes to an encoded corpus

In order to **add positional attributes** to a corpus that has already been encoded, create input data in the standard verticalized format, but listing only the new attributes. Figure 4 shows an example of such an input file, containing WordNet synonyms for the tokens from Figure 1 (without attempting any form of word sense disambiguation). A corresponding list of synonyms for the complete `VSS` corpus can be found in the file *syns.vrt*.

```
|
|be|cost|live|work|equal|exist|occur|...|
|
|elephant|
|
```

Figure 4: WordNet synonyms for the text shown in Figure 1 (excerpt from file *syns.vrt*)

The special notation seen in Figure 4 indicates that the synonyms for any given word constitute an unordered **set** (or **feature set** in CWB terminology). Vertical bars (`|`) separate individual set elements and enclose the entire set; a single bar `|` denotes the empty set. Feature sets are stored as plain strings in a CWB-encoded corpus, but the special notation enables the query processor CQP to test whether a particular string is contained in the set, match all set elements against a regular expression, and compute the intersection of two sets.

- The file *syns.vrt* is encoded as usual, but the default `word` attribute has to be *suppressed* with the option `-p -`. It is highly recommended to check that the number of tokens in the new file (find this out on Unix with the command `wc -l syns.vrt`) is equal to the corpus size (as reported by `cwb-lexdecode -S EXAMPLE`), so that the new attribute is properly aligned to the rest of the corpus.

  ```
  $ cwb-encode -d /corpora/data/vss -f syns.vrt -p - -P syn/
  ```

  Notice the slash (`/`) appended to the attribute name `syn`. This notation indicates that the new attribute should be treated as a feature set; `cwb-encode` will automatically validate and normalise the supplied values, issuing warnings if they are not well-formed feature sets.[11]

- As of CWB v3.4.28, `cwb-encode` is more lenient with the feature set format, also accepting input without the leading and trailing `|`, e.g. `baggage|luggage` and `elephant` (for a single-member set). An empty string[12] or single underscore (`_`) is interpreted as an empty set. This change was introduced to provide better support for CoNLL-style set notation (also used e.g. by TreeTagger lemmas), which can now be encoded without a pre-processing step. As a consequence, there will no longer be a warning if an attribute is mistakenly declared as a feature set (e.g. `-P word/`); the values will silently be transformed into single-item sets.

- The registry file for the corpus `VSS` (which you will find in the registry folder specified when it was encoded, or if none was specified, the default registry) now needs to be edited to add a declaration of the new attribute. Add the line

  ```
  ATTRIBUTE syn
  ```

  at the bottom of the file. If the CWB/Perl interface has been installed, the registry file can also be edited from the command line with the `cwb-regedit` registry editor script:

[11] A feature-set attribute that is not declared as such at index-time *can* still be treated as a feature set in CQP, but in this case responsibility is with the user to ensure that the values are well-formed feature sets.

[12] Keep in mind that the option `-U ""` has to be specified in this case in order to allow empty strings as values.

```
$ cwb-regedit VSS :add :p syn
```

This script can also be used to list and delete attributes, and to print basic information about a corpus (similar to `cwb-describe-corpus`, but easier for further processing). Type `cwb-regedit -h` for further information.

- Now you can build index files and compress the new attribute:

```
$ cwb-make -V VSS
```

or

```
$ cwb-makeall -V VSS syn
$ cwb-huffcode -P syn VSS
$ cwb-compress-rdx -P syn VSS
```

In order to **add structural attributes** with computed start and end points (**corpus positions**), you can use the `cwb-s-encode` tool. The corresponding start and end positions of existing s-attributes can be obtained with `cwb-s-decode`. The following example adds information about sentence length to the `VSS` corpus.

- The existing `s` attribute is decoded into a temporary file, then `awk`[13] is used to compute sentence lengths, and the resulting annotated regions are encoded with `cwb-s-encode`.

```
$ cwb-s-decode VSS -S s > s.list
$ awk 'BEGIN { FS=OFS="\t" }  { print $1, $2, $2-$1+1 }' s.list > s_len.list
$ cwb-s-encode -d /corpora/data/vss -f s_len.list -V s_len
```

Note that it is currently *not necessary* to run `cwb-make` after adding an s-attribute.

- However, the new attribute still has to be declared in the registry file, either by manually adding

```
STRUCTURE s_len
```

or from the command line using the registry editor script:

```
$ cwb-regedit VSS :add :s s_len
```

Tables of corpus positions as input for `cwb-s-encode` can also be created from CQP query results using the `dump` or `tabulate` command in a CQP session.

7 Adding XML annotations

In order to **add XML annotations** (e.g. `<np>` and `<pp>` tags inserted by a chunk parser) to an existing corpus, the usual strategy is to decode the token stream (and other attributes if necessary) to a temporary file. A chunk parser will often expect `<s>` and `</s>` tags marking sentence boundaries.

- Decode token stream (word forms) with start and end tags for `<s>` regions.

```
$ cwb-decode -C VSS -P word -S s > word_s.vrt
```

- We then run the chunk parser on the temporary file. The chunk parser adds its `<np>` and `<pp>` tags to the token stream, creating the file shown in Figure 5. This file is also provided as part of the **data package** for this manual.

[13] `awk` is a standard Unix tool, not available on Windows by default, and not to our knowledge easy to install. On Windows, therefore, you would need to use some other program to process the corpus position data.

```
<s>
<np head="experience>
My
experience
<pp head="of">
of
<np head="life">
life
</np>
</pp>
</np>
did
not
...
</s>
```

Figure 5: Decoded text with chunk annotations (file *chunks.vrt*)

- It is important that the token stream is left intact when adding XML annotations. In particular, tokens (as well as XML tags) must remain on separate lines and may not be split or combined. As a preliminary check, make sure that the number of tokens in *chunks.vrt* is equal to the corpus size. On Unix, the `grep` and `wc` utilities can be used for this:

```
$ grep -v '^<' chunks.vrt | wc -l
```

Now we can use `cwb-encode` to encode the XML annotations as structural attributes. The start and end points of regions are automatically computed from the token stream. Since we do not want to overwrite the `word` attribute, we specify `-p -`. With no p-attributes declared, all lines in the input file except for the XML tags will be ignored. Recall that `-0 s` (digit zero) instructs `cwb-encode` to ignore `<s>` and `</s>` tags (without `-S s` they would otherwise be interpreted as literal tokens and mess up the token stream).

- Encode `<np>` and `<pp>` regions in *chunks.vrt* as new s-attributes:

```
$ cwb-encode -d /corpora/data/vss -f chunks.vrt
             -p - -0 s -S np:0+head -S pp:0+head
```

In this example, `cwb-encode` will issue warnings about nested regions being dropped. As can be seen from Figure 5, `<np>` (as well as `<pp>`) regions may be embedded recursively. In order to preserve such nested regions, change the `:0` modifier to `:2`, allowing up to two levels of embedding (separately for each element type, i.e. `<np>` regions embedded in larger `<np>` regions, etc.). In general, `:`n allows up to n levels of embedding. The embedded regions will automatically be renamed to `np1`, `np2`, `pp1`, and `pp2`, respectively.

- Encode *chunks.vrt*, allowing up to two levels of embedding for `<np>` and `<pp>` regions:

```
$ cwb-encode -d /corpora/data/vss -f chunks.vrt
             -p - -0 s -S np:2+head -S pp:2+head
```

- The full list of s-attributes created by this command is `np`, `np1`, `np2`, `np_head`, `np_head1`, `np_head2`, `pp`, `pp1`, `pp2`, `pp_head`, `pp_head1`, and `pp_head2`. They all have to be declared in the registry file of the corpus `VSS`, either by adding the appropriate entries manually, or with the registry editor script:

```
$ cwb-regedit VSS :add :s np np1 np2 np_head np_head1 np_head2
$ cwb-regedit VSS :add :s pp pp1 pp2 pp_head pp_head1 pp_head2
```

- Attribute-value pairs in XML start tags may contain feature sets, just as is possible for p-attributes. For instance, the German chunk parser YAC[14] uses this notation to represent partially disambiguated morphological features of NPs and PPs (see the CQP Query Language Manual for more information and examples). XML tags of the form

  ```
  <np agr="|Nom:F:Sg|Acc:F:Sg|" head="Wiese">
  ```

  might be encoded with the declaration `-S np:2+agr/+head`, where the slash `/` indicates that `agr` values are feature sets. Since `head` is not followed by a slash, the corresponding values are not treated as feature sets.

8 Decoding and analysing corpora

The `cwb-lexdecode` tool provides access to the **lexicon** of positional attributes, i.e. lists of all word forms or annotation strings (*types*) with their corpus frequencies. The `-S` option prints the size of corpus (*tokens*) and lexicon (*types*) only, `-P` selects the desired p-attribute, `-f` shows corpus frequencies, and `-s` lists the lexicon entries alphabetically (according to the internal sort order). In order to sort the lexicon by frequency, an external program (e.g. `sort`) has to be used.

```
$ cwb-lexdecode -S    -P lemma VSS
$ cwb-lexdecode -f -s -P lemma VSS | tail -20
$ cwb-lexdecode -f    -P lemma VSS | sort -nr -k 1 | head -20
```

It is also possible to annotate strings from a file (called *tags.txt* here) with corpus frequencies. The file must be in one-word-per-line format. `-0` (digit zero) prints a frequency of 0 for unknown strings rather than issuing a warning message; it can be combined with `-f` to the mnemonic form `-f0`.

```
$ cwb-lexdecode -f0 -P pos -F tags.txt VSS
```

With the `-p` option, word forms or annotations matching a regular expression can be extracted. Case-insensitive and accent-insensitive matching is selected with `-c` and `-d`, respectively. The example below is similar to the CQP query `[lemma = "over.+" %c];` but may be considerably faster on a large corpus.

```
$ cwb-lexdecode -f -P lemma -p "over.+" -c VSS
```

An **entire corpus** or selected attributes from a corpus can be printed in various formats with the `cwb-decode` tool. Note that options and switches must appear *before* the corpus name, and the flags used to select attributes *after* the corpus name. Use `-P` to select p-attributes and `-S` for s-attributes. With the `-s` and `-e` options, a part of the corpus (identified by start and end corpus position) can be printed.

```
$ cwb-decode -C -s 7299 -e 7303  VSS  -P word -P pos -S s
```

`-C` refers to the compact one-word-per-line format expected by `cwb-encode`. For a full textual copy of a CWB corpus, use `-ALL` to select all positional and structural attributes.

```
$ cwb-decode -C  VSS  -ALL  > vss-corpus.vrt
```

[14] See Kermes and Evert (2002): https://www.aclweb.org/anthology/L02-1202/

The resulting file *vss-corpus.vrt* can be re-encoded with `cwb-encode` (using appropriate flags) to give an exact copy of the `VSS` corpus. `-Cx` is almost identical to the compact format, but changes some details in order to generate a well-formed XML document (unless there are overlapping regions or s-attributes with "simple" annotations).[15]

```
$ cwb-decode -Cx  VSS  -ALL  > vss-corpus.xml
```

This output format can reliably be re-encoded if the `-xsB` options are used (see section 5).

As of CWB v3.4.33, the opposite round-trip is also supported, i.e. it is possible to **reconstruct a *.vrt* input file** almost exactly. To this end, nested XML regions and attribute-value pairs in start tags, which have been broken up into separate s-attributes by `cwb-encode` as described in Sec. 5, need to be recombined by giving corresponding `-S` specifications to `cwb-decode`.

- Reconstruct the file *vss.vrt* (Fig. 3) from the CWB corpus `VSS` indexed in Sec. 5:[16]

```
$ cwb-decode -C VSS -P word -P pos -P lemma
             -S s -S p -S story+num+title > vss_decoded.vrt
```

- Also decode the nested NP and PP elements added in Sec. 7 with these additional declarations:

```
$ cwb-decode -C VSS -P word -P pos -P lemma
             -S s -S p -S story+num+title -S np:2+head -S pp:2+head
             > vss_decoded.vrt
```

Finally, `-X` produces a native XML output format (following a fixed DTD), which can be post-processed and formatted with XSLT stylesheets.

```
$ cwb-decode -X -s 7299 -e 7303  VSS  -P word -P pos -S s -S np_head
```

Note that the regions of s-attributes are not translated into XML regions. Instead, the start and end tags are represented by special empty `<tag>` elements.

As of CWB v3.4.28, the `cwb-encode` and `cwb-decode` utilities provide improved support for reading and writing **CoNLL-style formats**; see section 2 for details and limitations. Section 3 covers how to index CoNLL files. Such a corpus can easily be decoded back into CoNLL format, with the option `-b s` adding a blank line after each sentence region:

```
$ cwb-decode -C -b s CONLL_CORPUS -P id -P word -P pos ...
```

If token numbers haven't been indexed explicitly, use numbered output mode (`-Cn`) to insert corpus positions as placeholders in the first output column:

```
$ cwb-decode -Cn CONLL_CORPUS -P word -P pos ...
```

An alternative strategy is to extract all sentence regions and decode them in "matchlist mode", which automatically adds blank lines as delimiters. In this approach, comment lines with metadata information can be added at the start of each sentence using `-V` flags:

[15] In order to re-create the original input file *vss.vrt* as a well-formed XML document, it would have been necessary to store the full strings of attribute-value pairs from XML start tags by using `-V` flags instead of `-S` in the `cwb-encode` attribute declarations (e.g. `-V story:0+num+title`). In the `cwb-decode` call, problematic s-attributes created by auto-splitting of these attribute-value pairs (`story_num`, `story_title`, `s_len`, `np_head`, ...) can then be omitted. The specification `-S story` would print the full attribute-value pairs in `<story>` tags, etc.

[16] There will be a few small differences due to escaping of XML metacharacters and the omitted `collection` attribute.

```
$ cwb-s-decode CONLL_CORPUS -S s |
  cwb-decode -Cn -p CONLL_CORPUS -P word -P pos ... -V text_id -V s_num
```

It is then also possible to decode a subset of the sentences, by running a suitable CQP query (with `expand to s`) and dumping the corresponding corpus positions. See `man cwb-decode` for examples.

`cwb-scan-corpus` computes **combinatorial frequency tables** for an encoded corpus. Similar to the `group` command in CQP, it is a faster and more memory-efficient alternative for the extraction of simple structures from large corpora, and is not restricted to singletons and pairs. The output of `cwb-scan-corpus` is an unordered list of n-tuples and their frequencies, which have to be post-processed and sorted with external tools. The simple example below prints the twenty most frequent (`lemma`, `pos`) pairs in the `VSS` corpus, using the `-C` option to filter punctuation and noise from the list of lemmata (note that `-C` applies to *all* selected attributes). [17]

```
$ cwb-scan-corpus -C VSS lemma pos | sort -nr -k 1 | head -20
```

A non-negative **offset** can be added to each field key in order to collect **bigrams**, **trigrams**, etc. The following example derives a simple language model in the form of all sequences of three consecutive part-of-speech tags together with their occurrence counts. Only the twenty most frequent sequences are displayed.

```
$ cwb-scan-corpus VSS pos+0 pos+1 pos+2 | sort -nr -k 1 | head -20
```

For a large corpus such as the `BNC`, the scan results can directly be written to a file with the `-o` switch. If the filename ends in `.gz`, `.bz2` or `.xz` (such as the file *language-model.gz* in the example below), the output file is automatically compressed (subject to the caveats discussed in Sec. 2).

```
$ cwb-scan-corpus -o language-model.gz BNC pos+0 pos+1 pos+2
```

The values of the selected p-attributes can also be filtered with regular expressions. The following command identifies part-of-speech sequences at the end of sentences (indicated by the tag `SENT` = sentence-ending punctuation).

```
$ cwb-scan-corpus VSS pos+0 pos+1 pos+2=/SENT/ | sort -nr -k 1 | head -20
```

Since the third key is used only for filtering, we can suppress it in the output by marking it as a constraint key with the `?` character.

```
$ cwb-scan-corpus VSS pos+0 pos+1 ?pos+2=/SENT/ | sort -nr -k 1 | head -20
```

`cwb-scan-corpus` can operate both on p-attributes and on s-attributes with annotated values. For instance, to obtain by-story frequency lists for the `VSS` corpus, use the following command:

```
$ cwb-scan-corpus -o freq-by-story.tbl VSS lemma+0 story_title+0
```

As a special case, s-attributes without annotated values can be used to restrict the corpus scan to regions of a particular type. For instance, the constraint key `?footnote` would only scan `<footnote>` regions. Keep in mind that such special constraints must not include a regular expression part.

The final example extracts pairs of adjacent adjectives and nouns from the `VSS` corpus, e.g. as candidate data for adjective-noun collocations. Constraint keys are used to identify adjectives and nouns, and only nouns starting with a vowel are accepted here. Note the `c` and `d` modifiers (case- and diacritic-insensitive matching) on this regular expression. It is recommended to put all keys with non-trivial constraints in single quotes in order to avoid misinterpretation of shell metacharacters.

[17] Windows users be aware: the data in this command, and some of the subsequent examples, is piped via the Unix tools `sort` and `head`. On Windows a more typical approach would be to redirect the output to file and then use some GUI program (e.g. Notepad++, Microsoft Excel, etc.) to open the file and manipulate the data.

```
$ cwb-scan-corpus -C VSS lemma+0 '?pos+0=/JJ.*/'
                         'lemma+1=/[aeiou].+/cd' '?pos+1=/NN.*/'
```

Except for the `-C` option, this command line is equivalent to the following CQP commands, but it will execute much faster on a large corpus.

```
> A = [pos = "JJ.*"] [pos = "NN.*" & lemma = "[aeiou].+" %cd];
> group A matchend lemma by match lemma;
```

The `cwb-scan-corpus` command is limited to relatively simple constraints on tokens, and it can only match patterns with fixed offsets (but not e.g. determiner and noun separated by an arbitrary number of adjectives). To obtain frequency tables for more complex patterns, use CQP queries in combination with the `tabulate` function. The resulting data tables can be saved to disk and loaded into a relational database or processed with some other software package for statistical analysis.

As of CWB v3.4.26, n-grams can be restricted with the `-w` option to be contained in a single region of a specified s-attribute, similar to the `within` constraint of a CQP query. List the most frequent POS trigrams inside noun phrases with

```
$ cwb-scan-corpus -f 10 -w np VSS pos+0 pos+1 pos+2 | sort -nr
```

(Note that a hidden constraint key is added so that the scan will skip efficiently from the end of one region to the start of the next.)

Only a single `-w` constraint can be specified, but normal existence constraints can be used to restrict the scan further, e.g. to NPs within a PP:

```
$ cwb-scan-corpus -f 10 -w np VSS pos+0 pos+1 pos+2 ?pp+0 | sort -nr
```

This will only work correctly if the `-w` regions are fully contained in the regions tested with existence constrains.

It is also possible to compute **document frequencies** based on an arbitrary s-attribute, using the `-d` option. This command will list all lemmas that occur in all six stories of the `VSS` collection:

```
$ cwb-scan-corpus -f 6 -d story VSS lemma+0
```

The `-d` option automatically enforces a corresponding `-w` constraint. It cannot be combined with an explicit `-w` option (i.e. `-d story -w story` is invalid), nor with the `-F` option for summing over pre-computed frequency counts.

9 Sentence alignment

An alignment between two parallel corpora (e.g. a collection of source texts and their translations into some other language) can be encoded as a corpus attribute within CWB.

- *Alignment attributes* (a-attributes) are unlike other types of attribute because *alignment presupposes the existence of the source and target corpora.* That is, *first* we need to encode the two corpora independently; *then* we can add the alignment attribute that links them.
- Alignment attributes are usually employed for sentence alignment, and we will assume throughout that it is sentences that we are aligning.

- However, you can also align at some other level (e.g. clauses or paragraphs or chapters). Aligning regions that are much smaller than a sentence will not be very useful because of the limitations of how CQP deals with a-attributes.
- Only one a-attribute linking any particular pair of corpora can exist.
- There are two ways that a pair of corpora can be aligned.
  - First, the `cwb-align` tool can be used to automatically align the sentences of the two corpora, with its output subsequently encoded as an a-attribute using `cwb-align-encode`.
  - Second, an existing alignment scheme encoded in the corpus markup can be imported as an a-attribute using `cwb-align-import`.

CWB supports many types of alignment link: one to one, many to many, and crossing. However, the regions in the corpora that are the units to be aligned with one another cannot be discontinuous.

9.1 The example corpora

First, let's introduce the tutorial data we'll be working with. All the files mentioned here are available as part of the **data package** provided alongside the CWB Encoding Manual. The corpus we'll use to practice alignment consists of a very short excerpt from the novel *The Hound of the Baskervilles* by Arthur Conan Doyle, which we'll call the *Holmes* corpus after the main character. As well as the original English, we have a German translation of the same text. We'll use the CWB labels `HOLMES-EN` for the source corpus and `HOLMES-DE` for the target corpus (i.e. the translation) respectively. Using language codes to distinguish components of a parallel corpus in this way is a useful way to organise labels for aligned corpora in CWB.

Before going any further, you should index these two corpora, using the following commands:

```
$ cwb-encode -d /corpora/data/example -c utf8 -f holmes_en.vrt
             -R /usr/local/share/cwb/registry/holmes-en
             -P pos -P lemma -S s+id -S p+num
$ cwb-encode -d /corpora/data/example -c utf8 -f holmes_de.vrt
             -R /usr/local/share/cwb/registry/holmes-de
             -P pos -P lemma -S s+id -S p+num
```

(you should, of course, amend the `-d` and `-R` options to suit your own setup).

All the example commands given in the following sections are based on these two corpora. They do not include the `-r` option to specify the registry directory location. If you have placed the registry files for the two corpora anywhere other than the default registry, you will need either to add the `-r` option, or else to use the `CWB_REGISTRY` environment variable.

9.2 Using the sentence aligner

The `cwb-align` program is a very simple text aligner. It can be considered a "fallback" option for sentence alignment, designed to provide basic functionality when nothing better is available. If your corpus is already aligned, it is always better to use that existing alignment data. Similarly, if you have a properly-designed and trained aligner for a given language pair, it is always better to use that than to rely on `cwb-align`.

```
<p num="3">
<s id="a">
Mr.         NP     Mr.
Sherlock    NP     Sherlock
Holmes      NP     Holmes
[...]
was         VBD    be
seated      VBN    seat
at          IN     at
the         DT     the
breakfast   NN     breakfast
table       NN     table
.           SENT   .
</s>
<s id="b">
I           PP     I
[...]
stood       VBD    stand
upon        IN     upon
the         DT     the
hearth-rug  NN     hearth-rug
and         CC     and
picked      VBD    pick
up          RP     up
the         DT     the
stick       VB     stick
[...]
.           SENT   .
</s>
[... two more sentences ...]
</p>
```

Figure 6: Example from the source corpus (file *holmes_ en.vrt*), with abbreviations

In particular, `cwb-align` will not work well on languages that are unrelated to the extent of sharing little or no vocabulary, as it works by looking for similarities in the words used in the two corpora it analyses.

`cwb-align` makes use of very basic techniques to align units in two parallel corpora by spotting those units - assumed to be of about sentence length - that have similar content. It looks for similarities in terms of:

- The length of each corpus segment, measured in characters.
- The presence of shared words across the two corpora (ignoring case and accents).
- The presence of shared letter sequences (for spotting similar but not identical words).
- The presense of words specified as translation equivalents (a file containing the list of word-pairs must be provided to look for these kinds of similarity).

Here is how we might **create an alignment from scratch and then encode it** using the two `HOLMES` corpora, assuming that the `<s>` elements are the units to be aligned.

The most basic use of `cwb-align` would be as follows:

```
<p num="3">
<s id="a">
Mr.              NN       Mr.
Sherlock         NN       Sherlock
Holmes           NE       Holmes
[...]
saß              VVFIN    sitzen
am               APPRART  an
Frühstückstisch  NN       Frühstückstisch
,                $,       ,
während          KOUS     während
ich              PPER     ich
auf              APPR     auf
dem              ART      die
Kaminvorleger    NN       Kaminvorleger
stand            VVFIN    stehen
und              KON      und
den              ART      die
Spazierstock     NN       Spazierstock
aufhob           VVFIN    aufheben
[...]
</s>
[... three more sentences ...]
</p>
```

Figure 7: Example from the target corpus (file *holmes_ de.vrt*), with abbreviations

```
$ cwb-align -o holmes.align HOLMES-EN HOLMES-DE s
```

This command has one option and three arguments. The `-o` option simply specifies a filename for the output data. The first and second arguments are the labels of the source corpus and the target corpus respectively. The third argument is the *grid attribute*, that is, the s-attribute used as the alignment grid.

The output file has five columns (see figure 8). The first line is a header line with the names of the aligned corpora and of the grid attribute. Each subsequent line specifies a pair of aligned regions:

- The beginning of the region in the source corpus
- The end of the region in the source corpus
- The beginning of the region in the target corpus
- The end of the region in the target corpus
- The type of alignment: 1:1, 2:1, 1:2, 2:2, 1:0 or 0:1 indicating one-to-one, two-to-one, one-to-two or two-to-two, one-to-zero (deletion) and zero-to-one (insertion), respectively
- A number indicating how sure the alignment engine is that this pair of regions really matches (the "quality").[18]

However, it is not normally necessary for a human being to read the file. Usually it is used only as input data for the next step (see below).

[18] More precisely, the quality score represents the weighted sum of features shared by the aligned regions. Therefore long alignment beads will usually achieve higher scores than shorter beads.


```
HOLMES-EN s         HOLMES-DE s
0         9         0         9         2:2       495
10        63        10        61        2:1       661
64        102       62        90        2:1       438
103       129       91        146       1:1       624
130       151       147       152       1:1       153
152       164       153       163       1:1       356
165       181       164       180       1:1       368
182       232       181       227       1:2       606
233       247       228       265       1:1       904
...       ...       ...       ...       ...       ...
```

Figure 8: Output from the most basic use of the aligner

To check whether the aligner worked correctly, you can view this file interactively using the `cwb-align-show` program. The command to run this program is:

```
$ cwb-align-show holmes.align
```

(you can use the `-w` option for a wider display, if your terminal window is big enough).

Press `Return` to display the next alignment pair, `h` for other key commands, and `q` to exit the viewer.

If your parallel corpus is large, it may be advisable to compress the `.align` file by specifying a filename with extension `.gz`, `.bz2`, or `.xz`. All CWB alignment tools handle such compressed files transparently.

9.3 Advanced use of the aligner

It is possible to get improved results from `cwb-align` by making use of different parts of the original data, or by tweaking the configuration of the weight it gives to different kinds of comparison.

One tweak we can make is to the p-attribute used by the aligner to measure similarity. The following command, for instance, will used the `lemma` attribute as the "text" of the corpora when comparing their content:

```
$ cwb-align -P lemma -o holmes.align HOLMES-EN HOLMES-DE s
```

The different p-attributes need to have the same name in both source and target corpora for this to work.

There are various reasons why you might use an attribute other than the default `word` for the lexical comparisons. You might choose to use the `lemma` attribute, for instance, if the two languages are closely related but differ in their inflections (in which case the lemmata would be overall more similar to one another, and thus easier to align, than the actual word-tokens). Alternatively, you might choose to align using the `lemma` attribute if you had a bilingual lexicon available which contained lemmata. `cwb-align` is able use such a lexicon if it is available: words which are identified in the lexicon as equivalent will then count as "similar" for alignment purposes even if they are formally nothing alike.

The format of a lexicon file is shown in figure 9. The aligner can be instructed to use it as follows:

```
$ cwb-align -P lemma -o holmes.align HOLMES-EN HOLMES-DE s -W:50:lex.txt
```

```
be sein
sit sitzen
stand stehen
```

Figure 9: A very short English-German bilingual lexicon file, *lex.txt*

The `-W` option is an *aligner configuration flag*, so it goes after the names of the corpora and the grid attribute - in contrast to the *general options* already discussed, which precede the names of the corpora.

When the `-W` flag is used, you must specify two things: first the *weight* to be given to words that match when aligning sentences, and then the name of the file containing the pairs of equivalents. The weight given in the example above, 50, is equal to the default weight given to an occurence of the exact same word in both languages. This number is one of the parameters that you can change to try to improve the alignment output; see also below. The second thing that must be specified is, of course, the name of the lexicon file.

There are many other parameters that can be tweaked and it may be worth experimenting to see what gives you the best results. We won't cover the details here. All are described in full in the `cwb-align` manual file (accessed by the command `man cwb-align` on Unix, provided as a separate file on Windows).

One thing worth noting, however, is that it is possible to use *pre-alignment*. "Pre-alignment" means that some correspondances are known in advance. In a novel, for instance, there may be chapter boundaries which match across translations, and we can say for certain that a sentence in chapter 1 in language A will not be aligned with a sentence in any other chapter than chapter 1 in language B. This makes the aligner's task easier.

If the indexed corpora contain such pre-alignment information encoded as an s-attribute, then the aligner can be instructed to use it.

In the `HOLMES` corpora there exist paragraphs (s-attribute `p`). Let us assume that these paragraphs are pre-aligned: we know that a given paragraph in `HOLMES-EN` matches one and only one paragraph in `HOLMES-DE`, and that these links are known; it is only the alignment of sentences within each paragraph pair that needs to be found out.

In this case we can add either the `-S` or `-V` option to `cwb-align`.

If we specify paragraph pre-alignment with `-S`, then the aligner assumes that the source and target corpora have the same number of paragraphs, and that the first paragraph in the source (`HOLMES-EN`) corresponds to the first paragraph in the target (`HOLMES-DE`), the second to the second, and so on. This would be done as follows:

```
$ cwb-align -S p -o holmes.align HOLMES-EN HOLMES-DE s
```

Alternatively we can use `-V`. In this case, paragraphs will not be matched up by order - rather, they are matched up by the value of the s-attribute. Since the *Holmes* corpora have *num* as an annotation on `<p>`, there is an s-attribute `p_num` which has values and can be used in this way. This is done as follows:

```
$ cwb-align -V p_num -o holmes.align HOLMES-EN HOLMES-DE s
```

In this case, the order of the paragraphs does not matter: the aligner will always try to match sentences in paragraph 3 in one corpus to sentences in paragraph 3 in the other corpus.

Using pre-alignment improves the output, because fewer possibilities have to be checked for the alignment of each sentence.

9.4 Encoding the aligner's output

An alignment attribute is added to an existing CWB corpus, which must be the *source* corpus of the alignment (not the target). There are two steps in this process.

The first step is to declare the new alignment attribute in the source corpus's registry file.

So, find the `holmes-en` file in the registry directory, and edit it to add the following line:

```
ALIGNED holmes-de
```

(note the use of the lowercase spelling of the attribute name!)

This declares an a-attribute linking this corpus to the `HOLMES-DE` corpus. An a-attribute has the same name as the target corpus.

If you've got the CWB/Perl tools installed, you can use the `cwb-regedit` to make this change, rather than manually editing the registry. The command would in this case be as follows:

```
$ cwb-regedit HOLMES-EN :add :a holmes-de
```

Once the registry file has been updated, the second and final step is to encode the alignment attribute:

```
$ cwb-align-encode -D holmes.align
```

(this command runs very fast and prints no output if everything has gone OK).

There is only one argument to `cwb-align-encode`: the name of the text file containing the alignment data. It is not necessary to name either of the corpora, because the `holmes.align` file contains both names.

It is, however, always necessary to state where you want the encoded files to be placed. The recommended way to do this is the method shown above: with the `-D` option. This puts the a-attribute's data files in the same directory used for the corpus's other attributes (as specified in the registry file).

Alternatively, you can specify a different location with the `-d` option.

Once encoding is complete, it's safe to delete the `holmes.align` file.

This procedure only creates an a-attribute in `HOLMES-EN`, linking it to `HOLMES-DE`. If you *also* want an a-attribute in `HOLMES-DE` linking it to `HOLMES-DE`, you must repeat the procedure with the source and target corpora switched.

You can either re-run the aligner, or re-use the same `holmes.align` file in "reverse mode". `cwb-align-encode`'s reverse mode switches the source and target corpora from what is specified in the `.align` file. This only works, however, provided that there are no crossing beads in your alignment.

That is, first run

```
$ cwb-regedit HOLMES-DE :add :a holmes-en
```

and then add the a-attribute data to `HOLMES-DE`. Assuming that we are re-using the same `holmes.align` file for this step, as explained above, we need the `-R` option to engage reverse mode:

```
$ cwb-align-encode -D -R holmes.align
```


9.5 Importing a pre-existing alignment

What if your corpora have already been aligned, either manually or using a better aligner than `cwb-align`? In this case, you can create an a-attribute by *importing* such existing alignment information with `cwb-align-import`.

`cwb-align-import` is *not* part of the main CWB core, but is instead one of the CWB/Perl tools.

The procedure to **import an alignment from existing information** is as follows.

First, you must encode your information into an **alignment beads file**. An *alignment bead* is one single point of alignment between the source and target corpora. An *alignment beads file* is a file defining a series of beads, plus some header information.

The header line of a beadfile has four items, separated by tabs:

- The CQP ID of the source corpus
- The CQP ID of the target corpus
- The ID of s-attribute encoding the regions to be aligned (i.e. the "grid", as explained above)
- A "key pattern".

After that, every line contains a single alignment bead. This consists of one or more source corpus IDs, then a tab, then one or more target corpus IDs. The IDs must follow the key pattern. Let's consider each of these elements in further detail.

The *key pattern* specifies how ID codes in the beadfile relate to ID codes for regions in your indexed corpora. The ID codes in the beadfile must be unique across each of the corpora.

The most basic case is when we can directly use an indexed s-attribute that has annotation values. For instance, let's assume that the grid attribute is `s`, and that in both corpora there is a subsidiary s-attribute called `s_id` which contains codes that uniquely identify the sentences: `s_s01`,`s_s02`, `s_s03` etc. in the source corpus, and `t_s01`,`t_s02`, `t_s03` etc. in the target corpus. In that case, we can simply use `s_id` on its own as the key pattern. That causes the IDs on the other lines of the beadfile to be matched against the contents of `s_id`. To accomplish this, we specify the key pattern simply as "id" within curly brackets.

Lines after the header must each contain a single *alignment bead*. A bead consists of two columns, separated by a tab. Each column contains one or more space-separated IDs (in our example, from the `s_id` attribute) associated with regions which are to be treated as aligned to one another. The IDs in the first column relate to the source corpus, and the IDs in the second column relate to the target corpus.

The overall beadfile for our hypothetical would then look something like this:

```
CORPUS-SL    NAME-TL    s    {id}
s_s01 s_s02    t_s01
s_s03    t_s02
s_s04    t_s03 t_s04
(...)
```

This specifies that:

- the sentences with ID codes `s_s01` and `s_s02` are equivalent to sentence `t_s01`

- the sentence with ID code `s_s03` is equivalent to sentence `t_s02`
- the sentence with ID code `s_s04` is equivalent to sentences `t_s03` and `t_s04`
- ... and so on.

Such a beadfile can then be imported, creating the a-attribute, using the following command:

```
$ cwb-align-import -p beadfile.txt
```

The `-p` option, short for *prune*, should normally be used. It makes `cwb-align-import` ignore any beads with one or more IDs that don't actually occur in the corpus. Without this option, `cwb-align-import` will abort with an error message if it encounters any bad IDs.

It is not necessary to specify in the `cwb-align-import` command which corpora are being aligned, because that information is on the header line of the beadfile. However, both corpora do need to exist in the active corpus registry (however that is specified).

In our bilingual *Holmes* corpus, things are a little more complicated. The sentences do have ID codes, so there *is* an s-attribute called `s_id`. However, the values of this attribute are not unique. Each paragraph re-uses the same ID codes, starting with `a`,then `b`, then `c`, and so on. So just "id" in the key pattern will not work.

We can *make* the codes unique by combining s-attributes together. Each paragraph in *Holmes* is numbered, in the s-attribute called `p_num`. Therefore, if we combine together the paragraph number and the sentence ID, we will have unique identifiers.

We do this by specifying both attributes in the key pattern, each within braces. We must specify their names in full, as `p_num` and `s_id`, unlike the simple case (where `id` is automatically expanded to `s_id` on the basis of the grid attribute). A key pattern can also contain constant elements around the s-attributes enclosed in curly braces, to allow the IDs to be friendlier in appearance.

An actual beadfile for the *Holmes* corpora is provided as part of the data package. Its filename is `holmes_en_de_align.txt`, and its full format is shown in figure 10). The key pattern in this beadfile consists of a constant `s` followed by the two s-attributes, `p_num` and `s_id`. This means that the leading "s" in each key in the rest of the file will be ignored, and the remainder looked up against `p_num` and `s_id`. The key `s1a` therefore matches paragraph 1, sentence a.

(For further examples of complex key patterns, see **man cwb-align-import**.)

The *Holmes* beadfile includes beads which express empty alignments, that is, correspondances between a region in one corpus and *nothing* in the other corpus. Region `s7a` in the English corpus corresponds to nothing in the German corpus; likewise region `s9b` in the German corpus corresponds to nothing in the English corpus. These kinds of alignments aren't possible in a CWB a-attribute, and by default will cause a fatal error. The flag `-e` ("empty") tells `cwb-align-import` to ignore these lines instead. So we must use `-e` when importing `holmes_en_de_align.txt`. Fortunately, `-e` is automatically activated by the `-p` flag.

The overall command is thus:

```
$ cwb-align-import -p holmes_en_de_align.txt
```

Another flag that is often useful is `-i`. This *inverts* the source corpus and target corpus from what is declared in the beadfile. It means that you can create two a-attributes, one going each way, from the same beadfile. So, since the command above creates an a-attribute in `HOLMES-EN`, pointing at `HOLMES-DE`, for an a-attribute in `HOLMES-DE`, pointing at `HOLMES-EN`, you can use the following command:

```
$ cwb-align-import -i -p holmes_en_de_align.txt
```

```
HOLMES-EN    HOLMES-DE   s    s{p_num}{s_id}
s1a          s1a
s2a          s2a
s3a s3b      s3a
s3c          s3b
s3d s3e s3f  s3c s3d
s4a          s4a
s5a          s5a
s5b          s5b s5c
s6a          s6a
s6b          s6b
s6c          s6c
s6d          s6d
s6e s6f      s6e
s6g          s6f
s7a
s8a          s8a
s9a s9b      s9a
             s9b
s9c          s9c
...          ...
```

Figure 10: Beadfile `holmes_en_de_align.txt` for the *Holmes* corpus pair

9.6 Importing alignment data from TMX format

Finally, an alignment data import tool recently added to CWB is `cwb-align-tmx2beads`. This is part of the CWB/Perl package. Its purpose is to generate beadfiles from corpus texts in the TMX format. TMX is a fairly common XML-based system for storing alignment data that is very different to the CWB representation. `cwb-align-tmx2beads` is not fully documented here because it remains experimental (as of version 3.5.0), but an explanation of its operation is contained within the POD documentation sections of the Perl script itself.

The beadfile generated by `cwb-align-tmx2beads` can be used with `cwb-align-import` as described in the previous section.

A Appendix: Registry file format

The following is a sample registry file created by **cwb-encode**. The **cwb-regedit** also creates registry files in this format.

```
##
## registry entry for corpus BNCSAMPLER
##

# long descriptive name for the corpus
NAME ""
# corpus ID (must be lowercase in registry!)
ID   bncsampler
# path to binary data files
```

```
HOME /home/Corpora/data/bncsampler
# optional info file (displayed by "info;" command in CQP)
INFO /home/Corpora//bncsampler/.info

# corpus properties provide additional information about the corpus:
##:: charset  = "utf8" # change if your corpus uses different charset
##:: language = "??"     # insert ISO code for language (de, en, fr, ...)


##
## p-attributes (token annotations)
##

ATTRIBUTE word
ATTRIBUTE pos
ATTRIBUTE hw
ATTRIBUTE semtag
ATTRIBUTE class
ATTRIBUTE lemma


##
## s-attributes (structural markup)
##

# <text id=".."> ... </text>
# (no recursive embedding allowed)
STRUCTURE text
STRUCTURE text_id              # [annotations]

# <s> ... </s>
STRUCTURE s


# Yours sincerely, the Encode tool.
```

CWB traditionally had a more flexible registry file format (which is still accepted for backward compatibility), which could contain a variety of other declarations. The standard format for new corpora, however, is as given above; we recommend that you stick to this format, since it is in fact enforced by the CWB/Perl scripts.

Finally, it is worth noting that directory and file paths in `HOME` and `INFO` entries have to be *double*-quoted if they contain blanks or other non-standard characters (ASCII letters, digits, `-`, `_`, `/` and `.` are ok, as long as the path does not begin with `.`). In a double-quoted path, `"` must be escaped as `\"` and the backslash `\` as `\\`. If you use `cwb-encode` and `cwb-regedit`, they should always create valid entries, with quotes added when necessary.

B Appendix: Limits

This section documents some technical limits of CWB. Most of these are due to the design of the index file format (notably the pervasive use of signed 32-bit integer values), but some additional restrictions

are imposed by implementation decisions.

- The maximum corpus size is 2,147,483,647 tokens (the largest value that can be stored as a signed 32-bit integer). In the CWB source code, this is represented by the macro `CL_MAX_CORPUS_SIZE`.
- The maximum size of a p-attribute lexicon is 2,147,483,647 bytes (or 2 GiB), due to the use of signed 32-bit integer offsets into the `.lexicon` file.[19]
- The same 2 GiB limit applies to the "lexicon" storing all distinct annotation strings of an s-attribute with values (the `.avs` file).
- The maximum length of an annotation string is 4096 bytes, including the NUL terminator. In UTF-8 encoding, a single character may occupy two or more bytes, so the maximum *character* length of an annotation string may be smaller.

  The length limit is determined by the macro `CL_MAX_LINE_LENGTH` in the CWB source code and can be increased if absolutely necessary. This is strongly discouraged, as it may create index files that are not compatible with standard builds of CWB.
- The maximum length of a filename is determined by the macro `CL_MAX_FILENAME_LENGTH` in the CWB source code. Its default value is currently 1024 bytes.
- The maximum length of an input line in the `.vrt` format is currently 65,536 bytes. It is determined by the macro `MAX_INPUT_LINE_LENGTH` in the **cwb-encode** source code.

C Appendix: Magic compression and decompression

The use of automagic[20] decompression of input files or compression of output files has been mentioned throughout. This appendix brings together this information in one place.

This behaviour applies to CQP as well as the various CWB utilities discussed in this manual.

- When an input file is specified, CWB reads it differently depending on its file extension.
- If a filename ends in `.gz`, it is assumed to be compressed in the **gzip** format.
- If a filename ends in `.bz2`, it is assumed to be compressed in the **bzip2** format.
- If a filename ends in `.xz`, it is assumed to be compressed in the **xz** format.

Single-file archives only are supported. That is, a file *somefile.vrt* can be read by CWB if it is compressed to *somefile.vrt.gz*, but *not* if it is placed in the compressed archive *somefiles.tar.gz*.

Automagic decompression require the appropriate program to be installed and findable by CWB, that is, they must be in one of the directories named in the PATH environment variable. However, if for whatever reason your binaries for `gzip/bzip2/xz` are not in a standard location, and you either can't or don't want to modify your PATH variable, another solution is possible:

- Set the environment variable `CWB_COMPRESSOR_PATH` to the folder that contains the programs in question.
- Only one folder may be specified, so all three need to be in the same place.

[19] Lexicon size refers to the sum of the byte lengths of all annotation strings in the lexicon, including NUL terminators.
[20] Automatic, as if by magic.

- If one of them isn't, you can add a (hard or symbolic) link for the missing program from the folder named in `CWB_COMPRESSOR_PATH` to the program in its actual location.
- Make the `CWB_COMPRESSOR_PATH` environment variable available to your programs. Depending on your shell, there will be different ways to do this; one common one is to set the environment variable prior to the invocation of the program:

  ```
  $ CWB_COMPRESSOR_PATH=/path/to/zipping/programs/ cwb-encode [...]
  ```

  but others exist.

This will make automagic decompression work as expected.

While `gzip`, `bzip2`, and `xz` are standard or widely-available utilities on most Unix-like operating systems, they are not easily available everywhere, and especially on Windows may be very hard to install. In this case, an alternative is to use **7-zip**.

7-zip is a free/open-source tool which can handle all three of the supported formats. It is easily installed on Windows (from https://www.7-zip.org/), and a package with ports of the non-GUI 7-zip executables is available for Unix-like systems as well: **p7zip**, installable via package managers or from http://p7zip.sourceforge.net/).

If you wish to use **7-zip** you must, again, make sure that the directory containing its executable (`7z`, or `7z.exe` on Windows) is on your PATH, or else use `CWB_COMPRESSOR_PATH` to specify its location.

You must then set the environment variable `CWB_USE_7Z` (normally to 1) to signal to CWB that it should use `7z` rather than the other programs. Your command might then be:

```
$ CWB_USE_7Z=1 CWB_COMPRESSOR_PATH=/path/to/7zip/programs/ cwb-encode [...]
```

The latest developments to automagic compression:

- `CWB_COMPRESSOR_PATH` is available in CWB v3.4.37+
- `CWB_USE_7Z` is available in CWB v3.4.35+

