A guideline for converting, tokenizing and parsing text corpora

Preparing a corpus in order to import it to Korp

Conversion to HRT

The format of the original data can differ between corpora. It can be for example plain text, PDF, XML, RTF. The aim is to convert this data to **HRT**, a simple form of XML, the pre-format of VRT (VeRticalized Text). The data must be **UTF-8** encoded Unicode. The data consists of <text> elements with all needed structural attributes (see https://www.kielipankki.fi/development/korp/corpus-input-format/). The text elements may contain child elements <paragraph> and those can be split to <sentences>.

More information about the conversion process can be found here: https://www.kielipankki.fi/development/korp/corpus-import/#Converting_a_corpus_and_importing_it_to_Korp

In case the original data does not have paragraph and/or sentence elements, these will be added in the tokenizing process.

In case the original data has other or additional structures (e.g. tables, linegroups), they should preferably be preserved in the VRT format (a guideline on how to preserve the original structure with a standardized set of element names is planned to be created).

Note that all tokens should be within sentence structures; otherwise Korp will not show them.

Note also, that all element tags have to be on their own line, and they have to be at the beginning of the line (be aware of empty spaces here).

Example of **HRT** format:

```
<text author="Agricola, Mikael" contributor="" title="Se meiden Herran Jesusen Christusen
pina, ylesnousemus ia taiuaisen astumus, niste neliest euangelisterist coghottu." year="1549"
lang="fin" datefrom="15490101" dateto="15491231" timefrom="000000" timeto="235959"
natlibfi="Klassikkokirjasto, Kansalliskirjasto." rights="" digitized="2017-04-20"
filename="KlK_with_timestamps.xml" book_number="975">
<paragraph>
<sentence>
(…)
</sentence>
</paragraph>
(etc.)
</text>
```

Tokenizing: Tools and process

The scripts for tokenizing were created by Jussi Piitulainen and can be found in GitHub: https://github.com/CSCfi/Kielipankki-konversio/vrt-tools

If you have your own copy of ‘Kielipankki-konversio’ on Taito already, you can just update it with the help of the command ‘git pull’. Then you can create a symbolic link to the folder ‘vrt-tools’ from the respective corpora folder in your working directory, e.g.:

```
ln -s /homeappl/home/username/Kielipankki-konversio/vrt-tools/ bin
```

Now you have the tools available in your corpus folder in a folder named ‘bin’. Of course, this is only one suggestion on how to use the tokenizing scripts.

The scripts for tokenizing are

1. **vrt-dehype**: This script attempts to remove line-breaking hyphens (and page numbers) in plaintext within VRT markup (only needed when such hyphens are found in the data!)
2. **vrt-paragraize**: This script inserts paragraph tags into plaintext within VRT markup using lines of all whitespace as boundary indicators within blocks of plaintext (not needed if the data already has paragraphs!)
3. **hrt-tokenize-udpipe**: This script tokenizes plaintext within VRT markup

In case all three scripts are needed, they should be run one after the other, in the given order. They could also be combined in a shell script.

Validation of the result VRT

The result of the tokenizing process should then be validated.
The script for this is ‘**vrt-validate**’. The command is:

```
bin/vrt-validate filename.vrt
```

In case the validator gives out messages, then they should be investigated further with ‘**linedump**’:

```
bin/linedump -k 2751241 --repr filename.vrt
```
(where the number is the number of the line in question)

Checking sentence lengths

The tool for **checking the sentence length** is ‘**vrt-report-element-size**’. Checking the number of tokens within a sentence is the default here. It is used like this:

```
bin/vrt-report-element-size --filename --summary=V5 tokenized/filename.vrt
```

The options –filename and --summary=H5 (or V5 or H11 or H13 for more quantiles and stuff) can be used to get a summary report, where the most interesting number for the present purpose is the maximum length of sentences (and for other purposes maybe median and the low and high quartiles).
There is no definite limit in the number of tokens per sentence, but if the number exceeds 100 significantly, it is worth to have a closer look.

There is a tool to split extra-long sentences: ask Jussi (It is within the TDPipe!)

Fixing sentence breaks

The data should be checked regarding **sentence breaks**. Possible problems are sentence breaks after abbreviations (e.g. “jne.”) or before words starting with a capital letter, like names.

If needed, fix the sentence breaks with the script **vrt-fix-sentbreaks:**

```
bin/vrt-fix-sentbreaks --out folder/filename_sentfix.vrt folder/filename.vrt
```

Parsing with TDPipe

It is recommended to create a copy of the script TDPipe (available in folder ‘bin’) to the corpus folder, one level higher than the data folders, and make it executable:

```
cp bin/TDPipe .

chmod +x TDPipe
```

1. **Insert names (naming or re-naming fields)**

   When the tokenizer ‘hrt-tokenize-udpipe’ was used to tokenize the data, the data already has field names. This is visible from the first line of the VRT data, e.g.:

   ```
   <!-- Positional attributes: id word spaces -->
   ```

   In this case no names need to be inserted, but the field ‘id’ should be renamed to e.g. ‘tokid’, because the parser will produce another field with the name ‘id’. The following command will fix this:

   ```
   bin/vrt-rename -i -m id=tokid path/file
   (-i means overwrite, -m means match)
   ```

2. **Packing**
   The data will be packed in files of a size which is suitable for the parser. In the areas where the data is splitted, markers will be added, so that the data can be put back together later at the correct cuts.

   ```
   bin/game --log logPack bin/vrt-pack -o data opensub-fi-2017-valid/
   ```

   The parser expects to find the input data in a folder named 'data', so this is created here for the packed data!

   NOTE: It does not work when there is a folder ‘data’ already!

   Check the log files: err should be size '0'

   In file *.out STATUS should be ‘0’.

3. **TDPipe**
   This tool runs the scripts for tagging and parsing.

```
bin/gamarr --job parse001 --log logAllParse --GiB=16 ./TDPipe // data/a001/*.vrf
```

   Give a job name (default job name is 'game'), give a name for log folder, maybe ask for more disk space (default is 8 GiB).
   The maximum is 1000 files per array, so this parsing command might have to be run separately for each folder within folder 'data'
   (if the files altogether do not exceed the amount of 1000, the command can be run on all subfolders at once: // data/*/*.vrf)

   NOTE: In log the err files are not supposed to be empty here!

   Possibilities to check the queue:

```
squeue -l -u username | grep PEN

squeue -l -u username | grep -Eo 'RUN|PEN' | sort | uniq -c
```

   (-E means I can use the pipe for 'or', and -o means that it shows only the lines with RUN or PEN)

```
squeue -l -u username | tail
```

4. **Unpacking**
   After parsing the data can be put back together at the correct cuts, which were marked during the packing process.

```
bin/game --log logPack --job unpack bin/vrt-unpack -o folder_out --vrf=vrf.Dparse data/
```

   Log can go to the same log folder than pack, the job name is 'unpack' here. The output directory has to be a new folder (here: folder_out).

   The input files names have the extension 'vrf.Dparse'. Folder 'data': all subfolders will be worked on.

5. **Cleaning up**
   In this step all columns with dots in their names will be removed.

```
bin/gamarr --log logCleanup bin/vrt-drop -i --dots // opensub_out/*
```

   Run as an array. Create a new log folder. Array always needs '//' and then all files in the given folder.

6. **Renaming and final ordering of pos attributes after parsing**
   Finally, the positional attributes have to be renamed and re-ordered, so that Korp can handle them. The commands are **vrt-rename** and **vrt-keep**. Of course they can also be run separately.

```
bin/vrt-rename --map tokid=initid --map feat=msd --map id=ref --map head=dephead --map
rel=deprel folder/filename_cleaned.vrt | bin/vrt-keep --names
"word,lemma,pos,msd,ref,dephead,deprel,spaces,initid" > folder_final/filename_final.vrt
```

If you have several files for one corpus, create a zip file containing the final files.

NOTE: It is recommended to upload the successfully parsed data (zip file) to the respective corpus folder in IDA, especially if it is not progressed further right away (e.g. OpenSubtitles).

Create a Korp package

Encode VRT data into the CWB database format and create a Korp package for the corpus, which can be done with a single command: **korp-make**.

The command should be run from your HOME directory in the respective corpus folder. In this corpus folder you need to have a folder ‘scripts’ containing your conversion scripts and a readme about the conversion process. Furthermore, in the corpus folder you need to have a readme about the corpus (get the info from METASHARE) and a file ‘corpus.conf’.

The basic content of the **corpus.conf** is just the path to the scripts' folder:

```
script-dir = $HOME/ClassicsLibrary/scripts/
```

For licensed data it could be:

```
script-dir = $HOME/Aikakauslehti/scripts/
licence-type = ACA
lbr-id = 201407301
```

(the libr-id is typically the URN of the top META-SHARE page for the corpus)

And you should add a line for the positional attributes:

```
input-attrs = lemma pos msd ref dephead deprel spaces initid
```

The command for **korp-make**:

```
/proj/clarin/korp/scripts/korp-make --config-file=corpus.conf --readme-
file=readme_about_corpus.txt korp-id /wrk/username/corpus/corpus-parsed.zip
```

If the script has to be re-run on the same corpus, add option --force (this overwrites all data).

Add corpus configuration to the Korp frontend

This consists of making changes to the Korp configuration and translation files *on your own branch* of the Korp frontend repository and committing the changes.

Instructions will be added here later.

Install the corpus package and configuration

At this stage, the corpus configuration should be installed on a separate test instance of the Korp frontend. Jyrki takes care of this! Just let him know, when you pushed your changes in GitHub.

Test the corpus in Korp

Check that the corpus shows up and works as expected in the Korp test instance. The corpus should be tested by at least one other person of our group.

We would need a detailed instruction on what has to be tested.

Inform others of the corpus and request feedback

After the test version works as expected, you should inform at least *fin-clarin (at) helsinki.fi* and the original corpus owner or compiler if applicable. If you get feedback, you may need to redo some of the previous steps.

Install the corpus configuration to the production Korp
Once the corpus works in Korp as desired, it is ready to be installed on the production Korp. Jyrki takes care of that.

Upload the corpus package to the IDA storage service

For a guide on how to proceed with these further steps see also
https://www.kielipankki.fi/development/korp/process-overview/