The SpeechRecorder — MAUS — EMU-SDMS Toolchain

Most phonetic research projects loosely adhere to the following work flow:

1. Record speech
2. Automatically annotate speech (using ASR / forced alignment techniques)
3. Manually check and correct the automatically generated annotations
4. Analyze speech (connecting the primary data with annotations and derived and complementary signals)

The EMU Speech Database Management System (EMU-SDMS) is largely focused on steps 3 and 4 of this work flow. However, two other tools can be used in conjunction with the EMU-SDMS to complete the work flow: SpeechRecorder and MAUS (or, more broadly speaking — and more correctly, for that matter — the BAS Web Services). This chapter describes how these tools can be systematically combined.

What do SpeechRecorder and MAUS do?

"SpeechRecorder is a platform independent audio recording software customized to the requirements of speech recordings" (http://www.speechrecorder.org/). To this end, SpeechRecorder lets you define prompts that participants will read (or otherwise react to) while you are recording them. At the end of a session, instead of one large recording of the whole session, you have a set of smaller audio recordings, each one representing a single prompt.

MAUS (Munich AUtomatic Segmentation) processes audio recordings with corresponding orthographic transcriptions, and outputs (1) a corresponding phonetic transcription and (2) a segmentation of the signal into individual speech sounds. As such, MAUS is a part of the BAS Web Services. [1]

Different types of prompts

Prompts are very often sentences that participants read out aloud (producing *read speech*). However, this need not be the case. Prompts may as well be, for example, images that participants have to name or describe, or written questions that participants answer.

In terms of processing, *read speech* has the advantage that the researcher already has orthographic transcriptions of all recordings, because the prompts *are* the transcriptions (this neglects hesitations, misread utterances, etc.).

For all cases besides read speech, transcriptions have to be prepared. For read speech, when hesitations etc. need to be considered, the existing transcriptions need to be corrected (per token). For the time being, this is out of the scope of this book.[2]

Combining the tools

The order of the processing steps is immutable: Initially the files have to be recorded, then annotated and then analyzed. However, there are different strategies of passing data between the various tools. For example SpeechRecorder could *export* files into the `emuDB` format (future feature of SpeechRecorder - see online road map) or alternatively the EMU-SDMS could *import* files stored in SpeechRecorder's format. After all, they are separate tools and can be used individually (although combining them makes a lot of sense). In this section, we describe the best current practice to combine these tools that tries to keep the conversion work down to a minimum. We will import SpeechRecorder's recordings and prompts into the EMU-SDMS and then use `emuR` to send the data to MAUS and other BAS Web Services.

[1] Strictly speaking, MAUS is used in conjunction with G2P and possibly the Chunker service to achieve this result. The whole package is often referred to as MAUS, however.

[2] It will be covered, at a later time, in this or a separate chapter.

Importing recordings and transcriptions into Emu

Using the import_speechRecorder() function

The `import_speechRecorder()` function is in the making, but unfortunately not finished yet. We therefore have to resort to a different strategy for importing SpeechRecorder's results into Emu:

Using intermediary text (`.txt`) files

SpeechRecorder, per default, saves one `.wav` file for each prompt. With additional settings, it can be made to save an additional `.txt` file containing the corresponding prompt along with each `.wav` file. This is great because it is exactly what MAUS needs as its input.

The following options must be configured *before recordings are made*:

- Under "Project / Preferences... / Annotation":
  - Under "Persist", tick the check-box that says "Simple text loader writer for annotation template for MAUS processing"
  - Under "Auto annotation", tick the check-box that says "Prompt template auto annotator" (note that the two checkboxes are very similar. Make sure to tick the right one.)
- In your script:
  - If you edit the XML file directly: Make sure each of your `<mediaitem>` elements has the attribute `annotationTemplate="true"`
  - If you use the integrated script editor: Make sure to tick the check-box "Use as annotation template" for *every recording*.

After your recording session, you will have audio and text files. In `emuR`, this combination is called a *txt collection*. Hence, we will use the function `convert_txtCollection()` to convert the *txt collection* produced by SpeechRecorder's into the `emuDB` format. In the following example, we will use the sample txt collection included with the emuR package.

```
# Load the emuR package
library(emuR)

# Create demo data in directory provided by tempdir()
create_emuRdemoData(dir = tempdir())

# Copy this path and open it in your file manager
tempdir()

# Import the sample txt collection.
# When used with real data, sourceDir should point to the RECS directory of your
# SpeechRecorder project.
convert_txtCollection(dbName = "myEmuDatabase",
                      sourceDir = file.path(tempdir(), "emuR_demoData", "txt_collection"),
                      targetDir = tempdir())
```

In your temporary directory, you will find a folder named "myEmuDatabase_emuDB", containing a json-file "myEmuDatabase_DBconfig.json" and a subfolder called "0000_ses". Within its subfolders, e.g. in "msajc003_bndl", you will find a wav-file and a json-file, which consists of the following lines only:

```
{
  "name": "msajc003",
  "annotates": "msajc003.wav",
  "sampleRate": 20000,
  "levels": [
    {
```

```
      "items": [
        {
          "id": 1,
          "labels": [
            {
              "name": "bundle",
              "value": ""
            },
            {
              "name": "transcription",
              "value": "amongst her friends she was considered beautiful"
            }
          ]
        }
      ],
      "name": "bundle",
      "type": "ITEM"
    }
  ],
  "links": []
}
```

Use `load_emuDB()` to load this database into `R`

```
dbHandle = load_emuDB(file.path(tempdir(), "myEmuDatabase_emuDB"))
```

Now, we have an Emu database. You can inspect it using

```
summary(dbHandle)
```

or visualize it in your browser via

```
serve(dbHandle)
```

The database contains all of our recordings, and exactly one type of annotation: orthographic transcription. No phonetic transcription, no segmentation. The next section is concerned with automatically adding these to the `emuDB`.

Feeding the data into MAUS

The `.wav` and `.txt` files could have been uploaded on the BAS web site, but we will do it using `emuR`. This is generally less error-prone and requires less manual work. Moreover, since it is a scripted way of doing things, we can reproduce it reliably.

To process the data, we make use of several of emuR's functions called `runBASwebservice_...`. They will upload the data to WebMAUS and accompanying services and save the results directly inside our existing `emuDB` (this of course takes some time, depending on the size of your database and the speed of your internet connection).

```
runBASwebservice_g2pForTokenization(handle = dbHandle,
                                    language = "eng-AU",
                                    transcriptionAttributeDefinitionName = "transcription",
                                    orthoAttributeDefinitionName = "Word")
```

This part splitted your transcription into word tokens and added something like

```
{
```

```
      "items": [
        {
          "id": 2,
          "labels": [
            {
              "name": "Word",
              "value": "amongst"
            }
          ]
        },
        {
          "id": 3,
          "labels": [
            {
              "name": "Word",
              "value": "her"
            }
...
```

to your `annot.json`.

The following will add canonical phonological forms to your `annot.json`;

```
runBASwebservice_g2pForPronunciation(handle = dbHandle,
                                     language = "eng-AU",
                                     orthoAttributeDefinitionName = "Word",
                                     canoAttributeDefinitionName = "Canonical")
```

e.g.

```
    {
      "items": [
        {
          "id": 2,
          "labels": [
            {
              "name": "Word",
              "value": "amongst"
            },
            {
              "name": "Canonical",
              "value": "@ m V N s t"
            }
          ]
        },
```

We can inspect these with

```
summary(dbHandle)
serve(dbHandle)
```

Until now, we have created additional `ITEM` only, i.e. other symbolic representations of our written transcription. We still do not have any relation to time and to our audio signal.

Therefore, we now use MAuS' forced-alignment algorithm to derive the most probable phonetical forms and to segment automatically the data:

```
runBASwebservice_maus(handle = dbHandle,
                      language = "eng-AU",
                      canoAttributeDefinitionName = "Canonical",
                      mausAttributeDefinitionName = "Phonetic")
```

When the 3 services are finished, we can use

```
serve(dbHandle)
```

to inspect the database with the new annotations.

All `annot.json` files now include lines like the following ones:

```
    {
      "items": [
        {
          "id": 9,
          "sampleStart": 0,
          "sampleDur": 3199,
          "labels": [
            {
              "name": "Phonetic",
              "value": "<p:>"
            }
          ]
        },
        {
          "id": 10,
          "sampleStart": 3200,
          "sampleDur": 1799,
          "labels": [
            {
              "name": "Phonetic",
              "value": "@"
            }
          ]
        },
...
      "name": "Phonetic",
      "type": "SEGMENT"
    }
  ],
```

Our database now includes words and phonemes as timeless `ITEM`s as well as phonetic `SEGMENT`s containing symbols for the most probable phonetic realization and, of course, start and end times (calculated from the information in `sampleSTART` and `sampleDur` in the `annot.json`-files).

```
summary(dbHandle)
# Name:  myEmuDatabase
# UUID:  4a282f0a-62ee-435a-aead-b503ff002b6d
# Directory:     /private/var/folders/vh/j2k1_0395x5_sgzpbl4bzzl00000gn/T/Rtmpj0j4RT/myEmuDatabase_emuD
# Session count: 1
# Bundle count: 7
# Annotation item count:  293
# Label count:  354
# Link count:  272
```

```
#
# Database configuration:
#
# SSFF track definitions:
# NULL
#
# Level definitions:
#       name    type nrOfAttrDefs           attrDefNames
# 1   bundle    ITEM            2 bundle; transcription;
# 2     Word    ITEM            2       Word; Canonical;
# 3 Phonetic SEGMENT            1              Phonetic;
#
# Link definitions:
#          type superlevelName sublevelName
# 1 ONE_TO_MANY         bundle         Word
# 2 ONE_TO_MANY           Word     Phonetic
```

This chapter described the current best practice of combining SpeechRecorder, MAUS and the EMU-SDMS to fit a typical phonetic project work flow.

However, be informed that we could have used `runBASwebservice_all()` to run all available BASwebservices at a time instead (be careful, this might take a while):

```
convert_txtCollection(dbName = "myEmuDatabaseBASall",
                      sourceDir = file.path(tempdir(), "emuR_demoData", "txt_collection"),
                      targetDir = tempdir())
dbHandleBASall = load_emuDB(file.path(tempdir(), "myEmuDatabaseBASall_emuDB"))


runBASwebservice_all(handle = dbHandleBASall,
                                    language = "eng-AU",
                                    transcriptionAttributeDefinitionName = "transcription")

serve(dbHandleBASall)
summary(dbHandleBASall)
# Name:  myEmuDatabase
# UUID:  7dd1a809-dcf4-49a4-9c55-a1b540dcea4d
# Directory:     /private/var/folders/vh/j2k1_0395x5_sgzpbl4bzzl00000gn/T/RtmpjOj4RT/myEmuDatabase_emuD
# Session count: 1
# Bundle count: 7
# Annotation item count:  576
# Label count:  691
# Link count:  353
#
# Database configuration:
#
# SSFF track definitions:
# NULL
#
# Level definitions:
#    name    type nrOfAttrDefs           attrDefNames
# 1 bundle    ITEM            2 bundle; transcription;
# 2    ORT    ITEM            3         ORT; KAN; KAS;
# 3    MAU SEGMENT            1                  MAU;
# 4  MINNI SEGMENT            1                MINNI;
```

```
# 5    MAS    ITEM            1                   MAS;
#
# Link definitions:
#         type superlevelName sublevelName
# 1 ONE_TO_MANY         bundle          ORT
# 2 ONE_TO_MANY            ORT          MAS
# 3 ONE_TO_MANY            MAS          MAU
```