Guide and Notes for the Belgian Parliamentary Data Converter

Tom Heremans

MA DTA: Text Mining

Date of publication: 2025/05/31

Index

Exploring the Source Data

The records of the Belgian Parliamentary debates contain a certain structure that could help us to extract the right information. The digitized files are based on the Integral Reports of the Chamber of Representatives (*Integraal verslag/Compte rendu intégrale*), which over the course of 195 years have evolved and changed. Since I will try to process all the available debates since 1830 till 2024, the text structure changed a bit over the course of the centuries. By mapping the most predominant elements of change, one can differentiate the approach to extract the wanted information. The goal is making Python lists of the speeches by the parliamentarians of the Belgian Parliament which isolate the speakers and connecting them to the speech they gave. The speech in general has been in French or Dutch. The preprocessed JSON-files I received by TextGain, isolated the paragraphs as published and showed which language was spoken. I received these text as published on Plenum, and they contain the following structure:


```
{
"paragraph": 11,
"text": "09.05 Gitta Vanpeborgh (sp.a): Mijnheer de minister, uw
woorden stellen mij alleszins gerust. Ik hoop van harte dat die
woorden ook geruststellend zullen zijn voor alle instanties en
zorgverstrekkers die eerstelijnshulp geven en in het bijzonder voor
de mensen die nood",
"language": "nl"
                                                    },
```


Source: 55_ip074.json (TEXTGAIN), Plenary session, 10 December 2020.

A paragraph number (paragraph), the actual speech in that paragraph (text) and the language used in the paragraph (language). German speech is not that common, but falsely labelled Dutch in the Plenum-source file when it does appear. Some OCR-transcriptions are not fully correct, so small mistakes may appear. And, in general a small discrepancy can appear between the spoken text and the transcribed text in the original source.

For comparison:

Published Transcription

PM Leo Tindemans, Session of October 11, 1978:

"*Wanneer ik in de speciale commissie het standpunt van de regering heb verduidelijkt, heb ik gezegd dat « de Grondwet voor mij geen vodje papier is ». Ik wil niet dat avonturiers van links of van rechts ooit een precedent van Tindemans zouden inroepen om hun daden goed te praten. Daarmede rekening houdend, na de scheldwoorden, beledigingen en aanvallen waarvan ik de laatste dagen het voorwerp ben geweest, en zoeven nog de insinuaties op deze tribune, is de conclusie voor mij duidelijk : ik verlaat deze tribune; ik ga naar de Koning en bied Hem het ontslag van de regering aan*."

Literal transcription

PM Leo Tindemans, Session of October 11, 1978:

"*Ik heb in de commissie gezegd wanneer ik het standpunt van de regering bekend maakte: "Voor mij is de Grondwet geen vodje papier"! En ik wil niet hebben dat morgen*

avonturiers van links of rechts het voorbeeld van Tindemans zouden inroepen om avonturen goed te praten in dit land. Daarmee rekening gehouden, na de scheldwoorden en de beledigingen en de aanvallen die ik de laatste dagen heb moeten incasseren, tot daar even nog de insinuaties op deze tribune, zeg ik: "Voor mij is er maar één conclusie. Ik ga van deze tribune weg, ik ga naar de Koning en ik bied het ontslag van de regering aan."[1]

These are small differences in wording, but apparent. The transcription supposedly says also that "The CVP applauds the standing Prime Minister." The CVP was the party of the PM. The applause is mentioned, but on the video footage of the session booing can also be heard. Literature on session transcriptions also shows that writers of transcriptions are often influenced by the ruling power, personal relations towards speakers and own writing styles.[2] This influences shows often only in small differences on wording, but could have a small influence on possible future results.

The ultimate goal was to create a dataset with a build up like this:

`[Speaker Name, paragraph n, paragraph n+1, ..., language]`

The goal was making a list, split on marker that separates the speaker from the spoken/transcribed text. In this case it is a column (:), but also a dash has been used (—). They are also used in different contexts throughout the text, so just making a split on these markers is not useful. Introductions to the sessions also have to be cleaned and deleted if they are not useful.

On Speakers and Speeches

An overview was needed of all existing speaker-speech structures that exist in these documents. I've detected four structures that are present in all documents over a given period. Some of these periods cover more than a one hundred years, while others are only present for a few years. The notebooks that construct the datasets are divided based on these four periods.

First Speaker structure/ 1830-02-12

"M. L'ABBÉ DE FOMBE : La monarchie héréditaire à titre de perpétuité …"

This consists of a name in uppercase letters, often followed by a title like "M." and finished by a column. General announcements are made by the guillemet quotation mark "»". Sometimes the column has been misinterpreted by the OCR, and changed by a number like "3" or "4".

[1] S.n., *Fragment Leo Tindemans, Session of October 11, 1978*, <https://www.youtube.com/watch?v=oksUHgfO6o4 >, consulted 17 February 2025.
[2] Benjamin MOREL, « Ce que conte le compte rendu : l'institution d'un ordre parlementaire idéalisé », *Droit et Société*, 98/2018, 179-199.

Second Speaker structure/ 1838-12-12

“M. DE LANGUE. — Je crois, messieurs, que dans les circonstan- ces …”

Still an uppercase name and an added dot behind it, with a new sign: the dash. However, titles are often transcribed in a non-uniform way. Sometimes the title of a minister is partially in lower and uppercase, eg. “Ministre des FINANCES”. This also due to the OCR. One problem arises. There is no significant difference between headings like “SOMMAIRE —”, only the dot behind the speaker.

Third speaker structure/ 1841-01-01

“DA. Vandenbossche. — Messieurs, j’appuie la proposition : du …”

The name is now spelled in a more conventional way, only with uppercase letters when needed and otherwise lowercase. However the title “M.” is often not recognised well by the OCR, only gradually progressing over time. Often also accompanied by a function when the speaker is a minister or the speaker of the Chamber. The difference between headers, like mentioned before, is bigger due to the difference in casing. Some files, like K00320030, do split the speaker already from the spoken text. Sometimes a naked dash is also noticed, one without accompanying speaker and only text behind it. The title or context is also sometimes placed between brackets eg. “De heer Heyman (op het spreekgestoelte) —”. A difference also appears from the 1930’s onwards. Dutch speakers are not using the title “M.” anymore, but are called “De heer” or a female variant “Mlle”, “Mme” and the Dutch “Mevr.” are more similar.

There were also a few mistakes in OCR that regularly appeared eg. “M. le president.-- -” with a dash and a hyphen. Sometimes a governing function also has a changing upper and lowercase element.

Fourth Speaker Structure/ 1991-12-16

“De heer L. Peeters (SP) : Mijnheer de voorzitter, mijnheer de minister, ik denk dat de cijfers voor zich spra ken, ik kom daarop dus niet terug.”

"Mme de T’Serclaes (PSC) : Alors,"

The fourth structure uses the older format by using a column. And often an abbreviation of the first name when handling a common last name. The party/political group of the speaker is mentioned in between brackets. This is not the case with ministers or the speaker of the Chamber. The titles are similar to the previous style. Titles for women can also take the form “Mw.” and later on this gendered title disappears fully. The title of the political group is always uppercase, and with a dot for abbreviation when needed. Sometimes the name is also preceded by numbers to assign the speaker to a certain posed question that appeared earlier in the original text.

General Remark

Some words were split in the original text to fit the lay-out, but this is also reflected in the OCR. Like in one of the examples above, eg. “spra ken”, a space or hyphen is present in the transcription. Another difficulty arises after August 23rd 2001, when every speech is translated along the original. This is reflected in the batches I received from TextGain, because often French and Dutch lines are not in sequence anymore and do not always reflect the speech of some MP’s who switch during their interpellation. In the digitized versions this resulted in text being attributed to the wrong MP and I could not find a real solution to this problem. The code is not reliable for all texts between August 23rd 2001 and December 19th 2024, when the Integral Reports shifted back to a pre-2001 make-up.

Most of the written reports of the debates lack any note of party affiliation of individual MP’s. Only from 1991 onwards is this incorporated. So I had to find out a way to have a certain exhaustive standard concerning metadata, that could be applied to every speaker since 1830 till now. In the following chapters I will delve deeper into the code I used to obtain the data for the dataset.

Preparing the JSON-files for Analysis

After you've obtained the raw JSON-files by Plenum or TextGain, some rearrangements need to be made. The original files need to be analysable and grouped together on a basis or unit. I've chosen to group these JSON-files of the reports based on the legislature they appeared in. At the moment of writing 57 different legislatures came about in the Belgian Parliament. (Including legislature 0, the constituent assembly a.k.a. the National Congress.) This is not the only unit of analysis, and one can refine the following code to ones wishes. By following the next steps, the files of the debates get copied or moved to a folder corresponding to the legislature. Simple adjustments could also make a year or group of years another unit of analysis.

Locating the JSON-files

Before the files can be copied, your computer needs to know where your files are. Most JSON-files for the legislatures are not ordered alphabetically, so the corresponding file needs to be matched in advance. One way to let your computer know which files are which, is making a TXT-files with all the paths of a certain legislature. Luckily there is an Excel-file with metadata on all the sessions and the corresponding file names. The following script matches the right file names with the right legislature, based on the metadata from the Excel-file. This information is transformed in TXT-file with the corresponding path names

Making a TXT-File with PATH names

First import the Pandas Library in Python:

```
import pandas as pd

session = pd.read_excel(r"path name to sessions\sessions.xlsx")

session
```

Inspect your dataframe and see if any changes appeared that you could implement in the following code. Then make a list of all numbers that represent a legislature:

```
legislat_list = []

for each in session["legislature"].values:

    if each in legislat_list:

        continue

    else:

        legislat_list.append(each)
```

It extracts all numbers that appear in the column "legislature" and puts them in a list. If the list already Now the variable legislat_list should contain all numbers from the legislatures. To extract the years one needs to make a simple adjustment and change `session["legislature"]` to `session["date"][:4]`.

Next piece of code finally prepares the TXT-files we need for matching:

```
for each in legislat_list:

      with open(f'Legislature_paths_{each}.txt', "w", encoding="utf-8") as dot:

            legislature = session[session["legislature"] == each].values

            for element in legislature:

                  dot.write(rf"your personal path to the file folder
                  \{element[0]}.json"+"\n")
```

This piece of code iterates over the list with all the legislature numbers. It opens and creates the new TXT-file by adding the sign "w". Once your file is created, instantiating this code will result in overwriting it. This can be handy when your source folder changes. This gets stored in the variable in dot. The variable "each" contains the number of the legislature in question and enables us to extract every row in our dataframe that is of a specific legislature. The last for loop iterates over every row that is stored in the variable "legislature" and adds the name of the file to the string that is written into the new TXT-file. The '+ "\n"' makes sure that every path is stored on a new line in the TXT-file.

Creating Folders and a BAT-file

I've done this through a Terminal simulator like GitBash (Apple and Linux proof), but this is also possible to do in Windows Command Prompt.

First target folders need to be created where the files will be stored. Go with the Terminal or Command Prompt to the folder you have stored your Legislature_path_x.txt-files and make the new folders with this command:

```
$ mkdir legislature{0..56}
```

This creates 57 folders with names that correspond to the legislature numbers in the Excel file.

Next piece of Python code creates 57 BAT-files that should transfer/copy/move the original files to the corresponding folder:

```
for each in legislat_list:

    with open(f'execute_copy{each}.bat', "w", encoding="utf-8") as dot:

        dot.write("@ECHO OFF\n")

        dot.write(rf'FOR /F "usebackq delims=" %%I IN ("personal
path\Legislature_paths_{each}.txt"') DO ('+"\n")

        dot.write(rf'xcopy %%I "personal path\legislature{each}"'+"\n")

        dot.write(')\n')

        dot.write('PAUSE')
```

This prepares the BAT-files that will store the code that will do the copying. The BAT-files can be instantiated in the Command Prompt, Terminal or GitBash by going to the right folder

and just running the file. You will need to do this for every file, and it may take a while to complete it for all 57 legislatures.

Separating the Spoken from the Speaker

For the extraction of speaker-speech information, a Python script needed to be made that could detect the difference and put them in Python lists. These lists could later on be converted to a Pandas DataFrame format that prepares a functional dataset. The four speaker structure styles were to be detected and isolated. The ultimate dataset should also contain metadata concerning dates, parties and source files. The goal was to create a simplified version of datasets inspired by how datasets of the People&Parliament project (Utrecht University/University of Jyväskylä) are made.[3]

Looking for a separator

I've explored two ways to look for the speaker in the texts: NER (named entity recognition) and classic rule based separation. First of all, for the rule based separation I focused on structures separating speakers from speech like dots and hyphens, but these do sometimes appear outside the scope of the speaker-speech relations. Total random stuff gets seen as a speaker and blurs the effectivity of my research. To get more specific results I made regular expressions to filter out any mistakes, but this makes the search for speakers heavier. For speaker structure four this causes the code to work on a JSON-file for around 4 to 10 seconds. To preprocess a full legislature it takes around two hours to generate output, that is incomplete. Speaker structure three is more simple in nature and takes just a few minutes to do the same task.

I needed to get a more simple, faster and cleaner approach to separate speaker from speech. The first advice I got was to get SpaCy to search for personal nouns. By using a French and Dutch NER-model on the JSON-files I tried to find out if it could recognise the speakers, but the overall performance did worse than my rule-based model.

Adapted rule-based splitter(s)

After consideration, I decided to go for a rule-based model and adapt it so the names of the speakers are standardized and the amount of blur in the data is minimized. No non-speaker element will be identified as being a speaker and this would also diminish the number of misidentified speeches. However, this code still makes mistakes. I will explain the code I made in the following paragraphs, but only applied to speaker structure three. The code I personally used was for this speaker structure and all other models are ultimately derivations from this code. Each with slightly different adaptations to better fit their periods and styles, but with the same principles. So they will be included, but not delved upon further. In the next chapter I will also delve deeper in the annotated files I made to clean up any noisy data.

[3] I went to their Workshop on 20 March 2025 in the KB of The Hague where they showed their approach to digitally preparing Parliamentary Data. My approach is largely different due to budget and time constraints, but I need to mention the help of dr. Pim Huijnen, dr. Pasi Ihalainen and MA. Mees Van Stiphout.

How does it look and work?

```
import json

import re

parl_seat = [33, 34, 35, 36, 37, 38, 39, 40, 41] #Fill here the legislatures you want to  search
col_list = []

for seat in parl_seat:

    parl_df = pd.read_excel(rf"speaker_lists/speakers_leg{seat}.xlsx")

    path = rf'’ \legislature{seat}' #Path to folder with all files of a legislature

    obj = os.scandir(path)

    print(seat)

    for entry in obj:

        with open(rf"legislature{seat}/{entry.name}", "r", encoding='utf-8') as f:

            session = json.load(f)

            countdown = len(session['pages']) #Looks to the length of a session

            counter = 0

            heading = 0

            pattern = re.compile("([a-z]|\))\.\s-?—") #This pattern notices a speaker structure 3

            suite_fin = re.compile("\(Suite\set\sfin\.\s—") # ending element in the early debates

            presid = re.compile("([Pp]résident|[Vv]oorzitter)\s—")#a common OCR mistake

            while counter != countdown: #when countdown and counter are the same the session has been wholly overed

                for each in session['pages'][counter]['paragraphs']:

                    check = pattern.search(each["text"]); #searches the potential of a new speaker in the
new paragraph

                    check_big = re.search('\.\s—', each["text"]);

                    pointer = re.search('\.\—', each["text"]);

                    num_check = re.match('[0-9]+(\.|°)', each["text"]); #Looks to formal summaries

                    suite = suite_fin.match(each["text"]);

                    presid_check = presid.search(each["text"]);

                    double_check = re.search('.\s—-', each["text"])


                    if num_check and heading == 0: #finds summaries in the start of the text

                        continue

                    elif suite:

                        continue
```

```
                    elif each["text"].isdigit():
                        continue
                    elif check or presid_check:
                        if check_big and each['text'].split('. —', 1)[0] in
parl_df["names"].values:
                            new_entry = each['text'].split('. —', 1)
                            load_dataset = parl_df[parl_df["names"]==new_entry[0]]
                            col_list.append([session["date"], load_dataset["uniform
name"].values[0], new_entry[1], load_dataset["affiliation"].values[0], each['language'],
seat, session["session"]])
                            heading +=1
                        elif pointer and each['text'].split('.—', 1)[0] in
parl_df["names"].values:
                            new_entry = each['text'].split('.—', 1)
                            load_dataset = parl_df[parl_df["names"]==new_entry[0]]
                            col_list.append([session["date"], load_dataset["uniform
name"].values[0], new_entry[1], load_dataset["affiliation"].values[0], each['language'],
seat, session["session"]])
                            heading +=1
                        elif presid_check and each['text'].split('—', 1)[0] in
parl_df["names"].values:
                            new_entry =each['text'].split('—', 1)
                            load_dataset = parl_df[parl_df["names"]==new_entry[0]]
                            col_list.append([session["date"], load_dataset["uniform
name"].values[0], new_entry[1], load_dataset["affiliation"].values[0], each['language'],
seat, session["session"]])
                            heading +=1
                        elif double_check and each['text'].split('. --', 1)[0] in
parl_df["names"].values:
                            new_entry =each['text'].split('. --', 1)
                            load_dataset = parl_df[parl_df["names"]==new_entry[0]]
                            col_list.append([session["date"], load_dataset["uniform
name"].values[0], new_entry[1], load_dataset["affiliation"].values[0], each['language'],
seat, session["session"]])
                            heading +=1
                        else:
                            if heading == 0:
                                continue
                            else:
```

```
                                col_list.append([session["date"], col_list[-1][1],
each['text'], col_list[-1][3], each['language'], seat, session["session"]])

                    elif heading == 0:

                        continue

                    elif each['text'].isupper():

                        continue

                    else:

                        col_list.append([session["date"], col_list[-1][1], each['text'],
col_list[-1][3], each['language'],seat, session["session"]])#appends text to the last speaker

                counter += 1
```

First the user needs to give the numbers of the legislatures he/she/they want(s) to investigate. These numbers will be connected to open the folders and files relevant to that legislature. After a few variables for storage are set up, the code proceeds to obtain the correct information. It opens the JSON-file the user wants to use as the variable "f". With "json.load()" this file gets loaded and stored in the "session" variable so it's ready for use. The "countdown" variable stores how many pages the original session transcription had. The counter needs to keep up with the index to communicate that to the session variable. By putting the counter outside the for-loop, it will follow the number of pages the original transcript had. When the counter and countdown reach the same value, the while-loop will stop.

Various variables containing regular expressions that search for speaker-like structures are loaded. Further on they are used to notify if a speaker structure occurs. If the speaker structure occurs in a preloaded dataframe with all correct speaker formats it will be permitted to be retrieved and stored as a speaker-speech relation. The same dataframe also contains a standardized speaker and the party the speaker belonged to in that legislature. These are added to the Python list.

A numeric checker looks for annotations that only exists of headers that use the summation form. This can't be used in all debates, since in the fourth speaker form it is also used to refer to posed questions. The moment a speaker structure gets noticed, the code creates a new list for a new speaker. Since OCR sometimes creates two different forms, two codes are deployed to search for speakers. When noticed a split-method creates a list that separates the speaker from the speech. Another element adds the language spoken in the paragraph.

Paragraphs that only contain upper text also gets removed. This is often a header. A last part appends paragraphs without speaker element to the previous one. It assigns the previous speaker to the new paragraph and signals which language the speaker uses.

Metadata provided in the JSON-file are also added through the session[""] variable in each list.

These lists are collected in the col_list variable and later to be used in creating a Pandas Dataframe which offers the possibility to work with the retrieved data. This has been done for every speaker structure. Although there are differences, the process is largely the same.

Some notes on the created Pandas Dataframe

By adding the col_list variable to the following code, a pandas data frame will be created of the legislatures of the collected data.

```
df = pd.DataFrame(col_list, columns=["Date", "Name", "Speech", "Party", "lang", "Legislature", "Session_ID"])

df["Date"] = pd.to_datetime(df["Date"])
```

The data frame contains the following data;

- Date: the date of the session (yyyy-mm-dd)
- Name: a standardized version of the speaker's name
- Speech: each paragraph in the original source files get's seen as a unit of speech
- Party: party or ideological affiliation of the speaker
- lang: the language of the speech (fr: French, "nl": Dutch, "na": none) German is left out, but present sometimes.
- "Legislature": the legislature number
- "Session_ID": the name of the source file said speech is coming from

Now the data frame is ready to be played with, however it's too large to export. In the notebooks I split the data frame in Dutch and French speech. I also removed the speech by the President. This one contains a lot of noise and his or her speech in itself is often noisy. One can choose to leave it in and add more detailed information on the speaker.

The Dutch/French split makes it easier to tokenize the files in their respective languages. It gets rid of some noise sorted under "na" language, none identifiable language tokens. And the set is smaller and linguistically more uniform. By importing the Python library "nltk" and its word tokenizer, the data set becomes searchable and thematically groupable. The words/themes you want to use and analyse can be extracted for later use.

E.g.:

```
import nltk

from nltk.tokenize import word_tokenize

nltk.download('punkt_tab')

import string

parl_fr = parl[parl["lang"] == "fr"]

parl_nl = parl[parl["lang"] == "nl"]


def remove_punct(input_string):

    result = ''.join([char for char in input_string if char not in string.punctuation])

    return result


parl_nl["Tokenized_Speech"] = parl_nl["Speech"].apply(remove_punct)
```

```
parl_fr["Tokenized_Speech"] = parl_fr["Speech"].apply(remove_punct)


parl_nl["Tokenized_Speech"] = parl_nl["Tokenized_Speech"].apply(word_tokenize)

parl_fr["Tokenized_Speech"] = parl_fr["Tokenized_Speech"].apply(word_tokenize)

Theme = 'Communism' #Set theme or topic, for later formalization

woordenlijst_nl = ["kommunisme", "kommunist", "kommunisten", "communist", "communisten",
"communisme", "bolsjewist", "bolsjewisten", "bolsjewisme", "bolsjevisme", "bolsjevist",
"marxisme", "marxisten", "marxist"]

vocabulaire_fr = ["communisme", "communiste", "communistes", "bolchevisme", "bolcheviste",
"marxisme", "marxiste", "marxistes"]


parl_nl = parl_nl[parl_nl["Tokenized_Speech"].apply(lambda word_list : any(word in
woordenlijst_nl for word in word_list))]

parl_fr = parl_fr[parl_fr["Tokenized_Speech"].apply(lambda word_list : any(word in
vocabulaire_fr for word in word_list))]
```

This generates a dataset concise and compact enough to be quickly analysed, vectorized and do anything you want. Although the check is double checked by comparing the split word groups to a list of speakers, some text specific distinctions still slip through despite it. For example, in the session of 5 November 1959 a member of the PSC (Christian democrats) yells at Gaston Moulin, member of the PCB. This is noted as followed : “'Une voix SUR UN BANC P.S.C. : Comme en Russie!” This is a pejorative use of Russia against a Communist Member of the Chamber, but is placed under Moulin’s speech because it doesn’t follow the speaker structure. If you want to research such expressions, than this is difficult under the current split method. It’s a small minority of texts, but still apparent.

Annotated speaker lists

The code checks the split token group that is assumed to be a speaker to a list of speakers to ensure that it really is a speaker and ultimately generate a cleaner dataset. I generated all elements that the original code identified as a speaker, and placed them in CSV or Excel files. Then I annotated all names and mistakes, got rid of the latter and annotated all names.

I used this code for the isolation of the speaker data after running the original code without the data frame check I later implemented:

```
[Implement code of the speaker splitter]

parl = []
for session in session_collection:

       for speech in session:

        parl.append(speech[0])

parl = set(parl)
parl = list(parl)
print(parl)
```

This creates a list of all speakers in a batch of files. It turns also the list into a set, so there every speaker appearance gets reduced to the one appearance in the final output. Then the set gets casted again into a list so it is more easy to handle.

```
import pandas as pd
import numpy as np

d = {"names":parl, "tag": np.nan}
df = pd.DataFrame(d)

print(df)

df.to_csv("speakers_leg52.csv", sep='\t', encoding='utf-8', index=False, header=True)
```

This code puts the list of all the speaker elements in a dictionary. This dictionary prepares to provide the framework for a dataframe with names in one column, and an empty column to add tags. This dictionary is made into a dataframe which gets exported as a CSV-file. These files can be found on the GitHub.

Annotation choices

After I got rid of all mistakes, I introduced two extra columns giving the following column names: “name”, “uniform name”, “affiliation”, “tag”. All word groups identified as names are linked to real people and their ideological affiliation. The uniform name get’s the form: “Surname, first name”. I based myself on the name list in Emmanuel GERARD, Els WITTE, Eliane GUBIN, Jean-Pierre NANDRIN, *Geschiedenis van de Belgische Kamer van Volksvertegenwoordigers, 1830-2002*, Brussel: Kamer van Volksvertegenwoordigers, 2003. This book is also available in French. I used mainly the names as mentioned in this book as well the party affiliations as mentioned in the lists. They are periodically defined and ready to be used. I also used the party names per legislature as mentioned by the Ministry of Internal

Affairs.[4] Except when uniformity was possible or preferable. For women MP's I mainly used their maiden names, except when they themselves preferred another one. Especially for most women before 1990, a combination of their last name with the one of their husband's was common. All presidents are noted as Président_Voorzitter and their party as NA or UNK. It's certainly possible to personalize them, albeit difficult for some legislatures.

List of used party affiliations annotations

BSP: Belgian Socialist Party (Flemish department)

BSP_PSB: unitary Belgian Socialist Party

Christelijke Volkspartij: Flemish Christian Social Party, Christian Democrats. Here written in full to avoid confusion with the Christene Volkspartij (1893) which had a Daensist agenda.

CVV: (*Christelijke Vlaamse Volksunie*), Christian Flemish People's Union predecessor to the Flemish nationalist Volksunie.

ECOLO-AGALEV: parliamentary fraction of ecologist Francophone ECOLO and Flemish Agalev. In most documentation they are mentioned together, but it is not clear when this fraction formed.

ECOLO-GROEN: see *ECOLO-AGALEV.*

FN: (*Front National*) Francophone far-right party

FDF : (*Front démocratique des francophones*) social liberal, francophone interest movement

FDF-PRL: cartel of PRL and FDF.

FDF-RW: cartel of RW and FDF.

Front Wallon: (*Front Wallon pour l'Unité et la Liberté de la Wallonie*) Walloon nationalist party coming from a syndicalist milieu. Merged with Parti Wallon des Travailleurs.

KPB : Flemish wing of the Communist Party of Belgium.

KPB_PCB : unitary Communist Party of Belgium.

LP_PL: unitary Liberal Party before 1961.

Parti Wallon des Travailleurs : trotskyst party with Walloon regionalist tendencies.

Parti Social Indépendant : far-right poujadist party.

PCB : Francophone wing of the Communist Party of Belgium.

PLP : (*Parti pour la Liberté et du Progrès*) Francophone wing of the Liberal Party.

PRL : (*Parti réformateur libéral*) Francophone Liberal Party.

PSB : Francophone wing of the Belgian Socialist Party.

[4] https://verkiezingsresultaten.belgium.be/nl, available in French, Dutch or German.

PSC: Francophone Christian Social Party.

PSC_CVP : unitary Christian Social Party.

PVV: Flemish Liberal Party

PVV_PLP : unitary Liberal Party between 1961 and 1972.

'PVV_PLP_BRUX': the Brussels wing of the Liberal Party was divided after the linguistical debates in the Liberal Party following the events of 1968. It was not always clear from the documentation who belonged to which party, so I put all Liberals in the election of 1971 from the voting district of Brussels-Halle-Vilvoorde in this category. Further investigation is needed.

Rassemblement National: far-right splinter party of the Christian Social Party.

ROSSEM: (*Radicale Omvormers en Sociale Strijders voor een Eerlijker Maatschappij*) libertarian party around the figure of Jean-Pierre Van Rossem.

RSCL: (*Rassemblement Social Chrétien de la Liberté*) anti-communist splinter party of the Christian Social Party.

RW : (*Rassemblement Wallon*) Walloon nationalist party.

Vlaams Blok: far-right Flemish nationalist party.

VLD: (*Vlaamse Liberalen en Democraten*) Flemish Liberal Party.

Volksunie: Flemish-nationalist party.

UDB : (*Union Démocratique belge***)** leftist Christian Democratic Party coming from the resistance movements of the Second World War.

UDRT-RAD: (*Union Démocratique pour le Respect du Travail/ Respect voor Arbeid en Democratie*) right wing liberal populist party.

UNK : for all people whose political affiliation could not be identified.