The EMU-SDMS Manual

2017-10-09

Contents

Chapter 1

Prerequisites

1.1 Installing the EMU-SDMS

1.) R

- Download the R programming language from http://www.cran.r-project.org
- Install the R programming language by executing the downloaded file and following the on-screen instructions.

2.) emuR

- Start up R.
- Enter `install.packages("emuR")` after the `>` prompt to install the package. (You will only need to repeat this if package updates become available.)
- As the `wrassp` package is a dependency of the `emuR` package, it does not have to be installed separately.

3.) EMU-webApp (prerequisite)

- The only thing needed to use the EMU-webApp is a current HTML5 compatible browser (Chrome/Firefox/Safari/Opera/…). However, as most of the development and testing is done using Chrome we recommend using it, as it is by far the best tested browser.

1.2 Installing additional R-packages that are used throughout this document

- ggplot2 for visualization purposes: `install.packages("emuR")`
- dplyr for data manipulation: `install.packages("dplyr")`

1.3 Version disclaimer

This document describes the following versions of the software components:

- `wrassp`
  - Package version: 0.1.4
  - Git SHA1: `jsonlite::fromJSON("https://api.github.com/repos/IPS-LMU/wrassp/branches/master")$com`
- `emuR`
  - Package version: `packageVersion("emuR")`

  - Git SHA1: `jsonlite::fromJSON("https://api.github.com/repos/IPS-LMU/emuR/branches/master")$commi`
- `EMU-webApp`
  - Version: `jsonlite::fromJSON("https://raw.githubusercontent.com/IPS-LMU/EMU-webApp/master/package`
  - Git SHA1: `jsonlite::fromJSON("https://api.github.com/repos/IPS-LMU/EMU-webapp/branches/master")`

As the development of the EMU Speech Database Management System is still ongoing, be sure you have the correct documentation to go with the version you are using.

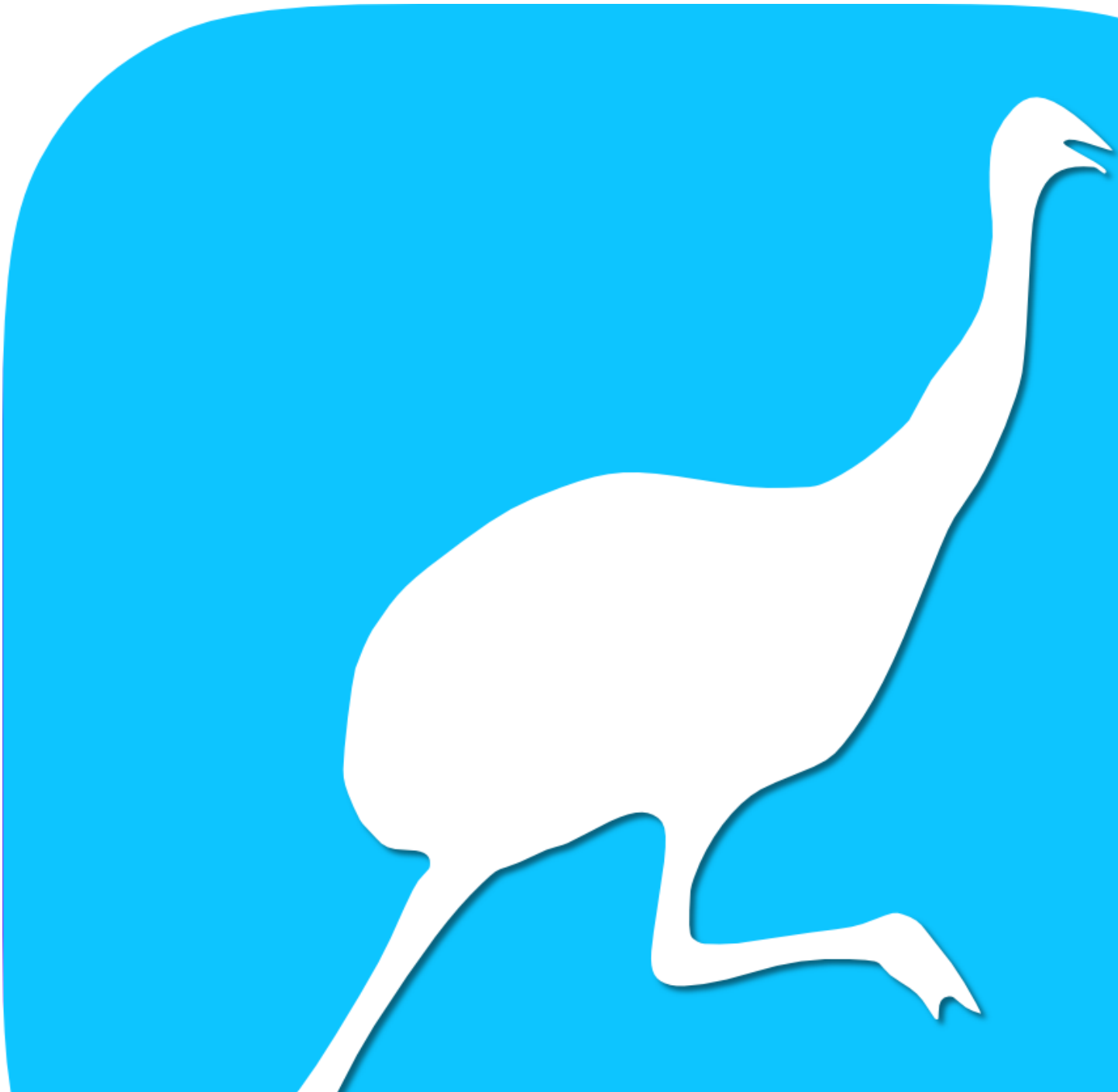
1.4 For developers and people interested in the source code

The information on how to install and/or access the source code of the developer version including the possibility of accessing the versions described in this document (via the Git SHA1 hashes mentioned above) is given below.

- `wrassp`
  - Source code is available here: https://github.com/IPS-LMU/wrassp/
  - Install developer version in R: `install.packages("devtools"); library("devtools"); install_github("IPS-LMU/wrassp")`
  - Bug reports: https://github.com/IPS-LMU/wrassp/issues
- `emuR`
  - Source code is available here: https://github.com/IPS-LMU/emuR/
  - Install developer version in R: `install.packages("devtools"); library("devtools"); install_github("IPS-LMU/emuR")`
  - Bug reports: https://github.com/IPS-LMU/emuR/issues
- `EMU-webApp`
  - Source code is available here: https://github.com/IPS-LMU/EMU-webApp/
  - Bug reports: https://github.com/IPS-LMU/EMU-webApp/issues

Chapter 2

An overview of the EMU-SDMS [1]



The EMU Speech Database Management System is a collection of software tools which aims to be as close to an all-in-one solution for generating, manipulating, querying, analyzing and managing speech databases as possible. It was developed to fill the void in the landscape of software tools for the speech sciences by providing an integrated system that is centered around the R language and environment for statistical computing and graphics (R Core Team, 2016). This manual contains the documentation for the three software components `wrassp`, `emuR` and the `EMU-webApp`. In addition, it provides an in-depth description of the `emuDB` database format which is also considered an integral part of the new system. These four components comprise the EMU-SDMS and benefit the speech sciences and spoken language research by providing an integrated system to answer research questions such as: *Given an annotated speech database, is vowel height (measured by its correlate, the first formant frequency) influenced by whether it appears in a strong or weak syllable?*

This manual is targeted at new EMU-SDMS users as well as users familiar with the legacy EMU system. In addition, it is aimed at people who are interested in the technical details such as data structures/formats and implementation strategies, be it for reimplementation purposes or simply for a better understanding of the inner workings of the new system. To accommodate these different target groups, after initially giving an overview of the system, this manual presents a usage tutorial that walks the user through the entire process of answering a research question. This tutorial will start with a set of `.wav` audio and Praat `.TextGrid` (Boersma and Weenink, 2016) annotation files and end with a statistical analysis to address the hypothesis posed by the research question. The following Part **??** of this documentation is separated into six chapters that give an in-depth explanation of the various components that comprise the EMU-SDMS and integral concepts of the new system. These chapters provide a tutorial-like overview by providing multiple examples. To give the reader a synopsis of the main functions and central objects that are provided by EMU-SDMS's main R package `emuR`, an overview of these functions is presented in Part **??**. Part **??** focuses on the actual implementation of the components and is geared towards people interested in the technical details. Further examples and file format descriptions are available in various appendices. This structure enables the novice EMU-SDMS user to simply skip the technical details and still get an in-depth overview of how to work with the new system and discover what it is capable of.

A prerequisite that is presumed throughout this document is the reader's familiarity with basic terminology in the speech sciences (e.g., familiarity with the international phonetic alphabet (IPA) and how speech is annotated at a coarse and fine grained level). Further, we assume the reader has a grasp of the basic concepts of the R language and environment for statistical computing and graphics. For readers new to R, there are multiple, freely available R tutorials online (e.g., https://en.wikibooks.org/wiki/Statistical_Analysis:_an_Introduction_using_R/R_basics. R also has a set of very detailed manuals and tutorials that come preinstalled with R. To be able to access R's own "An Introduction to R" introduction, simply type `help.start()` into the R console and click on the link to the tutorial.

2.1 The evolution of the EMU-SDMS

The EMU-SDMS has a number of predecessors that have been continuously developed over a number of years (e.g., Harrington et al., 1993, Cassidy and Harrington (1996), Cassidy and Harrington (2001), Bombien et al. (2006), Harrington (2010), John (2012)). The components presented here are the completely rewritten and newly designed, next incarnation of the EMU system, which we will refer to as the EMU Speech Database Management System (EMU-SDMS). The EMU-SDMS keeps most of the core concepts of the previous system, which we will refer to as the legacy system, in place while improving on things like usability, maintainability, scalability, stability, speed and more. We feel the redesign and reimplementation elevates the system into a modern set of speech and language tools that enables a workflow adapted to the challenges confronting speech scientists and the ever growing size of speech databases. The redesign has enabled us to implement several components of the new EMU-SDMS so that they can be used independently of the EMU-SDMS for tasks such as web-based collaborative annotation efforts and performing speech signal processing in a statistical programming environment. Nevertheless, the main goal of the redesign and reimplementation was to provide a modern set of tools that reduces the complexity of the tool chain needed to answer spoken language research questions down to a few interoperable tools. The tools the EMU-SDMS provides are

designed to streamline the process of obtaining usable data, all from within an environment that can also be used to analyze, visualize and statistically evaluate the data.

Upon developing the new system, rather than starting completely from scratch it seemed more appropriate to partially reuse the concepts of the legacy system in order to achieve our goals. A major observation at the time was that the R language and environment for statistical computing and graphics (R Core Team, 2016) was gaining more and more traction for statistical and data visualization purposes in the speech and spoken language research community. However, R was mostly only used towards the end of the data analysis chain where data usually was pre-converted into a comma-separated values or equivalent file format by the user using other tools to calculate, extract and pre-process the data. While designing the new EMU-SDMS, we brought R to the front of the tool chain to the point just beyond data acquisition. This allows the entire data annotation, data extraction and analysis process to be completed in R, while keeping the key user requirements in mind. Due to the personal experiences gained by using the legacy system in various undergraduate courses (course material usually based on Harrington, 2010), we learned that the key user requirements were data and database portability, a simple installation process, an as pleasant as possible user experience and cross-platform availability. Supplying all of EMU-SDMS's core functionality in the form of R packages that do not rely on external software at runtime seemed to meet all of these requirements.

As the early incarnations of the legacy EMU system and its predecessors were conceived either at a time that predated the R system or during the infancy of R's package ecosystem, the legacy system was implemented as a modular yet composite standalone program with a communication and data exchange interface to the R/Splus systems (see Cassidy and Harrington, 2001, Section 3 for details). Recent developments in the package ecosystem of R such as the availability of the `DBI` package (R Special Interest Group on Databases (R-SIG-DB) et al., 2016) and the related packages `RSQLite` and `RPostgreSQL` (Wickham et al., 2014, Conway et al. (2016)), as well as the `jsonlite` package (Ooms, 2014) and the `httpuv` package (RStudio and Inc., 2015), have made R an attractive sole target platform for the EMU-SDMS. These and other packages provide additional functional power that enabled the EMU-SDMS's core functionality to be implemented in the form of R packages. The availability of certain R packages had a large impact on the architectural design decisions that we made for the new system.

R Example @ref(rexample:overview_install) shows the simple installation process which we were able to achieve due to the R package infrastructure. Compared to the legacy EMU and other systems, the installation process of the entire system has been reduced to a single R command. Throughout this documentation we will try to highlight how the EMU-SDMS is also able to meet the rest of the above key user requirements.

```
# install the entire EMU-SDMS
# by installing the emuR package
install.packages("emuR")
```

It is worth noting that throughout this manual R Example code snippets will be given in the form of R Example @ref(rexample:overview_install). These examples represent working R code that allow the reader to follow along in a hands-on manor and give a feel for what it is like working with the new EMU-SDMS.

2.2 EMU-SDMS: System architecture and default workflow

As was previously mentioned, the new EMU-SDMS is made up of four main components. The components are the `emuDB` format; the R packages `wrassp` and `emuR`; and the web application, the `EMU-webApp`, which is EMU-SDMS's new graphical user interface (GUI) component. An overview of the EMU-SDMS's architecture and the components' relationships within the system is shown in Figure @ref(fig:overview_archOver). In Figure @ref(fig:overview_archOver), the `emuR` package plays a central role as it is the only component that interacts with all of the other components of the EMU-SDMS. It performs file and DB handling for the files that comprise an `emuDB` (see Chapter @ref(chap:annot_struct_mod)); it uses the `wrassp` package for signal processing purposes (see Chapter **??**); and it can serve `emuDB`s to the `EMU-webApp` (see Chapter **??**).

Although the system is made of four main components, the user largely only interacts directly with the

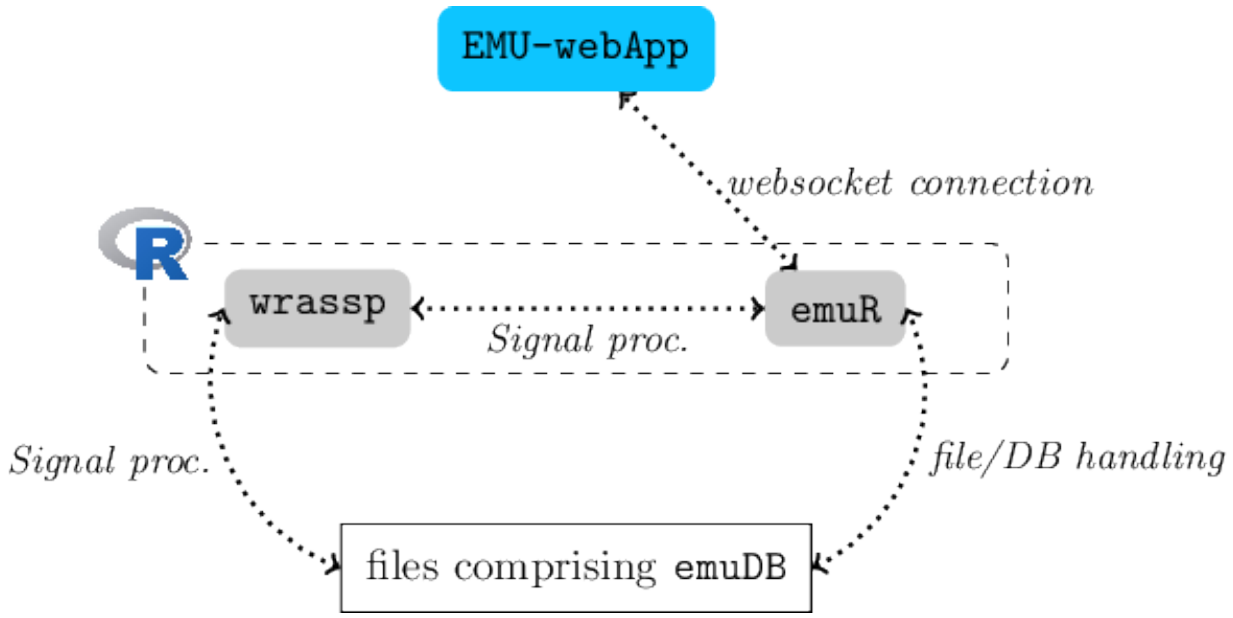


Figure 2.1: Schematic architecture of the EMU-SDMS.(#fig:overview_archOver)

`EMU-webApp` and the `emuR` package. A summary of the default workflow illustrating theses interactions can be seen below:

1. Load database into current R session (`load_emuDB()`).
2. Database annotation / visual inspection (`serve()`). This opens up the `EMU-webApp` in the system's default browser.
3. Query database (`query()`). This is optionally followed by `requery_hier()` or `requery_seq()` as necessary (see Chapter **??** for details).
4. Get trackdata (e.g. formant values) for the result of a query (`get_trackdata()`).
5. Prepare data.
6. Visually inspect data.
7. Carry out further analysis and statistical processing.

Initially the user creates a reference to an `emuDB` by loading it into their current R session using the `load_emuDB()` function (see step 1). This database reference can then be used to either serve (`serve()`) the database to the `EMU-webApp` or query (`query()`) the annotations of the `emuDB` (see steps 2 and 3). The result of a query can then be used to either perform one or more so-called requeries or extract signal values that correspond to the result of a `query()` or `requery()` (see step 4). Finally, the signal data can undergo further preparation (e.g., correction of outliers) and visual inspection before further analysis and statistical processing is carried out (see steps 5, 6 and 7). Although the R packages provided by the EMU-SDMS do provide functions for steps 4, 5 and 6, it is worth noting that the plethora of R packages that the R package ecosystem provides can and should be used to perform these duties. The resulting objects of most of the above functions are derived `matrix` or `data.frame` objects which can be used as inputs for hundreds if not thousands of other R functions.

2.3 EMU-SDMS: Is it something for you?

Besides providing a fully integrated system, the EMU-SDMS has several unique features that set it apart from other current, widely used systems (e.g., Boersma and Weenink, 2016, Wittenburg et al. (2006), Fromont and Hay (2012), Rose et al. (2006), McAuliffe and Sonderegger (2016)). To our knowledge, the EMU-SDMS is the only system that allows the user to model their annotation structures based on a hybrid

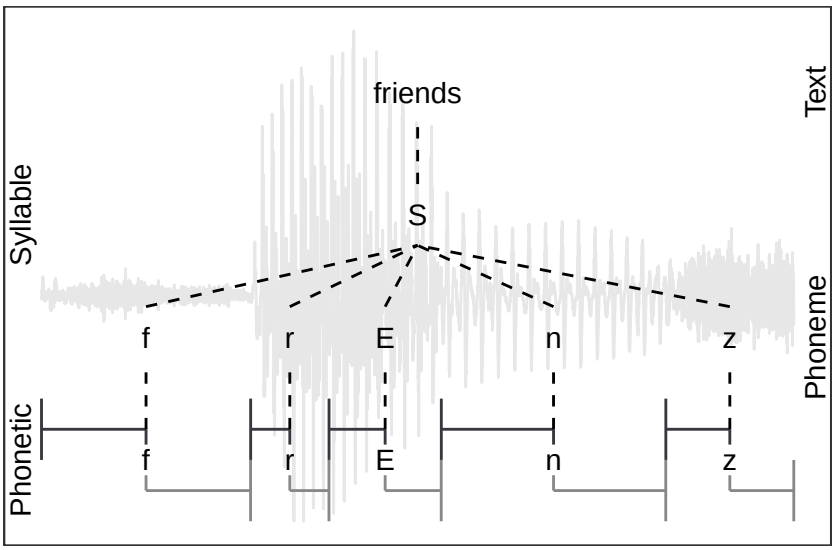


Figure 2.2: (#fig:overview_hybridAnnot)Example of a hybrid annotation combining time-based (*Phonetic* level) and hierarchical (*Phoneme*, *Syllable*, *Text* levels including the inter-level links) annotations.

model of time-based annotations (such as those offered by Praat's tier-based annotation mechanics) and hierarchical timeless annotations. An example of such a hybrid annotation structure is displayed in Figure @ref(overview_hybridAnnot). These hybrid annotations benefit the user in multiple ways, as they reduce data redundancy and explicitly allow relationships to be expressed across annotation levels (see Chapter @ref(chap:annot_struct_mod) for further information on hierarchical annotations and Chapter **??** on how to query these annotation structures).

Further, to our knowledge, the EMU-SDMS is the first system that makes use of a web application as its primary GUI for annotating speech. This unique approach enables the GUI component to be used in multiple ways. It can be used as a stand-alone annotation tool, connected to a loaded `emuDB` via `emuR`'s `serve()` function and used to communicate to other servers. This enables it to be used as a collaborative annotation tool. An in-depth explanation of how this component can be used in these three scenarios is given in Chapter **??**.

As demonstrated in the default workflow of Section @ref(sec:overview_sysArch), an additional unique feature provided by EMU-SDMS is the ability use the result of a query to extract derived (e.g., formants and RMS values) and complementary signals (e.g., electromagnetic articulography data) that match the segments of a query. This, for example, aids the user in answering questions related to derived speech signals such as: *Is vowel height (measured by its correlate, the first formant frequency) influenced by whether it appears in a strong or weak syllable?*. Chapter 3 gives a complete walk-through of how to go about answering this question using the tools provided by the EMU-SDMS.

The features provided by the EMU-SDMS make it an all-in-one speech database management solution that is centered around R. It enriches the R platform by providing specialized speech signal processing, speech database management, data extraction and speech annotation capabilities. By achieving this without relying on any external software sources except the web browser, the EMU-SDMS significantly reduces the number of tools the speech and spoken language researcher has to deal with and helps to simplify answering research questions. As the only prerequisite for using the EMU-SDMS is a basic familiarity with the R platform, if the above features would improve your workflow, the EMU-SDMS is indeed for you.

Chapter 3

A tutorial on how to use the EMU-SDMS [1]

–> –> –> –> –>
–> –> –> –> –> –>
–> –> –> –> –> –>
–> –> –> –> –> –>
–> –> –> –> –> –>

[1] Sections of this chapter have been published in Winkelmann et al. (2017)

Chapter 4

The query system

Chapter 5

Toolchain SpeechRecorder — MAUS — EMU-SDMS

Most phonetic research projects involve this workflow:

1. Record speech
2. Annotate speech using automatic tools
3. Check and correct the generated annotations by hand
4. Analyze speech (connecting the primary data with annotations and derived signals)

The EMU Speech Database Management System is focused on steps 3 and 4 of this workflow. For the first two steps, it can very usefully be complemented by two other tools: SpeechRecorder and MAUS (or, more broadly speaking — and more correctly, for that matter — the BAS Web Services). This chapter introduces how the tools can be combined — in a systematic way, with as little fuss as possible.

5.1 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]

5.2 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.

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

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]

5.3 Combining the tools

The order of the processing steps is unchangeable: The files have to be recorded first, then annotated and then analyzed.

However, there are different ways of passing data around between the tools. For example SpeechRecorder might *export* files into emuDB format or the EMU-SDMS might *import* files stored in SpeechRecorder's format. After all, they are separate tools and can be used individually (although the combination makes great sense).

Moreover, sometimes a database grows, and some new files are recorded while others have already been annotated or analyzed (which challenges the "unchangeable order of processing steps").

We can see now that we have to convert our data between different formats. We want to benefit from all tools but keep the conversion work down to a minimum. In this section, we explain the (currently) best way to do this. 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.

5.3.1 Importing recordings and transcriptions into Emu

5.3.1.1 Using the import_speechRecorder() function

The `import_speechRecorder()` function is in the making, but unfortunately not finished yet. We therefore have to resort to the second-best way of importing SpeechRecorder's results into Emu:

5.3.1.2 Using intermediary text files

SpeechRecorder, per default, saves one .wav file for each prompt. With additional settings, it will save a .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*. Note that parts of SpeechRecorder's user interface are in German:

- Under "Projekt / Einstellungen… / Annotation":
  - Under "Persist", tick the checkbox that says "Simple text loader writer for annotation template for MAUS processing"
  - Under "Auto annotation", tick the checkbox 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 checkbox "Use as annotation template" for *every recording*.

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

Now, after your recording session, you will have audio and text files. In emuR, this combination is called a *txt collection*. We will thus use the function `convert_txtCollection`, to import SpeechRecorder's files into Emu's 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())

# 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())

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

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

```
summary(dbHandle)
```

or

```
serve(dbHandle)
```

The database contains all our recordings, and exactly one type of annotation: orthographic transcription. No phonetic transcription, no segmentation. The next section will cover that.

5.3.2 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 your 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-US",
                                    transcriptionAttributeDefinitionName = "transcription",
                                    orthoAttributeDefinitionName = "Word")

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

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

When the services are finished, we can use

```
serve(db)
```

to inspect the database with the new annotations. This time, it includes segmentation into words and phonemes, canonical phonetic transcription and realized phonetic transcription.

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

Bibliography

Boersma, P. and Weenink, D. (2016). Praat: doing phonetics by computer (Version 6.0.19). http://www.fon.hum.uva.nl/praat/.

Bombien, L., Cassidy, S., Harrington, J., John, T., and Palethorpe, S. (2006). Recent developments in the Emu speech database system. In *Proc. 11th SST Conference Auckland*, pages 313–316.

Cassidy, S. and Harrington, J. (1996). Emu: An enhanced hierarchical speech data management system. In *Proceedings of the Sixth Australian International Conference on Speech Science and Technology*, pages 361–366.

Cassidy, S. and Harrington, J. (2001). Multi-level annotation in the Emu speech database management system. *Speech Communication*, 33(1):61–77.

Conway, J., Eddelbuettel, D., Nishiyama, T., Prayaga, S. K., and Tiffin, N. (2016). *RPostgreSQL: R interface to the PostgreSQL database system*. R package version 0.4-1 package version 0.4-1.

Fromont, R. and Hay, J. (2012). LaBB-CAT: An annotation store. In *Australasian Language Technology Association Workshop 2012*, volume 113. Citeseer.

Harrington, J. (2010). *Phonetic analysis of speech corpora*. John Wiley & Sons.

Harrington, J., Cassidy, S., Fletcher, J., and Mc Veigh, A. (1993). The mu+ system for corpus based speech research. *Computer Speech & Language*, 7(4):305–331.

John, T. (2012). *Emu speech database system*. PhD thesis, Ludwig Maximilian University of Munich.

McAuliffe, M. and Sonderegger, M. (2016). Speech Corpus Tools (SCT). http://speech-corpus-tools.readthedocs.io/.

Ooms, J. (2014). The jsonlite package: A practical and consistent mapping between json data and r objects. *arXiv:1403.2805 [stat.CO]*.

R Core Team (2016). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

R Special Interest Group on Databases (R-SIG-DB), Wickham, H., and Müller, K. (2016). *DBI: R Database Interface*. R package version 0.4.

Rose, Y., MacWhinney, B., Byrne, R., Hedlund, G., Maddocks, K., O'Brien, P., and Wareham, T. (2006). Introducing phon: A software solution for the study of phonological acquisition. In *Proceedings of the... Annual Boston University Conference on Language Development. Boston University Conference on Language Development*, volume 2006, page 489. NIH Public Access.

RStudio and Inc. (2015). *httpuv: HTTP and WebSocket Server Library*. R package version 1.3.3.

Wickham, H., James, D. A., and Falcon, S. (2014). *RSQLite: SQLite Interface for R*. R package version 1.0.0.

Winkelmann, R., Harrington, J., and Jänsch, K. (2017). EMU-SDMS: Advanced speech database management and analysis in R. *Computer Speech & Language*, pages –.

Wittenburg, P., Brugman, H., Russel, A., Klassmann, A., and Sloetjes, H. (2006). Elan: a professional framework for multimodality research. In *Proceedings of LREC*, volume 2006.