Corpus Encoding Tutorial: First Steps [Draft]

Stefan Evert

30 Jun 2002

The CWB **input format** is one-word-per-line (more precisely, one token per line), with annotations given as additional `TAB`-separated columns. XML **tags** must appear on separate lines.

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

Figure 1: file *example.vrt*

- create separate **data directory** for binary corpus data
- **encode**, i.e. convert to CWB binary format with

```
cwb-encode -d /path/to/data -f example.vrt -R /path/to/registry/example
           -P pos -P lemma -S s
```

The first column is automatically encoded as the default **positional attribute** (**p-attribute**) `word`. `-P` flags are used to declare additional p-attributes. `-S` flags declare **structural attributes** (**s-attributes**), which encode *non-recursive* XML tags and whose names must correspond to the XML element names. `-R` automatically creates a **registry file**, whose filename *must* be in *lowercase*. The CWB name of the corpus is identical to the name of the registry file, but is written in *uppercase* (here it will be `EXAMPLE`).

Input files with the extension `.gz` are assumed to be in gzip format and are automatically uncompressed. Multiple input files can be specified by using the `-f` switch, and will be read in the order in which they appear on the command line. Note that shell wildcards (e.g. `-f *.txt`) won't work. Switches and options *must precede* the flags used to declare attributes in the command line.

- create **lexicon** and **index** for p-attributes

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

The `-V` switch enables an additional validation pass when the index has been created. It should be omitted when encoding very large corpora ($\approx$ 100M tokens). 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 128 MB of RAM, `-M 64` is a safe choice.

- always erase all files in data directory before re-encoding corpus!
- get some information about the corpus (add `-s` option for details)

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

Text will often be available in **XML format**. CWB v3.0 offers improved XML support. Useful flags for encode are `-x` for XML compatibility mode (recognises default entities and comments), `-s` to skip empty lines in the input, and `-B` to strip whitespace from tokens. Typical XML input might look like this:

```
<!-- 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 2: file *vss.vrt*

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 XML parsing. The attributes of XML tags such as

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

can be stored as a plain text string by using `-V` instead of `-S`, but are not easily accessible from CQP. It is more desirable to declare XML attributes explicitly and split them into multiple s-attributes. Note that the flags `-xsB` should (almost) always be used and will automatically ignore the XML comment line.

```
cwb-encode -d /path/to/data -f vss.vrt -R /path/to/registry/vss
           -xsB -P pos -P lemma
           -S s:0 -S p:0 -S story:0+num+title
```

This will create a registry file for the corpus `VSS`, including the s-attributes `s`, `p`, `story`, `story_num`, and `story_title`. Don't forget to build indices for the p-attributes as above:

```
cwb-makeall -V VSS
```

If registry files are not written to the **default registry** directory `/corpora/c1/registry`, all CWB tools accept the `-r` flag to specify a different registry directory, e.g.

```
cwb-makeall -r /path/to/registry -V VSS
```

Data compression for p-attributes is accomplished with two separate tools: `cwb-huffcode` for the token stream data, and `cwb-compress-rdx` for the index. Use the `-P` flag to specify a single p-attribute, or compress all p-attributes with `-A`.

```
cwb-huffcode -A VSS
cwb-compress-rdx -A VSS
```

When compression was successful, the tools will list the data files which are now redundant and can be deleted (namely, *attrib*`.corpus` after running `cwb-huffcode`, and *attrib*`.corpus.rev` and *attrib*`.corpus.rdx` after running `cwb-compress-rdx`).

Running `cwb-makeall` now will show that the p-attributes are already compressed. Note that by default, the compressed data files are validated, so it is safe to remove the redundant files. Validation can be turned off with the `-T` option, but is less performance-critical than with `cwb-makeall`.

In order to **add p-attributes** after encoding, create input data in the standard one-word-per-line format, containing the new attributes only. Here is an example with WordNet synonyms encoded as **feature sets**.

```
|
|be|be identical to|characterize|constitute|...|
|
|elephant|
|
```

Figure 3: file *syns.vrt*

Encode as usual, but **suppress** the default `word` attribute with `-p -`. It is highly recommended to check first that the number of tokens in the new file (`wc -l syns.vrt`) is identical to the corpus size (as reported by `cwb-lexdecode -S VSS`).

```
cwb-encode -d /path/to/data -f syns.vrt -p - -P syn
```

The registry file must be edited manually, adding the line

```
ATTRIBUTE syn
```

Don't forget to create a lexicon and index for the new attribute

```
cwb-makeall -V VSS
```

and compress the p-attribute if this is desired. Before re-encoding the `syn` attribute, the corresponding data files (matching the shell pattern `syn.*`) **must** be deleted!

In order to **add s-attributes** with computed start and end points after encoding, use the `cwb-s-encode` tool. The start and end positions of existing s-attributes can be obtained with `cwb-s-decode`. The following example shows how sentence length annotations can be added to the `VSS` corpus. The existing `s` attribute is decoded into a temporary file, `gawk` 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
gawk 'BEGIN { FS=OFS="\t" }  { print $1, $2, $2-$1+1 }' s.list > s_len.list
cwb-s-encode -d /path/to/data -f s_len.list -V s_len
```

Note that it is currently *not necessary* to run `cwb-makeall` after adding an s-attribute to an existing corpus. However, the new attribute must be declared in the registry file by manually adding the line

```
STRUCTURE s_len
```

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

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

We then run the chunk parser on the temporary file, which adds its <np> and <pp> tags to the token stream, creating the file shown below.

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

Figure 4: file *chunks.vrt*

It is important that the token stream is left intact when adding the XML annotation. 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 is identical to the corpus size.

```
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, the non-XML lines in the input file will simply be ignored). The flag `-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).

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

`cwb-encode` will issue warnings about nested regions being dropped. As can be seen from Figure 4, <np> (as well as <pp>) regions may be embedded recursively. We can now change the `:0` modifier to `:2`, allowing up to two levels of embedding (for each element type, i.e. <np>s embedded in larger <np>s etc.). In general, `:`n allows up to n levels of embedding. Embedded regions will automatically be renamed to `np1`, `np2`, `pp1`, and `pp2`, respectively.

```
cwb-encode -d /path/to/data -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`. Again, the corresponding `STRUCTURE` lines in the registry file have to be added manually, but it is not necessary to run `cwb-makeall`.

The `cwb-lexdecode` tool gives **access** to the **lexicon** of positional attributes, listing word forms / annotation strings 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.

```
cwb-lexdecode -f0 -P pos -f tags.txt VSS
```

With the `-p` option, tokens / annotations matching a regular expression can be extracted. Case- and diacritics-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
```

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 in the corpus).

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

This output format can reliably be re-encoded when the `-xsB` options are used. 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.

The `cwb-scan-corpus` command extracts **combinatorial information** from 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 isn't 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).

```
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` (such as the file *language-model.gz* in the example below), the output file is automatically gzipped.

```
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. Note that it may be necessary to enclose more complex keys (containing shell metacharacters) in single quotes.

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

The final example extracts pairs of adjacent adjectives and nouns from the `VSS` corpus, e.g. as candidate data for Adj+N collocations. Constraint keys are used to identify adjectives and nouns, and only nouns starting with a vowel are accepted. Note the `c` and `d` modifiers (case- and diacritics-insensitive matching) on this regular expression.

```
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;
```