The EMU-SDMS Manual

2017-10-09

Contents

Chapter 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.

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

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

1.2 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.

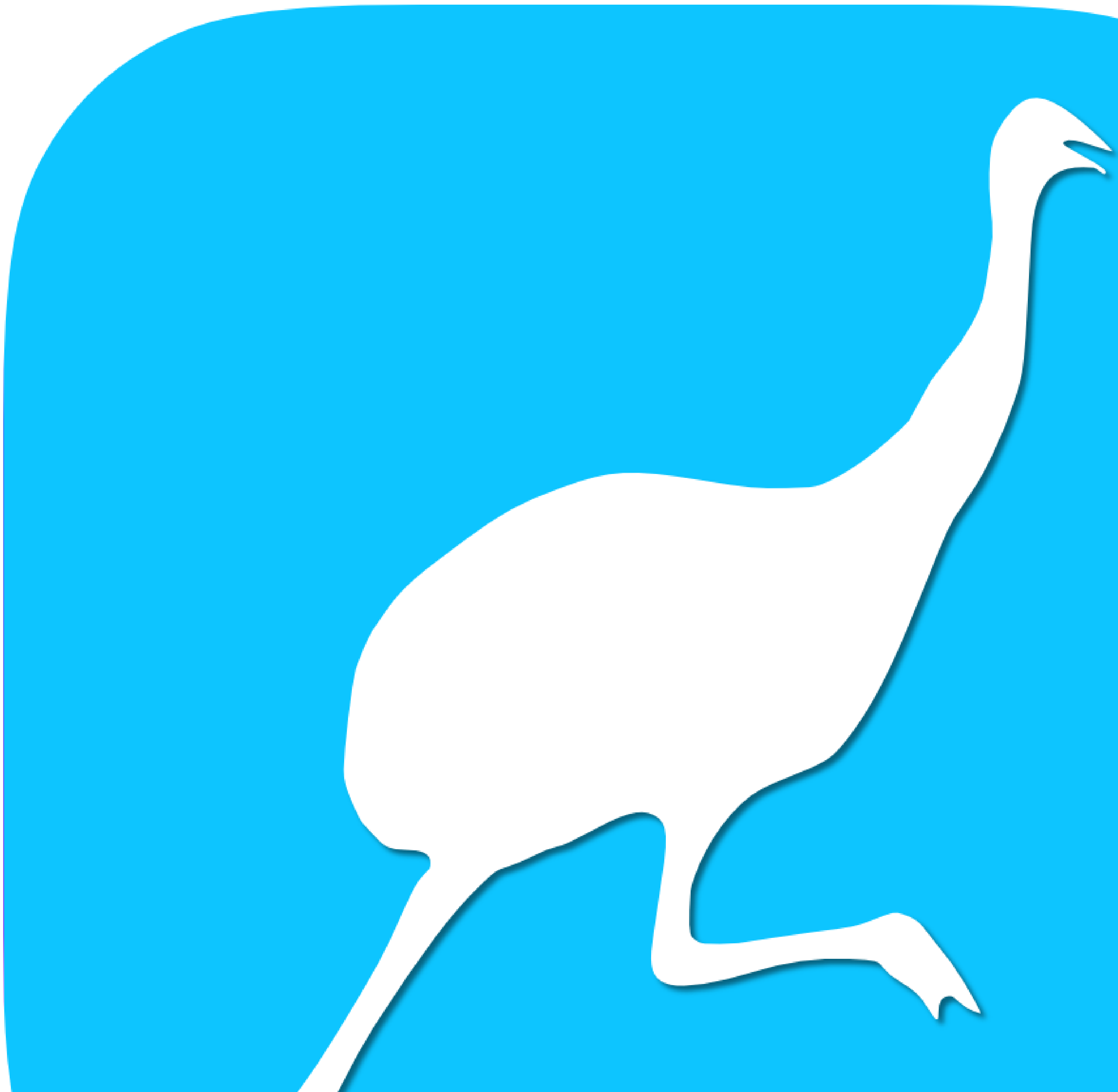
1.3 For developers and people interested in the source code

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.

```r
# 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

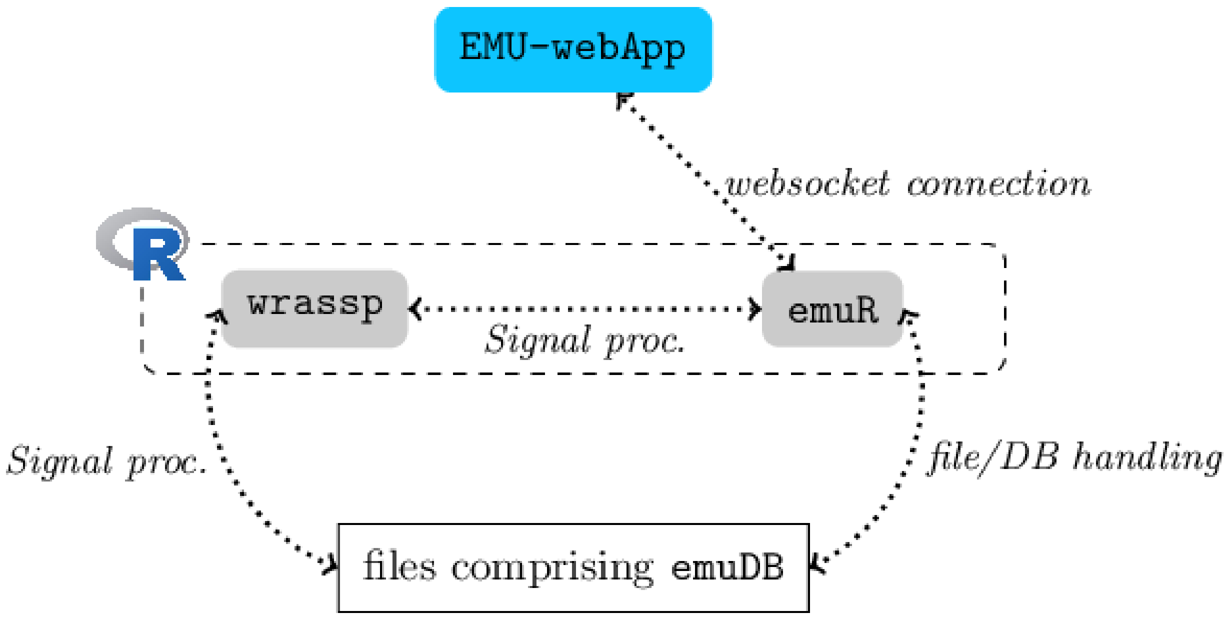


Figure 2.1: Schematic architecture of the EMU-SDMS.(#fig:overview_archOver)

`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

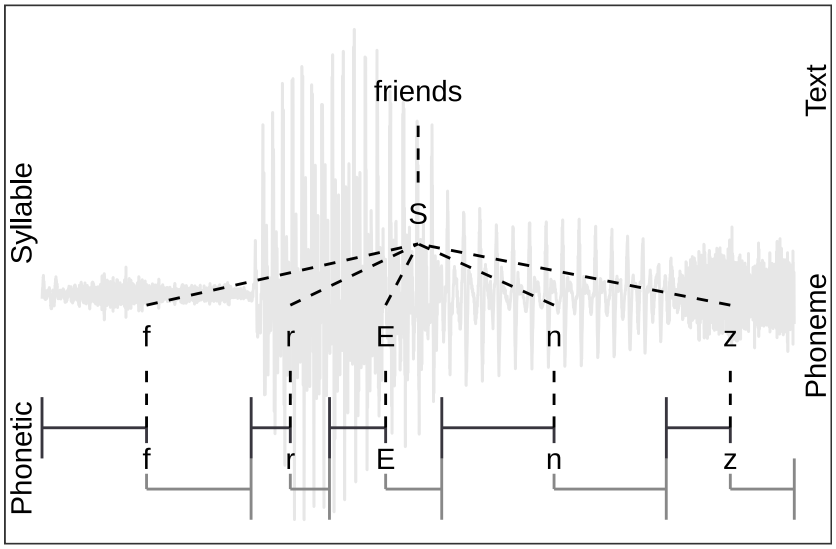


Figure 2.2: (#fig:overview_hybridAnnot)Example of a hybrid annotation combining time-based (*Phonetic* level) and hierarchical (*Phoneme*, *Syllable*, *Text* levels including the inter-level links) annotations.

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]

Using the tools provided by the EMU-SDMS, this tutorial chapter gives a practical step-by-step guide to answering the question: *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?* The tutorial only skims over many of the concepts and functions provided by the EMU-SDMS. In-depth explanations of the various functionalities are given in later chapters of this documentation.

As the EMU-SDMS is not concerned with the raw data acquisition, other tools such as SpeechRecorder by Draxler and Jänsch (2004) are first used to record speech. However, once audio speech recordings are available, the system provides multiple conversion routines for converting existing collections of files to the new `emuDB` format described in Chapter **??** and importing them into the new EMU system. The current import routines provided by the `emuDB` package are:

- `convert_TextGridCollection()` - Convert TextGrid collections (`.wav` and `.TextGrid` files) to the `emuDB` format,
- `convert_BPFCollection()` - Convert BPF collections (`.wav` and `.par`} files) to the `emuDB` format,
- `convert_txtCollection()` - Convert plain text file collections format (`.wav` and `.txt` files) to the `emuDB` format,
- `convert_legacyEmuDB()` - Convert the legacy EMU database format to the `emuDB` format and
- `create_emuDB()` followed by `add_link/levelDefinition` and `import_mediaFiles()` - Creating `emuDB`s from scratch with only audio files present.

The `emuDB` package comes with a set of example files and small databases that are used throughout the `emuDB` documentation, including the functions help pages. These can be accessed by typing `help(functionName)` or the short form `?functionName`. R Example @ref(rexample:tutorial_create_emuRdemoData) illustrates how to create this demo data in a user-specified directory. Throughout the examples of this documentation the directory that is provided by the base R function `tempdir()` will be used, as this is available on every platform supported by R (see `?tempdir` for further details). As can be inferred from the `list.dirs()` output in R Example @ref(rexample:tutorial_create_emuRdemoData), the `emuR_demoData` directory contains a separate directory containing example data for each of the import routines. Additionally, it contains a directory containing an `emuDB` called *ae* (the directories name is `ae_emuDB`, where `_emuDB` is the default suffix given to directories containing a `emuDB`; see Chapter **??**).

```
# load the package
library(emuR)

# create demo data in directory provided by the tempdir() function
```

[1] Some examples of this chapter are adapted versions of examples of the `emuR_intro` vignette.

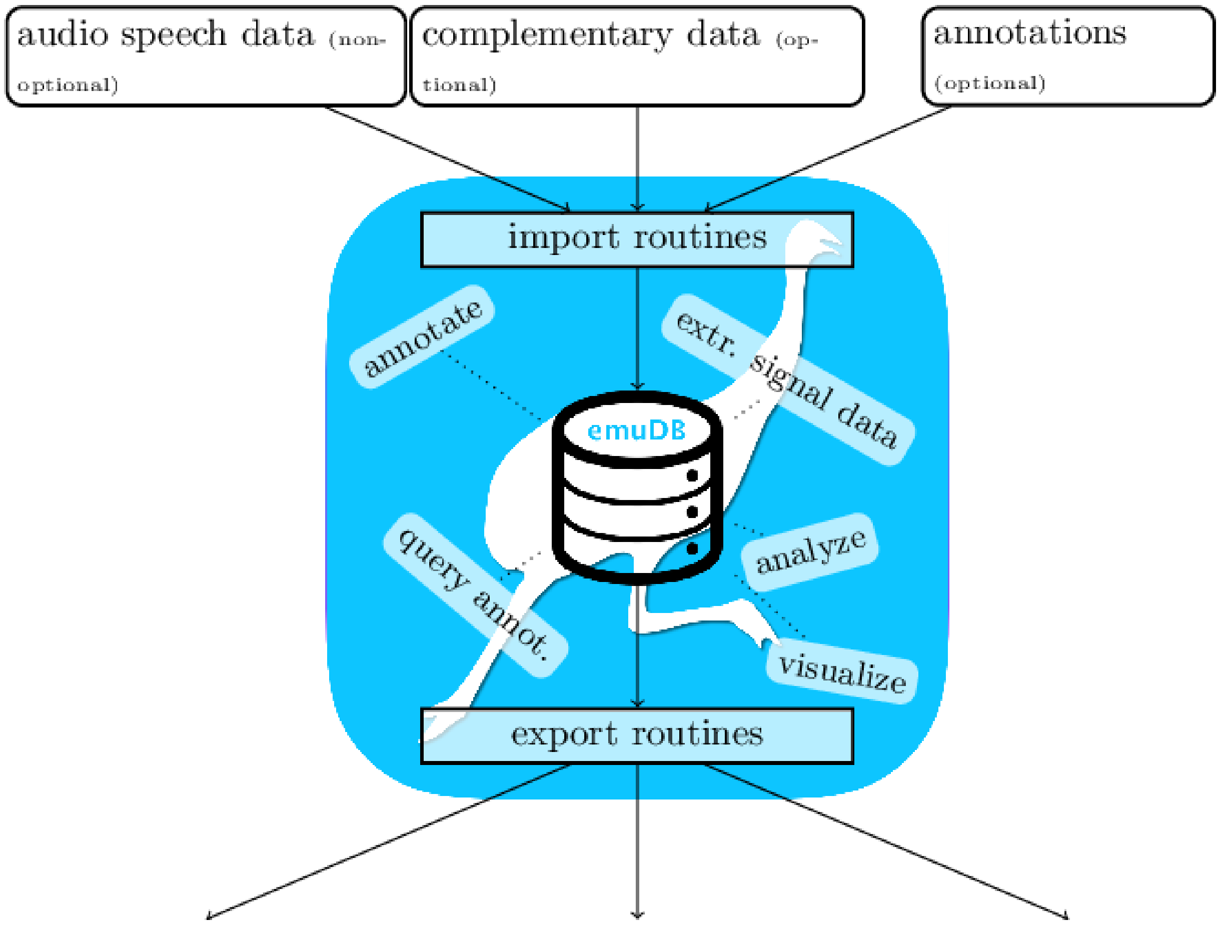

audio speech data (non-optional)
complementary data (optional)
annotations (optional)
import routines
annotate
extr. signal data
emuDB
query annot.
analyze
visualize
export routines

Figure 3.1:

```r
# (of course other directory paths may be chosen)
create_emuRdemoData(dir = tempdir())

# create path to demo data directory, which is
# called "emuR_demoData"
demoDataDir = file.path(tempdir(), "emuR_demoData")

# show demo data directories
list.dirs(demoDataDir, recursive = F, full.names = F)
```

```
## [1] "ae_emuDB"            "BPF_collection"      "legacy_ae"
## [4] "TextGrid_collection" "txt_collection"
```

This tutorial will start by converting a TextGrid collection containing seven annotated single-sentence utterances of a single male speaker to the `emuDB` format[2]. In the EMU-SDMS, a file collection such as a TextGrid collection refers to a set of file pairs where two types of files with different file extentions are present (e.g., `.ext1` and `.ext2`). It is vital that file pairs have the same basenames (e.g., `A.ext1` and `A.ext2` where `A` represents the basename) in order for the conversion functions to be able to pair up files that belong together. As other speech software tools also encourage such file pairs (e.g., Kisler et al., 2015) this is a common collection format in the speech sciences. R Example **??** shows such a file collection that is part of `emuDB`'s demo data. Figure @ref(fig:msajc003_praatTG) shows the content of an annotation as displayed by Praat's `"Draw visible sound and Textgrid..."` procedure.

```r
# create path to TextGrid collection
tgColDir = file.path(demoDataDir, "TextGrid_collection")

# show content of TextGrid_collection directory
list.files(tgColDir)
```

```
##  [1] "msajc003.TextGrid" "msajc003.wav"      "msajc010.TextGrid"
##  [4] "msajc010.wav"      "msajc012.TextGrid" "msajc012.wav"
##  [7] "msajc015.TextGrid" "msajc015.wav"      "msajc022.TextGrid"
## [10] "msajc022.wav"      "msajc023.TextGrid" "msajc023.wav"
## [13] "msajc057.TextGrid" "msajc057.wav"
```

3.1 Converting the TextGrid collection

The `convert_TextGridCollection()` function converts a TextGrid collection to the `emuDB` format. A precondition that all `.TextGrid` files have to fulfill is that they must all contain the same tiers. If this is not the case, yet there is an equal tier subset that is contained in all the TextGrid files, this equal subset may be chosen. For example, if all `.TextGrid` files contain only the tier `Phonetic: IntervalTier` the conversion will work. However, if a single `.TextGrid` of the collection has the additional tier `Tone: TextTier` the conversion will fail. In this case the conversion could be made to work by specifying the equal subset (e.g., `equalSubset = c("Phonetic")`) and passing it on to the `tierNames` function argument `convert_TextGridCollection(..., tierNames = equalSubset, ...)`. As can be seen in Figure @ref(fig:msajc003_praatTG), the TextGrid files provided by the demo data contain eleven tiers. To reduce the complexity of the annotations for this tutorial we will only convert the tiers *Text* (orthographic word annotations), *Syllable* (strong: *S* vs. weak: *W* syllable annotations), *Phoneme* (phoneme level annotations) and *Phonetic* (phonetic annotations) utilizing the `tierNames` parameter. This conversion can be seen in R Example @ref(rexample:tutorial_tgconv).

[2]The other input routines are covered in the Section @ref(sec:emuRpackageDetails_importRoutines).

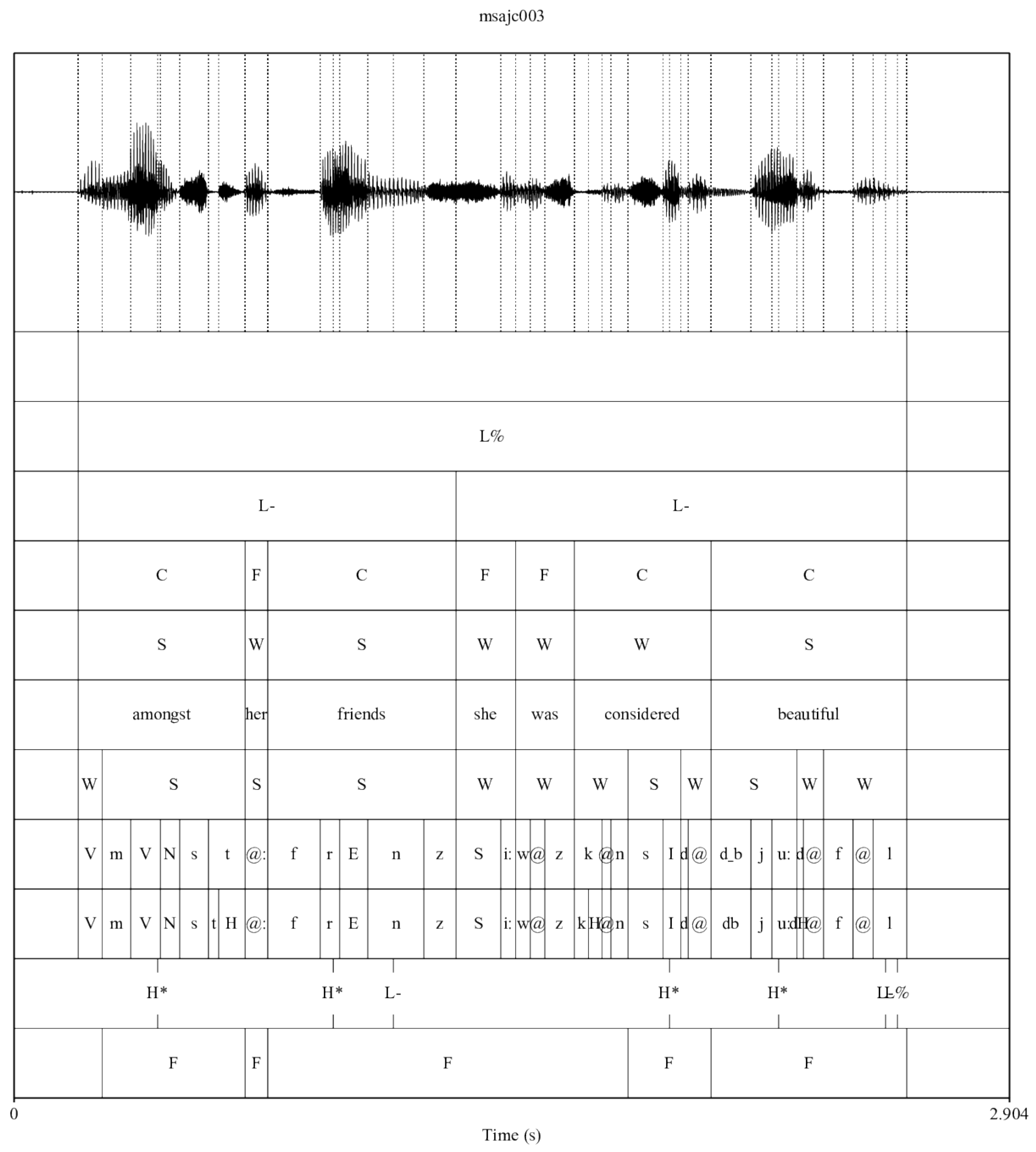

msajc003
L%
L-
L-
C F C F F C C
S W S W W W S
amongst her friends she was considered beautiful
W S S S W W W S W S W W
V m V N s t @: f r E n z S i: w @ z k @ n s I d @ d_b j u: d @ f @ l
V m V N s t H @: f r E n z S i: w @ z k H @ n s I d @ db j u: d H @ f @ l
H* H* L- H* H* L- L%
F F F F F
0
2.904
Time (s)

Figure 3.2: TextGrid annotation of the `emuR_demoData/TextGrid_collection/\allowbreak msajc003.wav / .TextGrid` file pair containing the tiers (from top to bottom): *Utterance*, *Intonational*, *Intermediate*, *Word*, *Accent*, *Text*, *Syllable*, *Phoneme*, *Phonetic*, *Tone*, *Foot.*(#fig:msajc003_praatTG)

```
# convert TextGrid collection to the emuDB format
convert_TextGridCollection(dir = tgColDir,
                           dbName = "myFirst",
                           targetDir = tempdir(),
                           tierNames = c("Text", "Syllable",
                                         "Phoneme", "Phonetic"))
```

The above call to `convert_TextGridCollection` creates a new `emuDB` directory in the `tempdir()` directory called `myFirst_emuDB`. This `emuDB` contains annotation files that contain the same *Text*, *Syllable*, *Phoneme* and *Phonetic* segment tiers as the original `.TextGrid` files as well as copies of the original (`.wav`) audio files. For further details about the structure of an `emuDB`, see Chapter **??** of this document.

3.2 Loading and inspecting the database

As mentioned in Section @ref(sec:overview_sysArch), the first step when working with an `emuDB` is to load it into the current R session. R Example @ref(rexample:tutorial_loadEmuDB) shows how to load the converted TextGrid collection into R using the `load_emuDB()` function.

```
# get path to emuDB called "myFirst"
# that was created by convert_TextGridCollection()
path2directory = file.path(tempdir(), "myFirst_emuDB")

# load emuDB into current R session
dbHandle = load_emuDB(path2directory, verbose = FALSE)
```

3.2.1 Overview

Now the *myFirst* `emuDB` is loaded into R, an overview of the current status and configuration of the database can be displayed using the `summary()` function as shown in R Example @ref(rexample:tutorial_summary).

```
# show summary
summary(dbHandle)
```

```
## Name:     myFirst
## UUID:     5952d681-2c6f-4491-a65e-21dc8f15a1f6
## Directory:    /private/var/folders/ll/pw5k9y6x64q5xqys350rz2ph0000gn/T/RtmptzvsXC/myFirst_emuDB
## Session count: 1
## Bundle count: 7
## Annotation item count:  664
## Label count:  664
## Link count:  0
##
## Database configuration:
##
## SSFF track definitions:
## NULL
##
## Level definitions:
##       name    type nrOfAttrDefs attrDefNames
## 1     Text SEGMENT            1        Text;
## 2 Syllable SEGMENT            1    Syllable;
## 3  Phoneme SEGMENT            1     Phoneme;
## 4 Phonetic SEGMENT            1    Phonetic;
##
## Link definitions:
## NULL
```

The extensive output of `summary()` is split into a top and bottom half, where the top half focuses on general information about the database (name, directory, annotation item count, etc.) and the bottom half displays information about the various Simple Signal File Format (SSFF) track, level and link definitions of the `emuDB`. The summary information about the level definitions shows, for instance, that the *myFirst* database has a *Text* level of type `SEGMENT` and therefore contains annotation items that have a start time and a segment duration. It is worth noting that information about the SSFF track, level and link definitions corresponds to the output of the `list_ssffTrackDefinitions()`, `list_levelDefinitions()` and `list_linkDefinitions()` functions.

3.2.2 Database annotation and visual inspection

The EMU-SDMS has a unique approach to annotating and visually inspecting data-bases, as it utilizes a web application called the `EMU-webApp` to act as its GUI. To be able to communicate with the web application the `emuDB` package provides a `serve()` function which is used in R Example @ref(rexample:tutorial_serve).

```r
# serve myFirst emuDB to the EMU-webApp
serve(dbHandle)
```

Executing this command will block the R console, automatically open up the system's default browser and display the following message in the R console:

```
## Navigate your browser to the EMU-webApp URL:
##  http://ips-lmu.github.io/EMU-webApp/ (should happen automatically)

## Server connection URL:
##  ws://localhost:17890

## To stop the server press the 'clear' button in the
## EMU-webApp or close/reload the webApp in your browser.
```

The `EMU-webApp`, which is now connected to the database via the `serve()` function, can be used to visually inspect and annotate the `emuDB`. Figure @ref(fig:tutorial_emuWebAppMyFirst) displays a screenshot of what the `EMU-webApp` looks like after automatically connecting to the server. As the `EMU-webApp` is a very feature-rich software annotation tool, this documentation has a whole chapter (see Chapter **??**) on how to use it, what it is capable of and how to configure it. Further, the web application provides its own documentation which can be accessed by clicking the EMU icon in the top right hand corner of the application's top menu bar. To close the connection and free up the blocked R console, simply click the `clear` button in the top menu bar of the `EMU-webApp`.

3.3 Querying and autobuilding the annotation structure

An integral step in the default workflow of the EMU-SDMS is querying the annotations of a database. The `emuDB` package implements a `query()` function to accomplish this task. This function evaluates an EMU Query Language (EQL) expression and extracts the annotation items from the database that match a query expression. As Chapter **??** gives a detailed description of the query mechanics provided by `emuDB`, this tutorial will only use a very small, hopefully easy to understand subset of the EQL.

The output of the `summary()` command in R Example @ref(rexample:tutorial_summary) and the screenshot in Figure @ref(fig:tutorial_emuWebAppMyFirst) show that the *myFirst* `emuDB` contains four levels of annotations. R Example @ref(rexample:tutorial_simpleQuery) shows four separate queries that query various

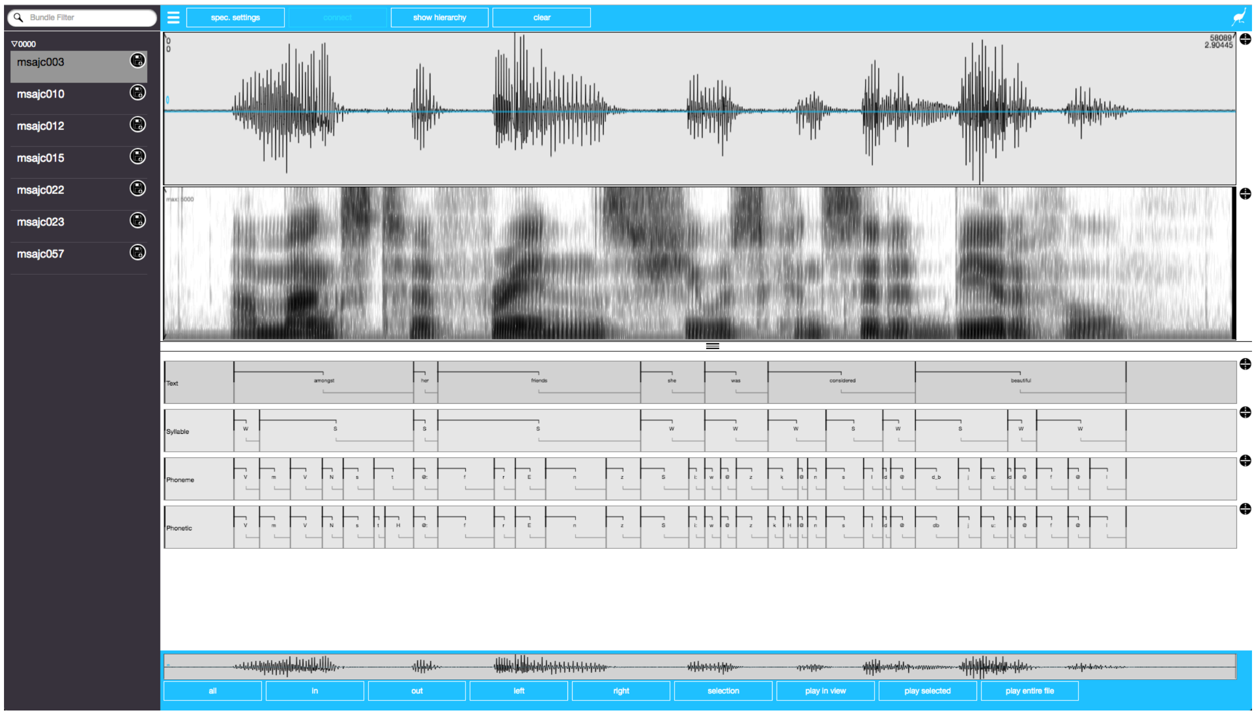


Figure 3.3: Screenshot of `EMU-webApp` displaying `msajc003` bundle of *myFirst* `emuDB`.(#fig:tutorial_emuWebAppMyFirst)

segments on each of the available levels. The query expressions all use the matching operator `==` which returns annotation items whose labels match those specified to the right of the operator and that belong to the level specified to the left of the operator (i.e., `LEVEL == LABEL`; see Chapter ?? for a detailed description).

```
# query all segments containing the label
# "was" of the "Phonetic" level
sl_text = query(emuDBhandle = dbHandle,
                query = "Text == was")

# query all segments containing the label
# "S" (==strong syllable) of the "Syllable" level
sl_syl = query(emuDBhandle = dbHandle,
               query = "Syllable == S")

# query all segments containing the label
# "n" on the "Phoneme" level
sl_phoneme = query(dbHandle,
                   query = "Phoneme == f")

# query all segments containing the label
# "n" of the "Phonetic" level
sl_phonetic = query(dbHandle,
                    query = "Phonetic == n")

# show class vector of query result
class(sl_phonetic)
```

```
## [1] "emuRsegs"   "emusegs"    "data.frame"
```

```
# show first entry of sl
head(sl_phonetic, n = 1)
```

```
## segment  list from database:  myFirst
## query was:  Phonetic == n
##   labels    start      end session   bundle    level    type
## 1      n 1031.925 1195.925    0000 msajc003 Phonetic SEGMENT
```

```
# show summary of sl
summary(sl_phonetic)
```

```
## segment  list from database:  myFirst
## query was:  Phonetic == n
##  with 12 segments
##
## Segment distribution:
##
##  n
## 12
```

As demonstrated in R Example @ref(rexample:tutorial_simpleQuery), the result of a query is an `emuRsegs` object, which is a super-class of the common `data.frame`. This object is often referred to as a segment list, or "seglist". A segment list carries information about the extracted annotation items such as the extracted labels, the start and end times of the segments, the sessions and bundles the items are from and the levels they belong to. An in-depth description of the information contained in a segment list is given in Section @ref(sec:query_emuRsegs). R Example @ref(rexample:tutorial_simpleQuery) shows that the `summary()` function can also be applied to a segment list object to get an overview of what is contained within it. This can be especially useful when dealing with larger segment lists.

3.3.1 Autobuilding

The simple queries illustrated above query segments from a single level that match a certain label. However, the EMU-SDMS offers a mechanism for performing inter-level queries such as: *Query all Phonetic items that contain the label "n" and are part of a strong syllable*. For such queries to be possible, the EMU-SDMS offers very sophisticated annotation structure modeling capabilities, which are described in Chapter @ref(chap:annot_struct_mod). For the sake of this tutorial we will focus on converting the flat segment level annotation structure displayed in Figure @ref(fig:tutorial_emuWebAppMyFirst) to a hierarchical form as displayed in Figure @ref(fig:tutorial_violentlyHier), where only the *Phonetic* level carries time information and the annotation items on the other levels are explicitly linked to each other to form a hierarchical annotation structure.

As it is a very laborious task to manually link annotation items together using the `EMU-webApp` and the hierarchical information is already implicitly contained in the time information of the segments and events of each level, we will now use a simple function provided by the `emuDB` package to build these hierarchical structures using this information called `autobuild_linkFromTimes()`. R Example @ref(rexample:tutorial_autobuild) shows the calls to this function which autobuild the hierarchical annotations in the *myFirst* database. As a general rule for autobuilding hierarchical annotation structures, a good strategy is to start the autobuilding process beginning with coarser grained annotation levels (i.e., the *Text*/*Syllable* level pair in our example) and work down to finer grained annotations (i.e., the *Syllable*/*Phoneme* and *Phoneme*/*Phonetic* level pairs in our example). To build hierachical annotation structures we need link definitions, which together with the level definitions define the annotation structure for the entire database (see Chapter @ref(chap:annot_struct_mod) for further details). The `autobuild_linkFromTimes()` calls in R Example @ref(rexample:tutorial_autobuild) use the `newLinkDefType` parameter, which if defined automatically adds a link definition to the database.

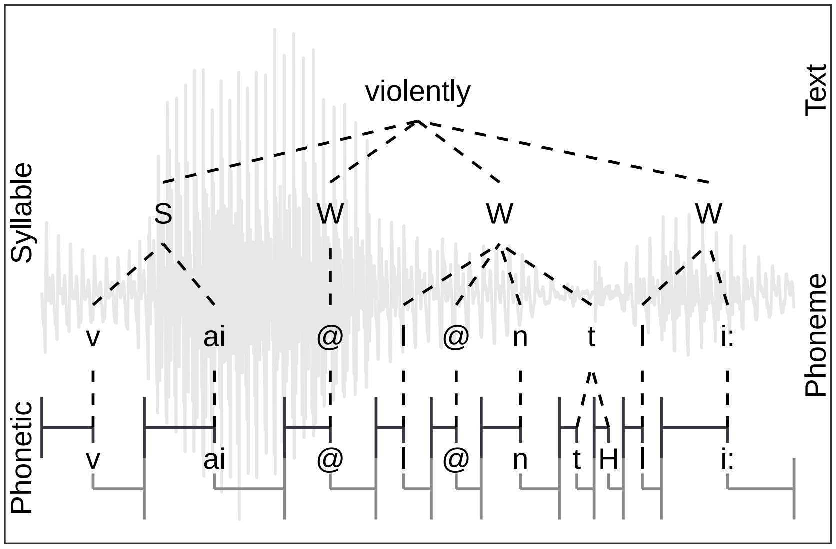


Figure 3.4: (#fig:tutorial_violentlyHier)Example of a hierarchical annotation of the word *violently* belonging to the *msajc012* bundle of the *myFirst* demo `emuDB`.

```r
# invoke autobuild function
# for "Text" and "Syllable" levels
autobuild_linkFromTimes(dbHandle,
                        superlevelName = "Text",
                        sublevelName = "Syllable",
                        convertSuperlevel = TRUE,
                        newLinkDefType = "ONE_TO_MANY")

# invoke autobuild function
# for "Syllable" and "Phoneme" levels
autobuild_linkFromTimes(dbHandle,
                        superlevelName = "Syllable",
                        sublevelName = "Phoneme",
                        convertSuperlevel = TRUE,
                        newLinkDefType = "ONE_TO_MANY")

# invoke autobuild function
# for "Phoneme" and "Phonetic" levels
autobuild_linkFromTimes(dbHandle,
                        superlevelName = "Phoneme",
                        sublevelName = "Phonetic",
                        convertSuperlevel = TRUE,
                        newLinkDefType = "MANY_TO_MANY")
```

As the `autobuild_linkFromTimes()` function automatically creates backup levels to avoid the accidental loss of boundary or event time information, R Example @ref(rexample:tutorial_delBackupLevels) shows how these backup levels can be removed to clean up the database. However, using the `remove_levelDefinition()` function with its `force` parameter set to `TRUE` is a very invasive action. Usually this would not be recommended, but for this tutorial we are keeping everything as clean as possible.

```r
# list level definitions
# as this reveals the "-autobuildBackup" levels
```

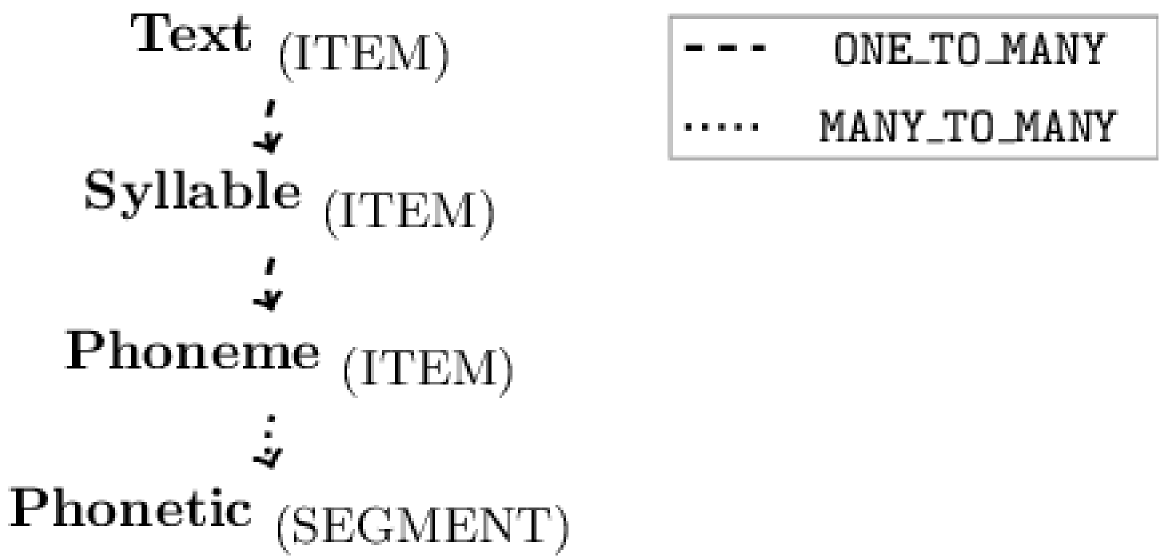


Figure 3.5: Schematic annotation structure of the emuDB after calling the autobuild function in R Example @ref(rexample:tutorial_autobuild).(#fig:tutorial_simpleAnnotStruct)

```r
# added by the autobuild_linkFromTimes() calls
list_levelDefinitions(dbHandle)
```

```
##                       name    type nrOfAttrDefs              attrDefNames
## 1                     Text    ITEM            1                     Text;
## 2                 Syllable    ITEM            1                 Syllable;
## 3                  Phoneme    ITEM            1                  Phoneme;
## 4                 Phonetic SEGMENT            1                 Phonetic;
## 5     Text-autobuildBackup SEGMENT            1     Text-autobuildBackup;
## 6 Syllable-autobuildBackup SEGMENT            1 Syllable-autobuildBackup;
## 7  Phoneme-autobuildBackup SEGMENT            1  Phoneme-autobuildBackup;
```

```r
# remove the levels containing the "-autobuildBackup"
# suffix
remove_levelDefinition(dbHandle,
                       name = "Text-autobuildBackup",
                       force = TRUE,
                       verbose = FALSE)

remove_levelDefinition(dbHandle,
                       name = "Syllable-autobuildBackup",
                       force = TRUE,
                       verbose = FALSE)

remove_levelDefinition(dbHandle,
                       name = "Phoneme-autobuildBackup",
                       force = TRUE,
                       verbose = FALSE)

# list level definitions
list_levelDefinitions(dbHandle)
```

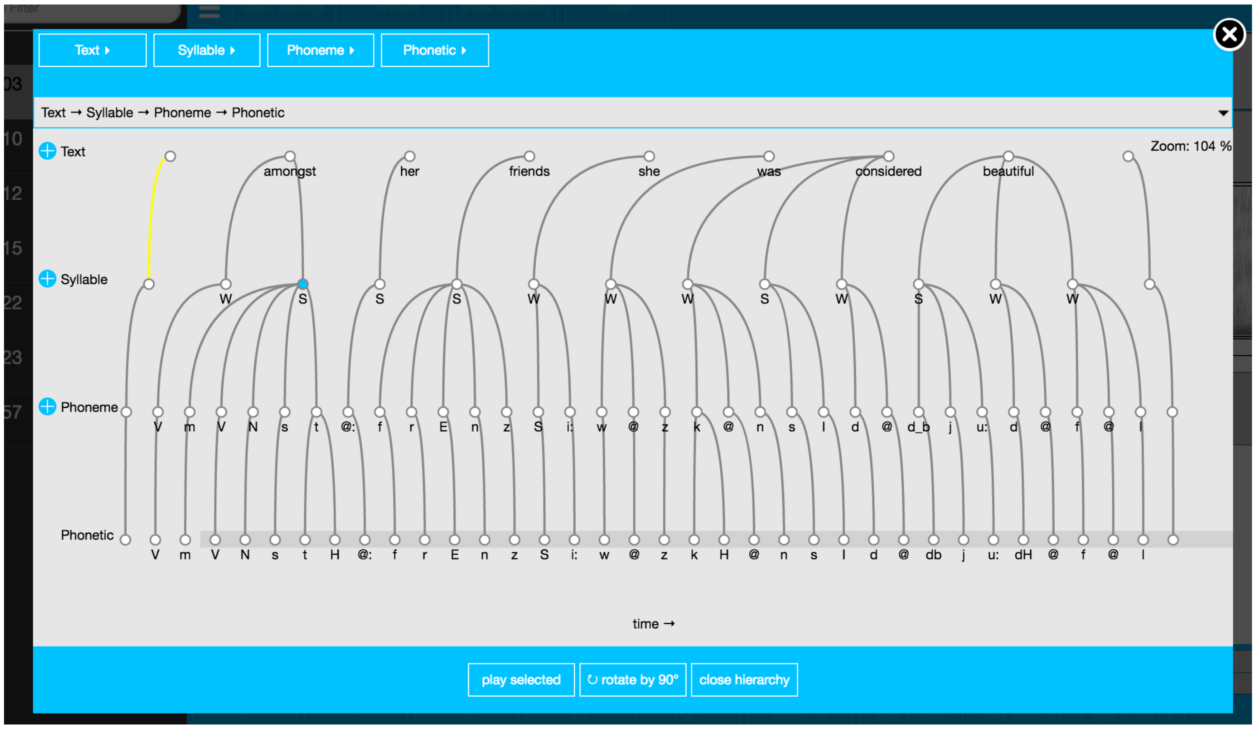


Figure 3.6: Screenshot of `EMU-webApp` displaying the autobuilt hierarchy of the *myFirst* `emuDB`.(#fig:tutorial_EMU-webAppScreenshotTutorialPostAutobHier)

```
##       name    type nrOfAttrDefs attrDefNames
## 1     Text    ITEM            1        Text;
## 2 Syllable    ITEM            1    Syllable;
## 3  Phoneme    ITEM            1     Phoneme;
## 4 Phonetic SEGMENT            1    Phonetic;
```

```
# list level definitions
# which were added by the autobuild functions
list_linkDefinitions(dbHandle)
```

```
##          type superlevelName sublevelName
## 1  ONE_TO_MANY           Text     Syllable
## 2  ONE_TO_MANY       Syllable      Phoneme
## 3 MANY_TO_MANY        Phoneme     Phonetic
```

As can be seen by the output of `list_levelDefinitions()` and \ `list_linkDefinitions()` in R Example @ref(rexample:tutorial_autobuild), the annotation structure of the *myFirst* `emuDB` now matches that displayed in Figure @ref(fig:tutorial_simpleAnnotStruct). Using the `serve()` function to open the `emuDB` in the `EMU-webApp` followed by clicking on the `show hierarchy` button in the top menu (and rotating the hierarchy by 90 degrees by clicking the `rotate by 90 degrees` button) will result in a view similar to the screenshot of Figure @ref(fig:tutorial_EMU-webAppScreenshotTutorialPostAutobHier).

3.3.2 Querying the hierarchical annotations

Having this hierarchical annotation structure now allows us to formulate a query that helps answer the originally stated question: *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?*. To keep things simple, here we will focus only on the vowels *i:*, *o:* and *V* (SAMPA annotation TODO: cite + mention SAMPA annotation above). R Example @ref(rexample:tutorial_labelGroupQuery) shows how all the vowels in the *myFirst* database are queried.

```r
# query the label group on the Phonetic level
sl_vowels = query(dbHandle, "Phonetic == i: | o: | V ")

# show first entry of sl
head(sl_vowels, n = 1)
```

```
## segment  list from database:  myFirst
## query was:  Phonetic == i: | o: | V
##   labels   start     end session   bundle    level    type
## 1      V 187.425 256.925    0000 msajc003 Phonetic SEGMENT
```

As the type of syllable (strong vs. week) for each vowel that was just extracted is also needed, we can use the requery functionality of the EMU-SDMS (see Chapter **??**) to retrieve the syllable type for each vowel. A requery essentially moves through a hierarchical annotation (vertically or horizontally) starting from the segments that are passed into the requery function. R Example @ref(rexample:tutorial_requery) illustrates the usage of the hierarchical requery function, `requery_hier()`, to retrieve the appropriate annotation items from the *Syllable* level.

```r
# hierarchical requery starting from the items in sl_vowels
# and moving up to the "Syllable" level
sl_sylType = requery_hier(dbHandle,
                          seglist = sl_vowels,
                          level = "Syllable",
                          calcTimes = FALSE)

# show first entry of sl
head(sl_sylType, n = 1)
```

```
## segment  list from database:  myFirst
## query was:  FROM REQUERY
##   labels start end session   bundle    level type
## 1      W    NA  NA    0000 msajc003 Syllable ITEM
```

```r
# show that sl_vowel and sl_sylType have the
# same number of row entries
nrow(sl_vowels) == nrow(sl_sylType)
```

```
## [1] TRUE
```

As can be seen by the `nrow()` comparison in R Example @ref(rexample:tutorial_requery), the segment list returned by the `requery_hier()` function has the same number of rows as the original `sl_vowels` segment list. This is important, as each row of both segment lists line up and allow us to infer which segment belongs to which syllable (e.g., vowel `sl_vowels[5,]` belongs to syllable `sl_sylType[5,]`).

3.4 Signal extraction and exploration

Now that the vowel and syllable type information including the vowel start and end time information has been extracted from the database, this information can be used to extract signal data that matches these segments. Using the `emuDB` function `get_trackdata()` we can calculate the formant values in real time using the formant estimation function, `forest()`, provided by the `wrassp` package (see Chapter **??** for details). R Example @ref(rexample:tutorial_getTrackdata) shows the usage of this function.

```r
# get formant values for
td_vowels = get_trackdata(dbHandle,
                          seglist = sl_vowels,
                          onTheFlyFunctionName = "forest",
                          verbose = F)

# show class vector
class(td_vowels)
```

```
## [1] "trackdata"
```

```r
# show dimensions
dim(td_vowels)
```

```
## [1] 12  4
```

```r
# display all values for fifth segment
td_vowels[5,]
```

```
## trackdata from track: fm
## index:
##  left right
##     1    27
## ftime:
##       start    end
## [1,] 1187.5 1317.5
## data:
##         T1  T2   T3   T4
## 1187.5 244 928 1827 3417
## 1192.5 277 903 1832 3402
## 1197.5 306 884 1851 3215
## 1202.5 329 862 1873 3194
## 1207.5 337 833 1897 3209
## 1212.5 340 816 1913 3269
## 1217.5 343 798 1931 3310
## 1222.5 345 768 1942 3300
## 1227.5 349 745 1936 3019
## 1232.5 357 745 1930 3227
## 1237.5 364 750 1931 3182
## 1242.5 371 760 1934 3271
## 1247.5 377 779 1941 3327
## 1252.5 380 801 1946 3339
## 1257.5 381 819 1953 3346
## 1262.5 382 832 1961 3354
## 1267.5 382 841 1973 2900
## 1272.5 380 849 1993 2936
## 1277.5 379 855 2016 3390
## 1282.5 375 860 2031 3076
## 1287.5 369 865 2054 3166
## 1292.5 362 880 2079 3271
## 1297.5 350 909 2120 3325
## 1302.5 339 930 2179 3357
## 1307.5 323 940 2198 3354
## 1312.5 301 940 2180 3300
## 1317.5 277 950 2228 3295
```

As can be seen by the call to the `class()` function, the resulting object is of the type `trackdata` and has 12 entries. This corresponds to the number of rows contained in the segment lists extracted above (i.e., `nrow(sl_vowels)`). This indicates that this object contains data for each of the segments that correspond to each of the row entries of the segment lists (i.e., `td_vowels[5,]` are the formant values belonging to `sl_vowels[5,]`). As the columns `T1`, `T2`, `T3`, `T4` of the printed output of `td_vowels[5,]` suggest, the `forest` function estimates four formant values. We will only be concerned with the first (column `T1`) and second (column `T2`). R Example @ref(rexample:tutorial_dplot) shows a call to `emuDB`'s `dplot()` function which produces the plot displayed in Figure @ref(fig:tutorial_dplot). The first call to the `dplot()` function plots all 12 first formant trajectories (achieved by indexing the first column i.e., `T1`: `x = td_vowels[, 1]`). The 12 trajectories are color coded by vowel label (`labs = sl_vowels\$labels`). To clean up the cluttered left plot, the second call to the `dplot()` function additionally uses the `average` parameter to plot only the ensemble averages of each vowel label and time-normalizes the trajectories (`normalise = TRUE`) to an interval between 0 and 1.

```r
# two plots next to each other
par(mfrow = c(1,2))

dplot(x = td_vowels[, 1],
      labs = sl_vowels$labels,
      xlab = "Duration (ms)",
      ylab = "F1 (Hz)")

dplot(x = td_vowels[, 1],
      labs = sl_vowels$labels,
      normalise = TRUE,
      average = TRUE,
      xlab = "Normalized time",
      ylab = "F1 (Hz)")

# back to single plot
par(mfrow = c(1,1))
```

Figure @ref(fig:tutorial_dplot) gives an overview of the first formant trajectories by vowel class. For the purpose of data exploration and to get an idea of where the individual vowel classes lie on the F2 x F1 plane, which indirectly provides information about vowel height and tongue position, R Example @ref(rexample:tutorial_eplot) makes use of the `eplot()` function. This produces Figure @ref(fig:tutorial_eplot). To be able to use the `eplot()` function, the `td_vowels` object first has to be modified, as it contains entire formant trajectories but two dimensional data is needed to be able to display it on the F2 x F1 plain. This can, for example, be achieved by only extracting temporal mid-point formant values for each vowel using the `get_trackdata()` function utilizing its `cut` parameter. R Example @ref(rexample:tutorial_eplot) shows an alternative approach using the `dcut()` function to essentially cut the formant trajectories to a specified proportional segment. By using only the `left.time = 0.5` (and not specifying `right.time`) only the formant values that are closest to the temporal mid-point are cut from the trajectories.

```r
# cut formant trajectories at temporal mid-point
td_vowels_midpoint = dcut(td_vowels,
                          left.time = 0.5,
                          prop = TRUE)

# show dimensions of td_vowels_midpoint
dim(td_vowels_midpoint)

# generate plot
eplot(x = td_vowels_midpoint[,1:2],
```

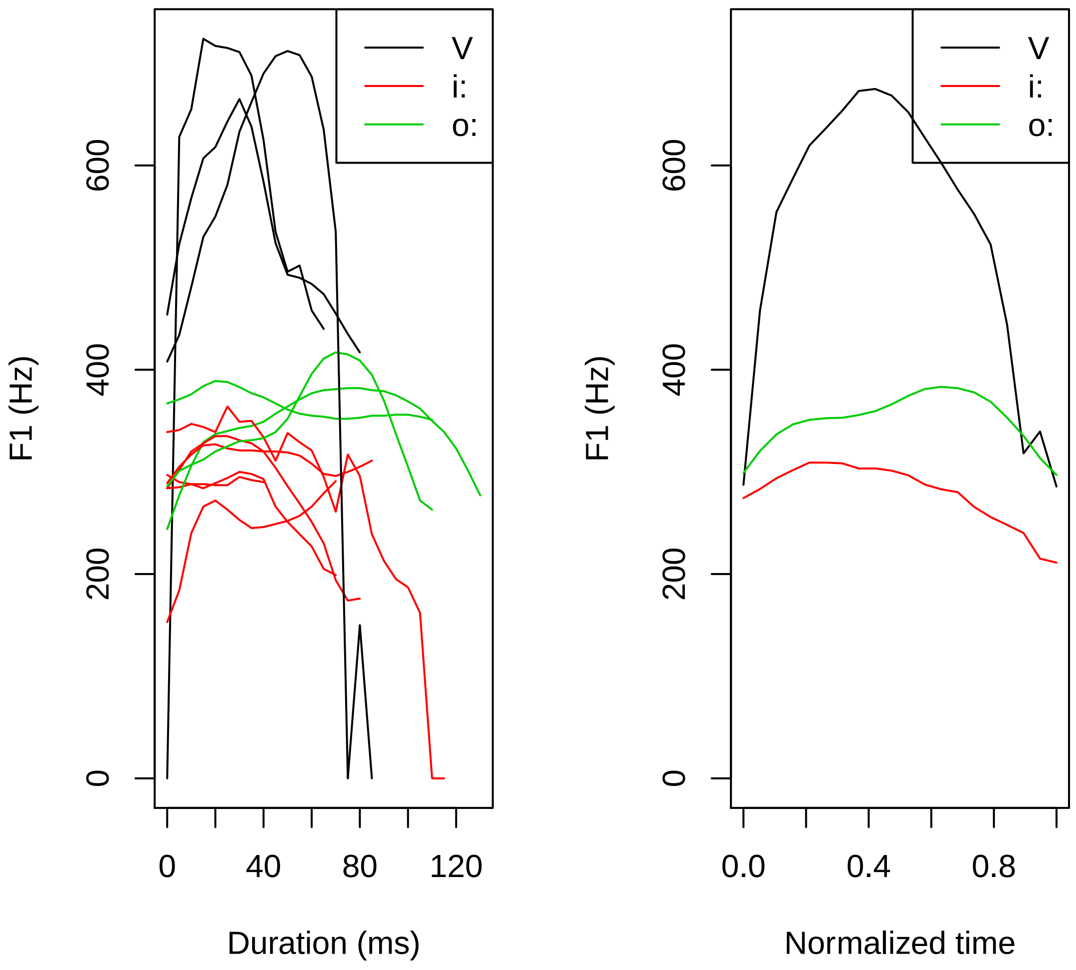


Figure 3.7: (#fig:tutorial_dplot)`dplot()` plots of F1 trajectories. The left plot displays 81 trajectories while the right plot displays ensemble averages of each vowel.

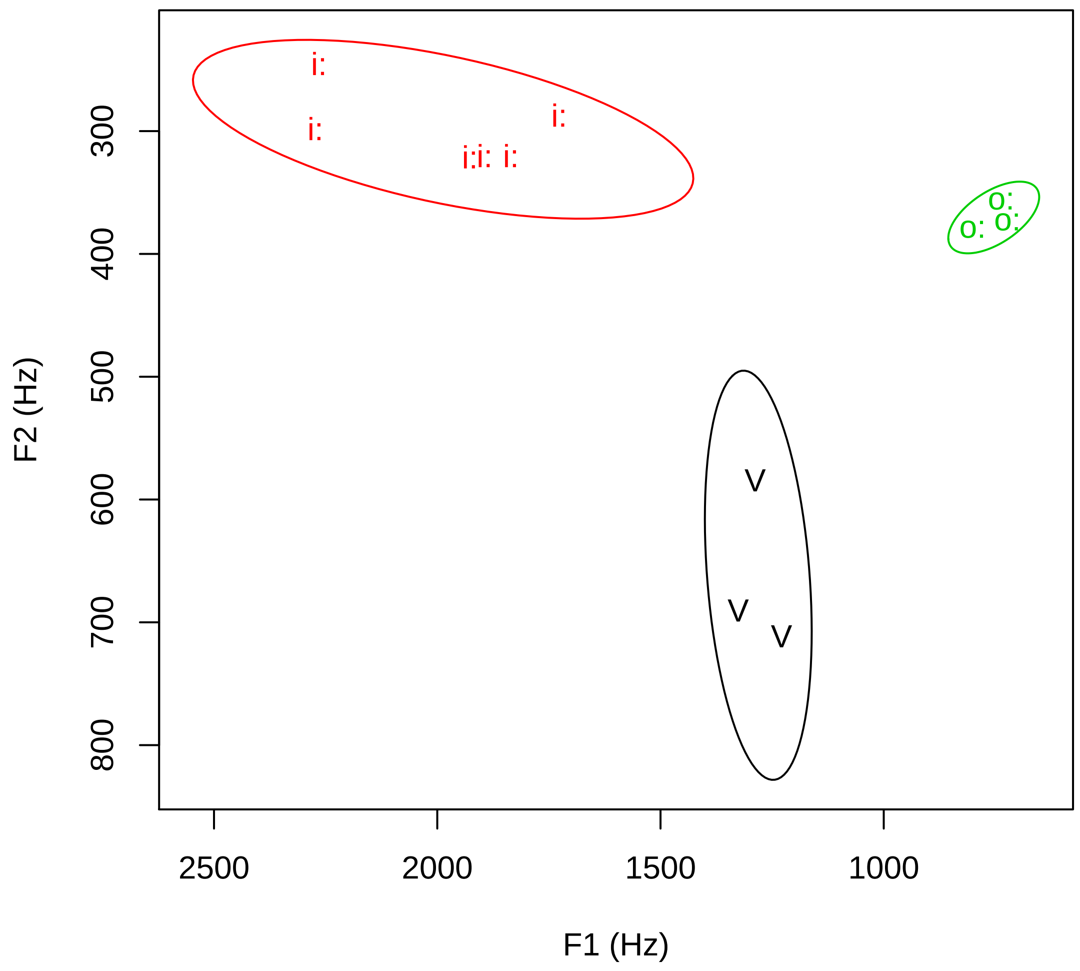


Figure 3.8: (#fig:tutorial_eplot)95% ellipses for F2 x F1 data extracted from the temporal midpoint separated by vowel.

```
      labs = sl_vowels$labels,
      dopoints = TRUE,
      formant = T,
      xlab="F1 (Hz)",
      ylab = "F2 (Hz)")
```

Figure @ref(fig:tutorial_eplot) displays the first two formants extracted at the temporal midpoint of every vowel in `sl_vowels`. These formants are plotted on the F2 x F1 plane, and their 95% ellipsis distributions are also shown. Although not necessarily applicable to the question posed at the beginning of this tutorial, the data exploration using the `dplot()` and `eplot()` functions can be very helpful tools for providing an overview of the data at hand.

3.5 Vowel height as a function of syllable types (strong vs. weak): evaluation and statistical analysis

The above data exploration only dealt with the actual vowels and disregarded the syllable type they occurred in. However, the question in the introduction of this chapter does not distinguish between vowel

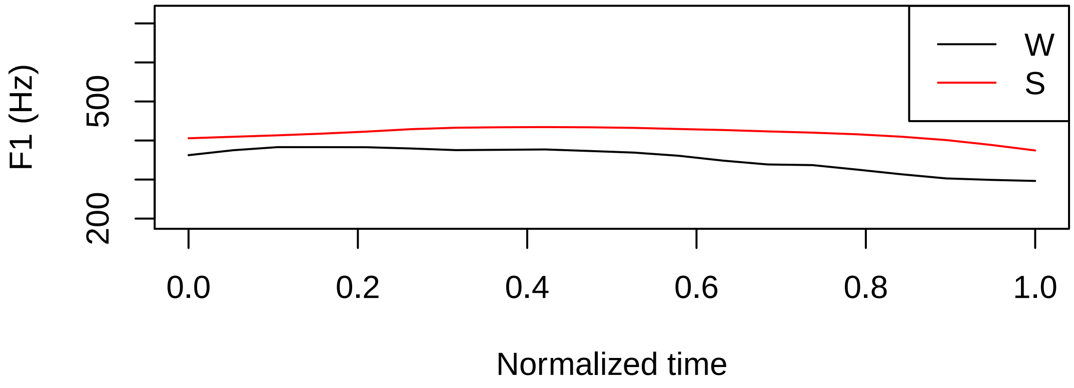


Figure 3.9: (#fig:tutorial_dplotSylTyp)Ensemble averages of F1 contours of all tokens of the central 60% of vowels grouped by syllable type (strong (S) vs. weak (W)).

classes but focuses on whether a vowel occurs in a strong or weak syllable. For the sake of this tutorial we will disregard the fact that the vowel class influences the syllable type and only focus on whether a vowel occurred in a syllable labeled *S* (strong) or *W* (weak). For data inspection purposes, R Example @ref(rexample:tutorial_dplotSylTyp) initially extracts the central 60% (`left.time = 0.2` and `right.time = 0.8`) of the formant trajectories from `td_vowels` using `dcut()` and displays them using `dplot()`. It should be noted that the call to `dplot()` uses the labels of the `sl_sylType` object as opposed to those of `sl_vowels`. This causes the `dplot()` functions to group the trajectories by their syllable type as opposed to their vowel class as displayed in Figure @ref(fig:tutorial_dplotSylTyp).

```r
# extract central 60% from formant trajectories
td_vowelsMidSec = dcut(td_vowels,
                       left.time = 0.2,
                       right.time = 0.8,
                       prop = TRUE)

# plot first formant trajectories
dplot(x = td_vowelsMidSec[, 1],
      labs = sl_sylType$labels,
      normalise = TRUE,
      average = TRUE,
      xlab = "Normalized time",
      ylab = "F1 (Hz)")
```

As can be seen in Figure @ref(fig:tutorial_dplotSylTyp), there seems to be a distinction in F1 trajectory height between vowels in strong syllables and weak syllables. R Example @ref(rexample:tutorial_boxplot) shows the code to produce a boxplot using the `ggplot2` package to further visually inspect the data (see Figure @ref(fig:tutorial_boxplot) for the plot produced by R Example @ref(rexample:tutorial_boxplot)).

```r
# use trapply to calculate the means of the 60%
# formant trajectories
td_vowelsMidSec_mean = trapply(td_vowelsMidSec[, 1],
                               fun = mean,
                               simplify = T)

# create new data frame that contains the mean
# values and the corresponding labels
df = data.frame(sylType = sl_sylType$labels,
                meanF1 = td_vowelsMidSec_mean)
```

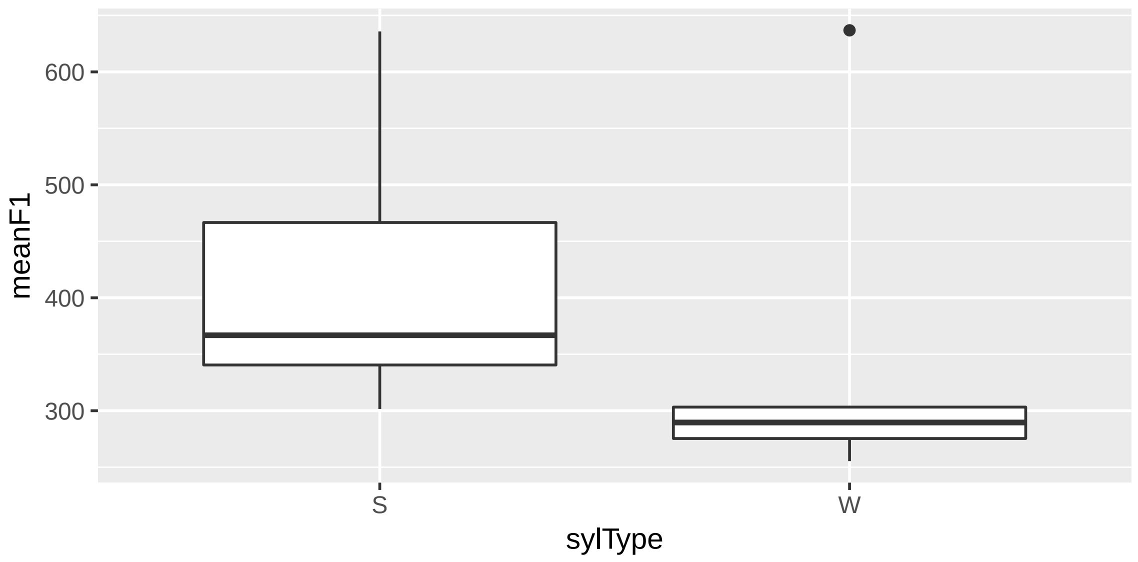


Figure 3.10: (#fig:tutorial_boxplot)Boxplot produced using `ggplot2` to visualize the difference in F1 depending on whether the vowel occurs in strong (S) or weak (W) syllables.

```r
# load library
library(ggplot2)

# create boxplot using ggplot
ggplot(df,
       aes(sylType, meanF1)) + geom_boxplot()
```

3.6 Conclusion

The tutorial given in this chapter gave an overview of what it is like working with the EMU-SDMS to try to solve a research question. As many of the concepts were only briefly explained, it is worth noting that explicit explanations of the various components and integral concepts are given in following chapters. Further, additional use cases that have been taken from the `emuR_intro` vignette can be found in Appendix @ref(app_chap:useCases). These use cases act as templates for various types of research questions and will hopefully aid the user in finding a solution similar to what she or he wishes to achieve.

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.

```r
# 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

```r
summary(dbHandle)
```

or

```r
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).

```r
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.

Draxler, C. and Jänsch, K. (2004). SpeechRecorder - a Universal Platform Independent Multi-Channel Audio Recording Software. In *Proc. of the IV. International Conference on Language Resources and Evaluation*, pages 559–562, Lisbon, Portugal.

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.

Kisler, T., Schiel, F., Reichel, U. D., and Draxler, C. (2015). Phonetic/linguistic web services at bas. ISCA.

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.