Translating and aligning using Translog and Yawat

This is an elaborate guide on how to use a pipeline consisting of Translog-II, TPR-DB, YAWAT, and manually editing and Python scripts. The guide is quite long and detailed, so a flow chart is given as a summary on the last page.

1 Setting everything up

1.1 Install Python

You will need to run a Python script from time to time. (You will not have to program yourself.) Therefore, you need to have Python installed on your system. To check whether it is installed, open your command prompt (Windows key + R, type "cmd"). Here, type `python --version`. If Python is installed, you should see something like "Python 3.8.2", where "3.8.2" is the version number.

```
C:\Users\bramv>python --version
Python 3.8.2
```

If you do not see such text, or the version number is lower than 3.6, you have to install (a new version of) Python first. First **make sure to close the command window**, then go to https://www.python.org/downloads/ and download and install Python (3.6 or higher). When asked whether or not to add Python the `PATH` variable, **click yes**.

After the installation of Python, run the above steps again (open a new command window, type `python --version`). Now you should see the version number. Python has been installed successfully!

1.2 Installing Translog

You will translate texts in the Translog environment. This program records the keystroke that you type in its fields (so it will not record what you do outside the program). To download the program, go to https://www.dropbox.com/s/35edkvkvewpufjp/TranslogInstaller-2020-02-20.msi?dl=1 and download the installer file (a MSI file). When installed, double click it to install. **You may get a Windows warning** that the origin of this file cannot be verified (or something along those lines), but you can ignore that and just continue (the program is safe to use) by clicking "More info" and then "Run anyway". After installation, two shortcuts will be placed on your desktop: one *Supervisor* and one *User*. This distinction will be important later on.

1.3 Installing Visual Studio Code

As part of the process, you will need to make some manual changes in generated XML files (we will come back to that later), and for this you need a good text editor. It is very helpful if you have a text editor that allows you to do find-and-replace in only the selected text. I strongly recommend using Visual Studio Code, which you can download from here https://code.visualstudio.com/.

1.4 Recommended directory structure

Because you will be creating many files, it is a good idea to structure your data clearly. In case something goes wrong, you can then always retrace your steps. I recommend the following directory structure, where bullet points indicate the hierarchy:

`C:\Users\you\Desktop\experiment\`: root directory for this experiment

- `scripts\`: the directory where you will download some Python scripts to in the next section
- `translog_projects\`: where you would place all Translog *.project files
- `translog_orig\`: where you would place all the resulting Translog log files (*.xml). This directory will serve as a back-up folder and should not be used
- `translog_lang\`: we will automatically add a special tag to the XML files. As a precaution, we will do that in this separate directory rather than `translog_orig\` so that if something goes wrong, we do not lose our original translation file
- `aligns_orig\`: where you will save the first alignment files (*.atag, *.src, *.tgt), created by Yawat. This directory will serve as a back-up folder and should not be used
- `aligns_manual_seg\`: where you will make manual edits to fix segment alignment and segmentation
- `aligns_manual_tok\`: where you will make manual edits to the tokenisation
- `aligns_final\`: where you will download the final word alignments to
- `tables\`: the TPR-DB can automatically create tables with a lot of useful information. After completing the word alignment, you can download the tables into this directpry

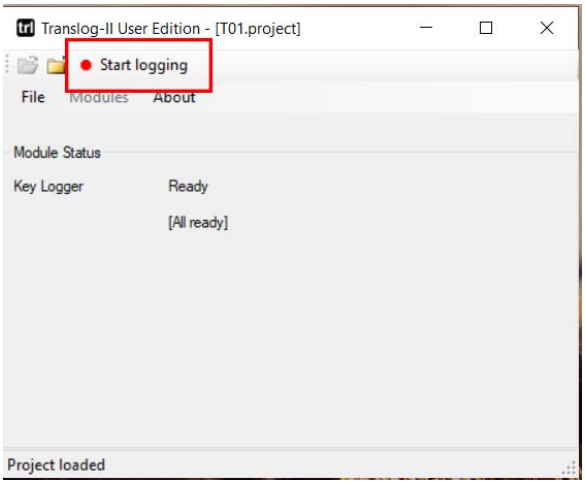
1.5 Download scripts from GitHub

The scripts that you may need to run, are available on GitHub. You can clone this repository: https://github.com/BramVanroy/predict-translate-annotation. **However**, if you have no experience with cloning repositories or using git in general, it might be easier to just download the zip file. You can download it here: https://github.com/BramVanroy/predict-translate-annotation/archive/master.zip and unzip it into `scripts\` but make sure that you do not include the wrapping directory of the zip file. In other words, your `scripts\` directory should **directly** contain the *.py Python files.

2 Translating in Translog

You will start your translation journey with the Translog program, which should be installed by now. You should also have a Translog project file. This file (actually an XML file) is used by Translog to create the correct settings (window size etc.) and also fill in the source text for you. It leaves the translation window open for you to translate into.

Because you will be translating (and not creating new projects), you should find the *Translog-II* ***User*** shortcut on your desktop and launch it. Once the program has launched, you can open a project by clicking *File > Open project*.



As soon as you click the *Start logging* button, the translation process will start. This means that the program will begin monitoring your keystrokes in the program as well as the duration. Therefore, **it is important to start the task immediately after clicking the *Start logging* button and to not go out of the program until the task is finished**. Do not leave the program until the task is finished (so do not go to Facebook or browse the web). This will all influence the results. This also means that you should not leave your computer to go to the bathroom or to grab some coffee. Do those things *before* you start logging or after you have closed the logging process. Due to the nature of the processing of your translation in the TPR-DB, you should **not use single quotes (')** for marking direct speech or emphasis. Instead, use double quotes (“). Single quotes may lead to wrong tokenization. In grammatical markings (e.g. a possessive) you can still use the single quotation mark.

When you are satisfied with your translation, click *Stop logging*. You will be asked whether or not to save the log file. Click *yes* and save the file to the `translog_output\` directory under a name that is clear. Typically, you would use your participant ID + the abbreviation of the task + the project ID. example, if you are **P**articipant **01**, doing a **T**ranslation task, and translating project **04**, then you would save the log file as **P01_T04.xml**. Now copy-and-paste this file into `translog_lang\` so that we can use it in a later step: the log file should be saved in both `translog_orig\` and `translog_lang\`.

For the following steps it is easier if you already have translated multiple texts. These steps work with batches of texts (multiple texts at once) so if you can process many texts at once, you need to run these steps fewer times.

3 Add a language tag automatically

In a following step we will upload our data to the Translation Process Research Database (or TPR-DB). It expects that our XML file contains a special `<Languages/>` tag to indicate which language it contains as well as the type of task that was performed. This tag is not yet present, so we need to add it. To do so, we can use one of the scripts in the `scripts\` directory by using the command line.

First, you need to open the command line interface (or CLI). You can do this by going to the root directory of your experiment in Windows explorer `C:\Users\you\Desktop\experiment\`. Then, hold the Shift key and right click on the `scripts\` directory. This will give you some advanced options. Look for the option to *Open Command Prompt here* or alternatively *Open PowerShell window here*. If that does not work, you can press Windows key + R and type “cmd”, which will open the Command line. To navigate to your experiment’s folder, you can type `cd "<your-directory>"` (`cd`: **c**hange **d**irectory), e.g. `cd "C:\Users\you\Desktop\experiment\scripts\"`.

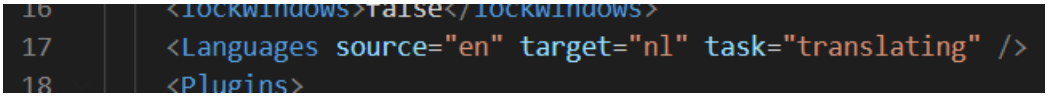
Now you are ready to run a Python script to automatically add the required tags to the XML files. Type the following in the command window:

```
python add_lang_tag.py ..\translog_lang\ --src_lang en --tgt_lang nl --task translating
```

This command will ask python to run the script on the directory `..\translog_lang\`[1], and adding source language "en" (for English), target language "nl" (Dutch), and "translating" as the task, to each file. You can change this to what you need. English, Dutch, and "translating", are the default values so if those are the ones that you need, you can leave them out and instead run:

```
python add_lang_tag.py ..\translog_lang\
```

Open up an XML file in `translog_lang\` and verify that you can indeed find a Languages element.



After this process finishes, we are ready to upload our data to the TPR-DB!

4 Uploading data to TPR-DB

See also the online guide: https://sites.google.com/site/centretranslationinnovation/tpr-db/uploading

The upload tool expects you to upload a ZIP file that contains all the Translog-II XML files according to the naming conventions of PXX_YZZ.xml which we discussed before. All files should be zipped together, so you would upload one ZIP file (e.g. experiment.zip) that contains all *.xml files. Specifically, you first need to zip all the XML files inside the `translog_lang\` directory that contains the modified XML files. Make sure that you ZIP the files. **Do not use other archiving/compression methods such as RAR, 7z, gzip, and so on.**

After zipping your data, go to https://critt.as.kent.edu/cgi-bin/yawat/yawat.cgi and login with your given username and password. If you do not have those, ask your supervisor. After logging in, go to https://critt.as.kent.edu/cgi-bin/yawat/tpd.cgi. Here you can upload new studies. Simply click the *Select File…* button and select the ZIP file that you just created. Because we already added our `<Languages/>` tag automatically, we can leave all fields empty *except* for the "Study name". **Use a short but distinctive name!** In all following steps, when uploading changes that you made to some files, you should upload those files to this same study name. When you upload the Translog file to the TPR-DB, the individual XML files will be converted into the YAWAT format. YAWAT allows us to do word alignment. Behind the scenes, a parser will first chunk our source and target text into sentences and into tokens (segmentation

[1] **The dots are important here!** `..` selects the parent directory of the directory we are currently in, so here the parent of scripts\ is `C:\Users\you\Desktop\experiment\`. Then we find `\translog_lang\` in that directory. This is called a *relative* path because it is relative to our current location. You could also use an absolute path, which would be `C:\Users\you\Desktop\experiment\translog_lang\`. As you can see, that is a lot more to type so we prefer relative paths whenever possible. If there is a space in the directory name that you want to give as an argument to a command line script, make sure to wrap it in quotation marks, e.g. `"C:\Users\you\My Files\experiment\translog_lang"`.

and tokenization). Because this is an automated system, there are bound to be errors in both the created sentences and the created tokens. Luckily, we can fix those manually.

5 Manually correcting data

Before you can start aligning the words, you will need to make sure that the segments correspond one to one. This is not always the case: as a translator, you may sometimes wish to translate one sentence into two or the other way around. On top of that, sentence segmentation itself may not have occurred correctly. Therefore, we must verify that the automatic sentence segmentation has been correctly and correct it where necessary. Finally, the tokenization process may have made some errors that we have to fix manually, too.

Make sure to check correct tokenization (Section Error! Reference source not found.**), correct segmentation, and correct segment alignment before starting word alignment in YAWAT.** If you have to change tokenization or segmentation after alignment, you may need to redo the whole word alignment process again because the aligned token IDs may have changed so the alignment mapping is not correct anymore.

To fix the segment alignments and redo the tokenization, we have to do some manual work. First, we need to download the files that the TPR-DB created. Go back to the TPD (https://critt.as.kent.edu/cgi-bin/yawat/tpd.cgi) and click *Download* Alignments. If the button is colored red and unavailable, click "save Yawat Alignments" first. Download and unzip the ZIP file in `aligns_orig\`. This directory should now contain three files for each XML file that you uploaded:

- *.atag: an alignment file that keeps track which segments are aligned with each other and which files to look into
- *.src: a source XML file that contains a lot of information about the source text, including keylogging info.
- ***.tgt**: a target XML file that contains a lot of information about the target text, including keylogging info. **These are typically the only files that you as an annotator should manually edit.** See Section 5.1 for more details about replacing existing src-files and atag-files.

`aligns_orig\` will serve as a backup directory. **Do not change the files in this directory.** At this point, you can **copy** its contents to `aligns_manual_seg\` where we will manually edit the files.

5.1 (Optional) Replacing src and atag with manually validated files

If your supervisor has provided you with existing src-files and/or atag-files, these should be uploaded as well. These files should contain manually verified source and alignment files. This is useful for two reasons. First of all, it decreases the manual work that each translator has to do. Second, it ensures that the process data of different translations of the same source text are comparable. If the translators of the same source text would all manually correct the tokenization and segmentation of the source text, then those versions might not be identical. Subsequently, the data that is calculated for a token or segment may not correspond directly to the token or segment of the other translator, preventing any generalizations to be made. Therefore, it is a safer bet to have all translators use the same tokenized and segmented version of the source text. Predefined alignment (atag) files may also be provided which at that point would only contain segment alignment, to ensure that the correct number of alignment points is already present.

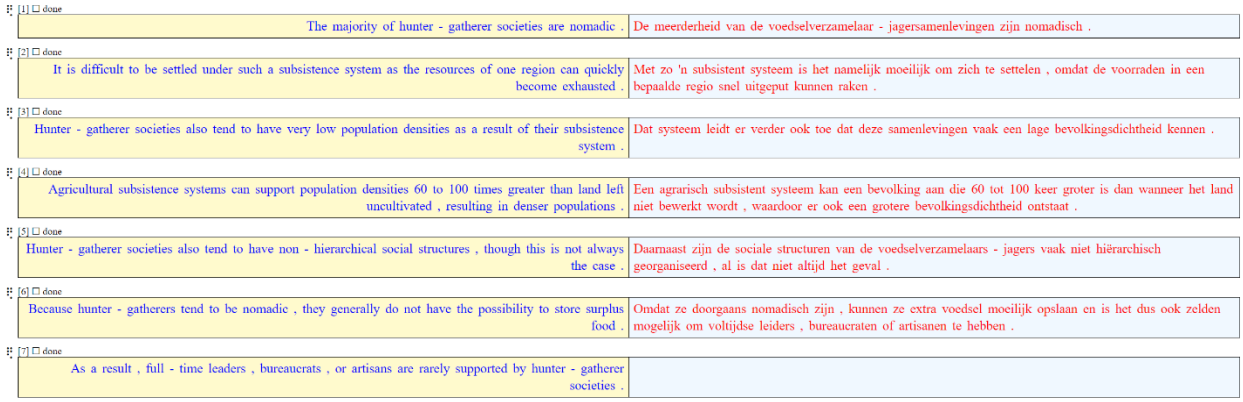
To replace existing src-files and atag-files, just upload the given files to the TPR-DB as you did before as a ZIP file. **After that, download all alignment files again.** (You can overwrite the previous ones in `aligns_manual_seg\`, *assuming that the replaced files are correct*.) This gives you a good base to start from because you still need to verify and possibly modify the target files (*.tgt).

Note that at any point in time, e.g. between fixing the segmentation and tokenization, you can ZIP alignment files and upload them to the TPR-DB. This is useful if you want to see your changes in YAWAT and want to make sure that your progress is headed in the right direction.

5.2 Segmentation and segment alignment

5.2.1 Checking the segment alignment

To have a look at the segment alignment, click on the *Open Yawat* button. This will give you a list with all the files that are part of this study. You can then click on a specific translation, e.g. *P01_T02*, which opens up the Yawat interface to word align your text. However, as said before, we must first verify for all files whether the segment alignment is correct. On the left side you see the original source text, and on the right side the translation. In the example below (albeit illegibly small), you can see for instance that there is a misalignment: all source segments must have a translation! In this particular case, the translator made the decision to translate source sentence six and seven as a single sentence. Your task, then, is to make sure that the segmentation is done correctly and that each source segment is aligned with the correct parts of the target. Typically, **we only edit the target segmentation, as state in Section 5.**



Note that the above example is not the only way to spot misalignment. It might very well be that the opposite is true, and that a translator translated a single source segment into two or more target sentences. This is a lot harder to spot because the tool will show an aligned segment for each source segment, but not all the target segments might then be present. **It is therefore very important to go over the alignments in detail!**

5.2.2 Fixing the segment alignments manually

```
<W cur="187" id="33" segId="2">.</W>
<W cur="189" id="34" segId="3" space=" ">Dit</W>
```

Important for us to know is that every word is assigned to a segment by a given segId in the *.src and *.tgt files. In the image, you can see that the dot (at the end of a previous sentence) is part of segment 2 (`segId="2"`), and that *Dit* is assigned to the next sentence, `segId="3"`. Let's assume, now, that the translators translated the second sentence in two sentences. That means, that we should make

this clear by replacing all segId `segId="3"` by `segId="2"` because the two sentences are part of the same translation of a single source segment.

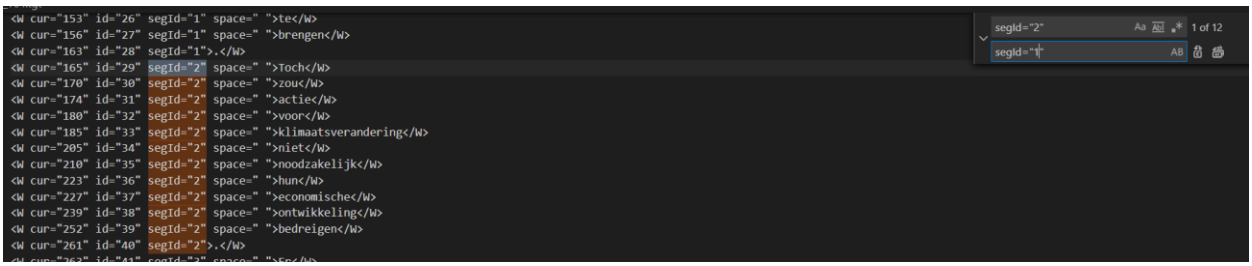
Below are some more detailed examples. **Do not forget to make your changes in the `aligns_manual_seg\` directory and not in the `aligns_orig\` directory!**

5.2.3 Double 1-to-2 example

In the example below, the translator has translated source sentence 1 in two parts (target 1 and 2), and source sentence 2 in two separate sentences (target 3 and 4).

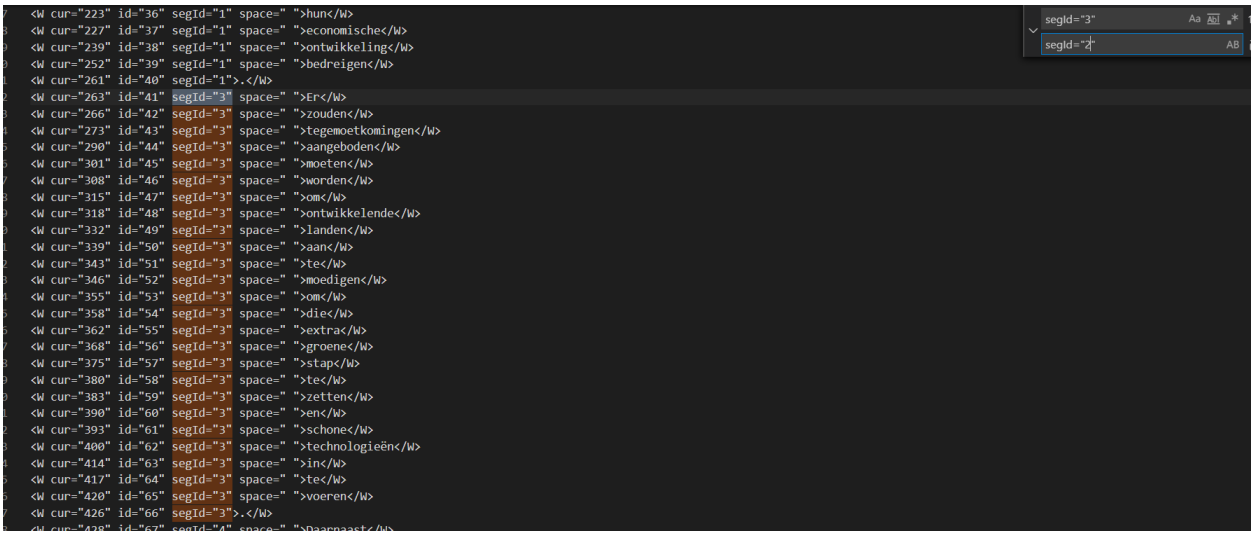
To fix this, we open up the corresponding *.tgt file.[2] It is easiest if you keep open the Yawat screen so that you can easily go back to it and check what was wrong. To recapitulate, what needs to happen is:

- Target segment 2 should be merged with target segment 1 because they are the translation of the same source segment 1

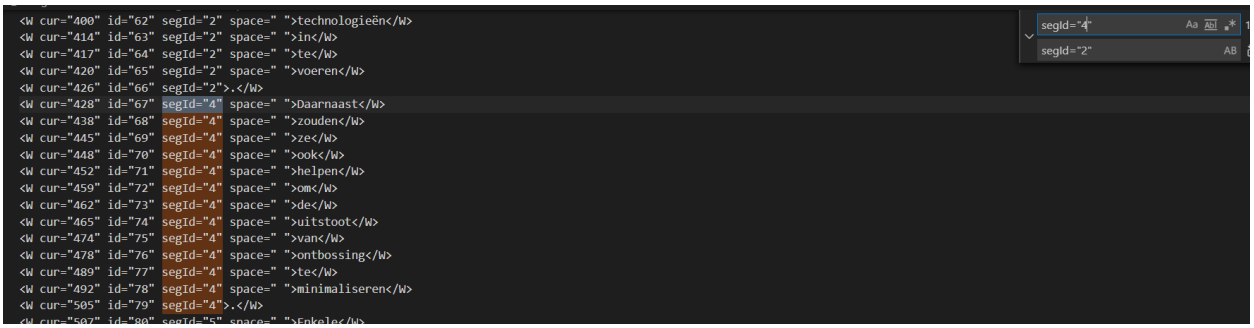


- Target segment 3 should be moved to target segment 2 (because it is aligned with the second source segment)

[2] I use Visual Studio Code to find-and-replace because it allows for only find-and-replacing in the *selected* portion of the file. This will be useful in the future. Press `ctrl+h` to open the find-and-replace window in the top-right corner.



- Target segment 4 should also be added to target segment 2, because the translator translated source segment 2 as two separate sentences



After doing that, we have fixed the problematic cases, but it does leave a gap: the file now contains words for `segId="1"` and `segId="2"`, but we replaced three and four so there are no `segId="3"` and `segId="4"`. So we have to change all the following words by moving them to ensure that all slots are filled. In other words: `segId="5"` should be replaced by `segId="3"` and `segId="6"` by `segId="4"`.

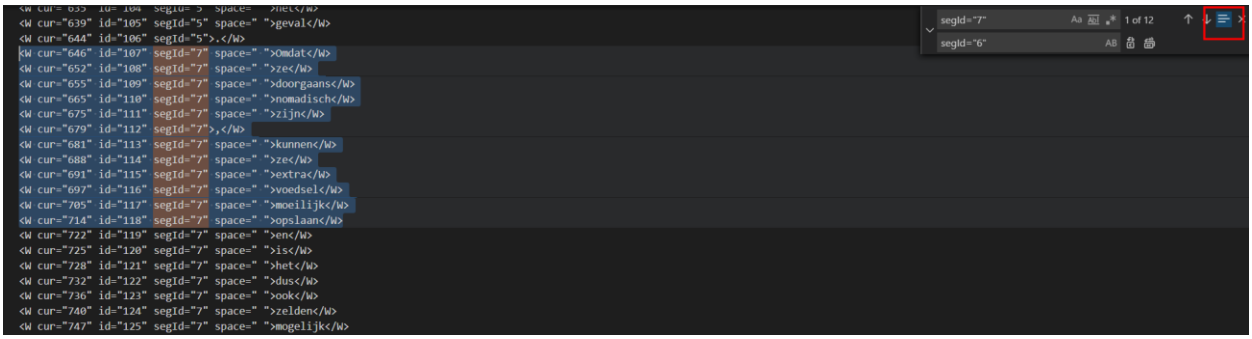
Note: always keep an eye out for mistakes in the sentences that you cannot see in the Yawat screen! Because the first two source sentences were translated as two each, the last two sentences were not visible on the target side. It might very well be, though, that another has happened in those lines. A good test is checking that the final `segId` number should always be the number of segments that you expect, in this case - looking at the Yawat screen - that is four. In addition, you can also have a look at the respective *.src file and compare it to the target file that you edited, to make sure that the alignments make sense.

5.2.4 2-to-1 example

In contrast with the previous example, translators may opt to merge two source sentences in a single sentence, for instance by using coordinated or subordinated clauses. In such an event, we have to cut the target sentence into multiple segments and align the parts with their corresponding source.

In the example below, you can see that a translator has chosen to do just that: source segment 6 and 7 are translated as a single target segment by using the conjunction *en*.

To fix this, we need to change the first part of the target sentence to be segment 6 rather than 7. This is where find-and-replace *inside a selection* is useful (toggle the button inside the red square in the example below). First select only the part that we need to change (all words of target segment 7 up to but not including *en*), make sure the selection-toggle is activated, and find-and-replace `segId="7"` by `segId="6"`. Only the words in the selected part of the file have now changed.



Note: in cases like this, I recommend to have the conjunction be part of the *second* segment. If there is a comma before the conjunction, then add the comma to *first* segment and the conjunction itself to the *second* segment. For instance, “The cookies, which I ate” can be separated into “The cookies,” and “which I ate”. Of course, depending on your study, you may choose a different approach.

5.3 Tokenization

As said before, after uploading your Translog files, they will be parsed automatically, which can lead to mistakes in sentence segmentation as well as mistakes in tokenization. In this section we will discuss how to fix the latter cases.

The src and tgt files that you downloaded consist of tokens `<W>`. The text value inside `<W></W>` is the text of that token. Its attributes are:

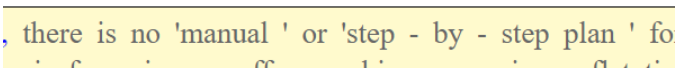
- `cur`: the cursor position at the start of this token
- `id`: this token’s index in the text
- `segId`: the segment that this token is part of
- `space`: if present, what kind of space character is present **before** this token. This is often incorrectly missing for special characters such as (and possibly others): `'"-()[]&{}`

Tokenization (and/or space attribute) might go wrong involving special characters such as: `'"-()[]&{}` even if that character is part of another token. E.g. ‘s avonds in Dutch will be tokenized as “’s” and “avonds” but “’s” will not receive a space-attribute, even though it needs it. The apostrophe ‘s here is a not a possessive but a clitic form of “des avonds”.

It might also occur when a space is used a thousand separator in numbers, e.g. 2 800 will be tokenized as “2” and “800”. I would normally tokenize this as one unit. **However**, even if you merge the tokens in the alignment files, YAWAT still seems to split each token by spaces regardless which may have

unexpected consequences. So apparently tokens (text inside <W></W>) **must not** contain spaces themselves, so do not merge those two numerical tokens.

Below are some more detailed examples. **If you need to do retokenization, copy the resegmented directory `aligns_manual_seg\` into the `aligns_manual_tok\` directory.** Make tokenization changes in this directory, and not in the `aligns_orig\` nor `aligns_manual_seg\` directory!



As an example, In the image on the left quotation marks are causing issues. Rather than having the quotation mark as a separate token, and the word as a token, they are merged together into one token by the automatic parser. This happens for both *'manual* and *'step*. To solve this, we have to modify the corresponding file.

In the image below, the original (wrong) alignment is on the left, and the corrected version is on the right. Three things to note:

- *'manual* is split up into two tokens, *'* and *manual* - *'step* is split up into *'* and *step*
- `space=" "` should be added for the tokens that have a space before them. **This is very important** and might be incorrectly missing!
- You do not have to add or change the `cur` or `id` attributes - this will be modified automatically in the next step - but you do **have to ensure that the `segId` is correct**.

```
<W cur="283" id="56" segId="4" space=" ">'manual</W>
<W cur="289" id="57" segId="4">'</W>
<W cur="291" id="58" segId="4" space=" ">or</W>
<W cur="295" id="60" segId="4">'step</W>
<W cur="299" id="61" segId="4">-</W>
<W cur="300" id="62" segId="4">by</W>
<W cur="302" id="63" segId="4">-</W>
<W cur="303" id="64" segId="4">step</W>
<W cur="308" id="65" segId="4" space=" ">plan</W>
<W cur="312" id="66" segId="4">'</W>
```

```
<W cur="283" id="56" segId="4" space=" ">'</W>
<W cur="283" id="56" segId="4">manual</W>
<W cur="289" id="57" segId="4">'</W>
<W cur="291" id="58" segId="4" space=" ">or</W>
<W cur="295" id="60" segId="4" space=" ">'</W>
<W cur="295" id="60" segId="4">step</W>
<W cur="299" id="61" segId="4">-</W>
<W cur="300" id="62" segId="4">by</W>
<W cur="302" id="63" segId="4">-</W>
<W cur="303" id="64" segId="4">step</W>
<W cur="308" id="65" segId="4" space=" ">plan</W>
<W cur="312" id="66" segId="4">'</W>
```

If you have changed the tokenization of a file, there is **an important caveat**: due to the nature of how files work together, you also need to fix missing `space`-attributes for all files with a given name (e.g. if you had to change T01.src, then you need to verify T01.tgt too and particularly make sure that the space attribute is correct for all items. **Hint**: this is typically wrong for involving the characters `'"-()[]&{}`).

Important: if you have not changed any file of a particular participant_text (so not its .src, .atag, or .tgt file), neither its segmentation nor its tokenization, you should now delete it from `aligns_manual_tok\` directory. So if you have not edited files P01_T02.src, P01_T02.tgt nor P01_T02.atag, you can delete those three from the "manual" directory. Because we do not need to edit them, we should remove them. Otherwise the scripts in the next steps will needlessly complain about them.

5.4 Running scripts

After all edits, the `id`s and `cur`s of the words are not correct, as the example above shows. `fix_tokenization.py` can fix that automatically. In addition, the script will warn you about differences between the original, unedited files, and the files that you modified. What it does is checking

whether the number of characters between the original file and edited file is the same, based on the cursor positions (`cur`). For this to be correct, it is required that you added `space`-attributes where necessary and did the tokenization correctly. Before running the script, you may want to **back up** the `aligns_manual_tok\` directory by copy-and-pasting it, just to make sure that if something goes wrong you can still recover all the manual changes that you made.

To run the script, use the following command where the first argument is the directory with the original, un-edited files, and the second is the directory containing the files that you manually edited.

```
python fix_tokenization.py ..\aligns_manual_seg\ ..\aligns_manual_tok\
```

The script will warn you of any issues that may be present. **Fix those before continuing!** If the script does not print any warnings, then you're good to go.

Next, we still have to change the atag-files. This is especially important to run when you already word-aligned some of the data. In the previous step, the script automatically updated the word ids and cur positions. However, the atag file contains the information about word alignment by linking words to each other by id. These word alignments are not correct anymore if the words' IDs have changed. Luckily the `atag_fix.py` script can solve that automatically. It will update all alignment points to reflect the new changes. Of course, if you re-tokenized words (rather than simply changing the space-attribute), it cannot deduce what to align. This implies that the alignment for words that have been re-tokenized will be removed and you will have to redo those word alignments. Therefore, it is recommend to only start word aligning after finishing this whole process.

To fix the alignments, run the following command where the arguments are identical to the previous ones: first the directory containing the original, un-edited files and second the files that you manually edited. Those files should by now have also been changed by the `fix_tokenization.py` script. Just to be safe, you may want to temporarily **back up** the data that was created by the previous script. (So for instance, you can copy the `\aligns_manual_tok` directory to `\aligns_manual_tok_bak`.)

```
python atag_fix.py ..\aligns_manual_seg\ ..\aligns_manual_tok\
```

After running the final script, the atag files are updated, too. The manual process is now finished!

Note: you should *never* run the script on the same input directory twice. Once an atag file has been fixed by the script, it cannot be changed again. In that case it would throw errors and make the script malfunction. In the event that you ran the script, but still want to make a change to one of its files **you have to copy those files to a different directory and replace the .atag file with the corresponding .atag file from your back up or from the `\aligns_manual_seg\` directory. You can the run the script on this new directory:**

```
python atag_fix.py ..\aligns_manual_seg\ ..\new_directory\
```

5.5 Re-uploading the modified files

After changing manually fixing segmentation and tokenization, it is time to upload the files. To do so, **only zip the files that you changed.** This should not be a problem because you should have already removed the irrelevant files before running the scripts in Section 5.4. This is important: the files that you upload will overwrite the existing files (although an automatic back-up is made), so to be safe you should only zip the changes. As an example, if you downloaded all alignment files for all translators and

all tasks, but only changed P01_T03.src, then just zip that file and upload it to the TPR-DB while using the same *Study name*. If you followed the directory structure suggested, you should ZIP and **upload the `aligns_manual_tok\` directory.**

If you now open Yawat again, you should the changes that you made reflected in the Yawat interface.

6 Word alignment with Yawat

See also the online guide: https://sites.google.com/site/centretranslationinnovation/tpr-db/alignment

Now that the study has been re-uploaded, we can start doing manual word alignments. In the row of the relevant study, click on *Open Yawat*. This will open a list with all participants and their tasks, also indicating how many alignments you have "completed".

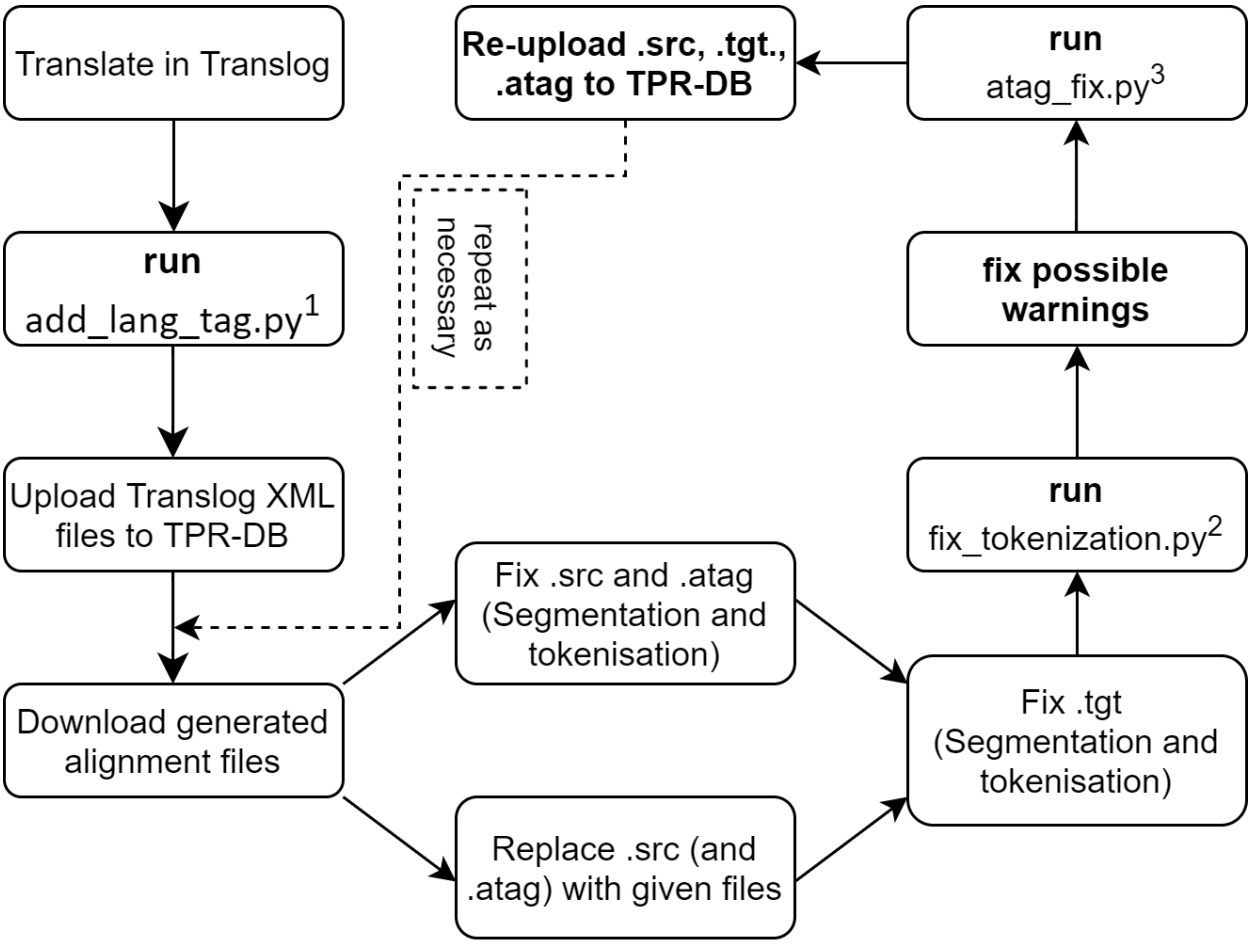
Click on one such file (e.g. *P01_T01*). Yawat will open. **You can easily align source and target words by using your mouse.** Left-click on the words that you want to align in source and target, and right-click on one of the aligned words to "save" that alignment. Aligned words are colored gray. When you hover over these aligned groups, all the words in the alignment group are highlighted. To edit a group, click on it, make changes (add or remove a word with a left click), and save the alignments again by right clicking in the group. You dissolve a whole group by right clicking and selecting "Dissolve this group".

When you have aligned a single segment, **do not forget** to click on the small *done* checkbox in the left corner above the segment. This will "save" the alignments. Finally, click on the "save" button in the top right corner of the screen.

When you are done with all your alignments, go back to the TPD overview: https://critt.as.kent.edu/cgi-bin/yawat/tpd.cgi. You should see that "Save Yawat Alignments" is colored red every time you made changes to the word alignment of the study, indicating that your latest changes have not been fully saved. **Click that button before continuing!!! This is very important; you may otherwise lose the progress that you made!** The red color should disappear after clicking the button.

Now that all alignments are complete and saved, the data tables can be generated. Click on *Make Tables*. This will run a script that will calculate different features by using the Translog files and the manual alignments. After the server is done running the script, you can download the generated tables by clicking on *Download tables*. Download the tables to your `tables\` directory, and download the final alignments to `aligns_final\`.

Flow chart



1 Adds Languages tag automatically to XML

2 Fixes id and cur attributes in .src and .tgt after manual re-tokenisation. Warns you if the number of total characters (incl. spaces) does not match with the original files

3 Modifies .atag to try to map previous word alignments (if any) to the new IDs that were created with 2. Sentence pairs where words were re-tokenized will have to be re-aligned. If you did not word align before running this script, it will not have any effect